The Intersection of

Planning and Learning

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op woensdag 10 maart 2021 om 15:00
uur

door

Thomas Marinus MOERLAND

Master of Science in Mathematics, Universiteit Leiden, Nederland,
Arts, Universiteit Leiden, Nederland,

geboren te Amsterdam, Nederland.

Dit proefschrift is goedgekeurd door de

promotor: Prof. dr. C.M. Jonker
promotor: Prof. dr. A. Plaat
copromotor: Dr. ir. J. Broekens

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. C.M. Jonker,	Technische Universiteit Delft
Prof. dr. A. Plaat,	Universiteit Leiden
Dr. ir. J. Broekens,	Universiteit Leiden
Onafhankelijke leden:	
Prof. dr. R. Babuska,	Technische Universiteit Delft
Dr. M.T.J. Spaan,	Technische Universiteit Delft
Prof. dr. P. Oudeyer,	INRIA, Bordeaux, Frankrijk
Dr. J.B. Hamrick,	Deepmind, Londen, Verenigd Koninkrijk
Prof. dr. G.C.H.E. de Croon,	Technische Universiteit Delft, reservelid

Keywords: Model-based reinforcement learning, planning, sequential decision making, Markov Decision Process.

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

CONTENTS

ABSTRACT

Intelligent sequential decision making is a key challenge in artificial intelligence. The problem, commonly formalized as a Markov Decision Process, is studied in two different research communities: *planning* and *reinforcement learning*. Departing from a fundamentally different assumption about the type of access to the environment, both research fields have developed their own solution approaches and conventions. The combination of both fields, known as *model-based reinforcement learning*, has recently shown state-of-the-art results, for example defeating human experts in classic board games like Chess and Go. Nevertheless, literature lacks an integrated view on 1) the similarities between planning and learning, and 2) the possible combinations of both. This dissertation aims to fill this gap. The first half of the book presents a conceptual answer to both questions. We first present a framework that disentangles the common algorithmic space of both fields, showing that they essentially make the same decisions. Moreover, we also present an overview on how to combine planning and learning in one algorithm. The second half of the dissertation provides experimental illustration of these ideas. We present several new combinations of planning and learning, such as a flexible method to learn stochastic dynamics models with neural networks, an extension of a successful planning-learning algorithm (AlphaZero) to deal with continuous action spaces, and a study of the empirical trade-off between planning and learning. Finally, we also illustrate the commonalities between both fields, by designing a new algorithm in one field based on inspiration from the other field. We conclude the thesis with an outlook for the planning-learning field as a whole. Altogether, the dissertation provides a broad theoretical and empirical view on the combination of planning and learning, which promises to be an important frontier in artificial intelligence research in the coming years.

Part I

PLANNING AND LEARNING

1 INTRODUCTION

> "It's like this," Winnie-the-Pooh said.
> "When you go after honey with a balloon, the great thing is not to let the bees know you're coming. Now, if you have a green balloon, they might think you were only part of the tree, and not notice you, and if you have a blue balloon, they might think you were only part of the sky, and not notice you, and the question is: Which is most likely?"
> "Wouldn't they notice you underneath the balloon?" Christopher Robin asked.
> "They might or they might not," said Winnie-the-Pooh. "You never can tell with bees."
>
> A.A. Milne, *Winnie-the-Pooh and Some Bees* (1926)

Intelligent sequential decision-making is a key challenge in artificial intelligence research. While this dissertation will focus on the integration of two successful approaches to this problem, *planning* and *reinforcement learning*, we will start with an informal illustration of the problem type. Intelligent sequential decision making is crucial in nearly every aspect of our daily lives: getting groceries, navigating to your work, playing sports, etc. The concepts of intelligent sequential decision making appear so natural to us that we hardly notice them. We will therefore illustrate the general problem with a small piece of text from one of the Winnie-the-Pooh stories by A.A. Milne, shown in the epigraph above this chapter. It is a story a young child would understand, and actually enjoy.

In the story, Pooh clearly does not act randomly, but acts to be *rewarded*, i.e., he has goals. Some situations in the environment, like honey, are positively rewarding, while others, like being stung by the bees, have a negative reward. Importantly, Pooh does not only bother about instant reward, but rather about *cumulative reward*: he wants to achieve as much positive outcomes in the future (honey) while avoiding negative outcomes (being stung). The cumulative reward that we expect to achieve in the future is commonly referred to as the *value*.

Pooh clearly also makes observations about the environment: he has spotted the honey in the tree, but also observes that he has not got the

honey himself yet. Observations define the *state* of the environment. Pooh is also able to identify a possible state of the environment, himself holding the honey, which he would like to reach. Moreover, Pooh realizes that he can *act*: he can influence the environment, for example by taking either a green or a blue balloon. Depending on these actions, the state of the environment may change.

In order to reach the rewarding honey, Pooh is clearly anticipating in his mind what may happen. If he takes the green balloon, then the bees may think he is part of the tree, while a blue balloon may make him seem part of the sky. Such explicit anticipation of different futures is better known as *planning*. Planning can improve our decision, as it separates out different futures and their possible pay-off. In order to plan, we need to have a *model* of the environment. When we take an action in a certain state, we want to be able to predict the relevant change in the environment, and the associated reward. Pooh did not have such a model of honey, bees and balloon types when he was born: clearly, he has *learned* this model from previous experience in his lifetime, i.e., from data.

Pooh also indicates that he is not certain about the outcome of an action. When he takes the green balloon, he does not know whether the bees will accept the balloon as a tree, nor whether they will see him below the balloon. To Pooh, the environment is *stochastic*: when he chooses an action, multiple futures may present, according to some probability distribution. Such stochasticity is present in most real-world tasks. Also note that Pooh implicitly takes the expectation (average) of the cumulative reward of the possible futures: he wants to select the balloon which on average gives the highest chance of honey with the lowest chance of detection.

Apart from stochasticity, Pooh may have a second reason to be uncertain. He may not have had enough experience in his life to be able to accurately predict what will happen. In other words, he may be uncertain due to a lack of data, better known as statistical uncertainty. Christopher Robin illustrates this phenomenon, by expecting the bees to spot Pooh below the balloon. Clearly, Christopher Robin has learned a different model than Pooh, likely because he has seen more or different data.

Finally, Pooh has not only learned a model, but also learned *action preferences* or *action values*. For example, he only considers two possible actions in his plan, while the total number of possible actions should be much larger. He does not consider to shout, write or dance to obtain the honey, which he must have learned from previous experience. He

clearly directs his plan based on previously learned action preferences. Thereby, Pooh has been learning in two ways: 1) to learn a model that predicts how the environment responds to his actions, and 2) to learn the preference (or value) of actions, based on the total sum of rewards that each action will obtain in the future.

As mentioned before, we usually do not notice these aspects when we read the above story. The concepts are probably so natural because this is what we do all day: attempting to make intelligent sequential decisions based on a cumulative reward criterion. When we do dig deeper we quickly identify that Pooh shows signs of both planning and learning in his behaviour. This dissertation will focus on these two approaches, their similarities, and the ways to combine both. The next section will briefly introduce both fields.

1.1 PLANNING AND REINFORCEMENT LEARNING

The computational study of cumulative reward optimization is commonly formalized as a *Markov Decision process* (MDP) (Puterman, 2014). Especially the partially observable MDP (POMDP) specification has great flexibility, and can deal with any decision-making problem in which we have some sensory observations (states[1]), can take certain actions, and desire to maximize some cumulative performance measure over time (cumulative reward). It is therefore a generic specification for any type of sequential decision making problem, with applications in, for example, robotics (Kober, Bagnell, and Peters, 2013), autonomous driving (Shalev-Shwartz, Shammah, and Shashua, 2016), and game playing (Silver et al., 2016, 2017c).

Several research fields have studied the MDP optimization problem. The two dominant approaches are *reinforcement learning* (RL) (Barto, Sutton, and Anderson, 1983; Sutton and Barto, 2018), a subfield of

1 The meaning of the term 'state' varies between research fields. In robotics and control, state refers to the most compact representation of the problem. For example, a high-dimensional image of a robot would be considered an *observation*, while the state of the system only consists of the underlying task relevant features. In contrast, in computer science researchers would typically refer to the entire high-dimensional image as state, and define the MDP over the entire image space. This distinction does not alter the underlying problem, but is just a matter of terminology. Throughout this thesis, we choose to follow the computer science definition of state. Note that this discussion applies to fully observable problems: in case of partial observability we always need to distinguish the non-Markovian observations from the underlying Markovian state, although even in these cases terminology can be sloppy (i.e., researchers define a POMDP problem as a MDP, but still incorporate methodology to deal with the partial observability in their algorithm, assuming the reader understands it is really a POMDP).

machine learning, and *planning*, whose discrete and continuous versions are studied in symbolic artificial intelligence[2] (Moore, 1959; Russell and Norvig, 2016) and control (Bellman, 1966; Bertsekas, 1995) research, respectively.

Note that in the broader AI community, the term 'planning' is used for other types of problems as well, like scheduling a group of trucks for a company (a combinatorial optimization problem, see Korte et al. (2012)), or planning over logical representations (Saffiotti, Konolige, and Ruspini, 1995). Though we could reformalize these problems into MDPs, we will not consider them in this book. When we refer to 'planning' throughout the book, we actually refer to *MDP* planning methods without any additional (logical) structure on the state space.[3]

The initial distinction between planning and learning originates from the way they can access the MDP transition dynamics. In planning, we have *reversible* access to the MDP dynamics, which we call a *model*.

A model is a form of reversible access to the MDP dynamics (known or learned).

A model allows us to repeatedly move forward from the same state, or move forward from any arbitrary state in the problem. This bears an analogy with how humans plan in their mind, repeatedly considering different action sequences. In contrast, reinforcement learning originally assumed that our access to the MDP dynamics is *irreversible*, and everytime we take an action we have to continue from the state we reach. This bears an analogy with the real world: whenever we try something, the effect is permanent. For some tasks, like board games, it is trivial to obtain a model, while for many real world tasks, like autonomous

2 Machine learning is of course part of artificial intelligence as a whole. But discrete planning has been traditionally studied in the symbolic artificial intelligence community, for which Haugeland (1989) introduced the term 'good old-fashioned AI' (GOFAI).

3 More precisely, this does include 'probabilistic planning' methods, since we do include stochastic MDP problems. However, we for example exclude specific logical description languages, like the 'planning domain definition language' (PDDL) (Ghallab et al., 1998) or its probabilistic extension PPDDL (Younes and Littman, 2004). These methods utilize additional structure on the state space and dynamics model, based on propositional logic. Although these approaches can plan more efficiently when the problem permits it, they do require prior knowledge about the logical structure of the state space and dynamics model. In this thesis, we focus on the generic approach that does not impose such additional structure. This has also been the main focus in the reinforcement learning community, although relational/logical representations have also received attention in this community (Garnelo, Arulkumaran, and Shanahan, 2016; Tadepalli, Givan, and Driessens, 2004; Van Otterlo, 2005).

driving, we do not known the dynamics model in advance and can only try in the real world.

However, we may also separate planning and reinforcement learning based on the way they represent the solution. Planning methods use *local* solutions, which focus on a particular state or subset of states, and are discarded after the solution gets executed. In contrast, reinforcement learning has focused on estimating *global* solutions for all possible states. Since under the original assumption (irreversible environment) RL agents cannot repeatedly simulate forward from the same state, our best bet is to estimate a global solution.

The two separate definitions of planning and reinforcement learning (reversible versus irreversible access to the dynamics, and local versus global solution representation) are unfortunately not consistent with eachother. For example, AlphaZero (Silver et al., 2018) has reversible access to the MDP dynamics (which would make it planning), but also learns global value and policy functions (which would make it reinforcement learning). We consider AlphaZero to be a model-based reinforcement learning algorithm (i.e., belong to reinforcement learning), and therefore chose to let the local versus global distinction dominate. This leads us to the following definitions of planning and reinforcement learning.

Planning is a class of MDP algorithms that 1) use a model and 2) store a local problem solution.

Reinforcement learning is a class of MDP algorithms that store a global solution.[4]

Note that the above definitions exclude the combination of irreversible access to the MDP dynamics (unknown model) and a location solution. Indeed, this combination does not make sense. The moment we would start building a local solution (after the first tried action), we cannot get back anymore, and therefore directly have to discard the local solution. A thorough introduction of MDPs, planning, reinforcement learning is provided in Chapter 2.

4 Sutton and Barto (2018) define reinforcement learning as 'learning what to do - how to map situations to actions - so as to maximize a numerical reward signal'. This definition however coincides with the MDP definition, and would not discriminate it from planning. The definition by Sutton and Barto is mostly intended to separate RL from supervised learning, in which case the agent would be exactly told what actions to take, instead of getting the partial information provided by rewards.

Table 1.1: Distinction between planning, reinforcement learning and model-based reinforcement learning. The cell entries +, - and +/- indicate a property is present, absent, or not defining, respectively.

	Model	**Local solution**	**Global solution**
Planning	+	+	-
Reinforcement learning	+/-	+/-	+
Model-free reinforcement learning	-	-	+
Model-based reinforcement learning	+	+/-	+

LEARNING We have already used the term 'learning' a few times, but it is actually not trivial to define. In psychology literature, learning can for example refer to non-associative learning (Peeke, 2012), like habituation and sensitization, or associative learning (Mackintosh, 1983), like classical conditioning (Pavlov and Gantt, 1928), instrumental conditioning (Skinner, 1937), observational learning (Miller and Dollard, 1941) and imprinting (Hess, 1959). In this thesis, and in the context of sequential decision making, we explicitly focus on reward- or goal-based learning, in line with instrumental conditioning.

As already defined above, in the context of MDP optimization we consider learning to be 'the optimization process towards a global solution'. This definition is mostly inspired by the distinction with planning, which focuses on local solutions. However, it is also in line with the ideas on reactive behaviour and (instrumental) conditioning. Learned skills become reactive (fast), and such reactive behaviour require a global solution (i.e., we can not reactively respond when we first have to build a local solution).

In artificial intelligence, learning is mostly studied in the machine learning community. There, learning is often associated with another property: generalization. For a function that maps some input (e.g, observations) to output (e.g., actions), generalization implies that similar input usually also leads to approximately similar output. Indeed, in de context of MDPs, generalization is associated with (non-tabular) reinforcement learning, and often considered as one of its defining characteristics and benefits. Generalization allows us to 1) make predictions for unobserved/similar states, and 2) store a solution in memory in approximate form (which is inevitable in larger problems). In contrast, planning has by convention focused on local, exact solution representations, which do not generalize.

We therefore again encounter two possible definitions, since learning can be distinguished by 1) a global solution, or 2) a solution that generalizes.[5] Unfortunately, these definitions do not agree in the case of a tabular/atomic/exact representation. The first definition would allow learning on tables (they can be global), the second definition would not (they can not generalize). For example, a classic RL approach like tabular Q-learning (Watkins and Dayan, 1992) is usually considered learning (using the global solution definition), but (especially in recent years) reinforcement learning is often associated with the benefits of generalization (using the generalization definition), see, e.g., Ponsen, Taylor, and Tuyls (2009).

The different uses of learning are in practice not a huge problem, especially since readers understand the intended use in the context of the paper. Only after writing the research papers in this thesis we discovered our own inconsistency as well. For example, in Chapter 3 we show that common planning updates can actually be rewritten as a form of tabular learning updates, which uses the global solution definition of learning. However, in other chapters we may write that 'learning adds generalization to planning'. The latter statement should therefore technically be read as 'non-tabular learning methods add generalization to planning'.

1.2 MODEL-BASED REINFORCEMENT LEARNING

Having discussed the differences between planning and learning, we will now focus on the ways to combine both approaches. The main class of algorithms that integrates planning and learning is *model-based reinforcement learning* (Hester and Stone, 2012b; Sutton, 1990; Sutton and Barto, 2018), which we define as:

> *Model-based reinforcement learning is a class of MDP algorithms that 1) use a model, and 2) store a global problem solution.*[6]

5 One could also define learning as a group of algorithms whose performance improves with additional data. However, in the case of RL and planning, this would not settle the definition either. Of course, both RL and planning improve their performance with additional data. Otherwise, when we only define irreversible samples as data, then we effectively recover the irreversible versus reversible separation between RL and planning. Since we already excluded this definition in the previous section, we do not include it here as a third possible definition of learning.

6 Note that a 'local solution', as required for pure planning, is not included in the definition of model-based RL. The reason is that well-known model-based RL algorithms, like Dyna (Sutton, 1990), learn a reversible model from data, but then sample single transitions/-

Most of the empirical work in this book deals with the model-based RL setting. A general scheme for the possible connections in model-based RL algorithms is shown in Figure 1.1, bottom. We have three boxes: one for planning, one for model learning, and one for policy or value learning. The boxes can be connected in various ways, which are labeled with letters a-g in the figure. The figure caption explains each of these connections, which we further detail in Chapter 4.

Figure 1.1 also illustrates the difference between the model-based reinforcement learning and the individual research fields of planning and model-free reinforcement learning. The top-left of the figure shows the arrows used by planning, where we plan over a model (arrow a) and use the results to act in the environment (arrow d). The top-right of the figure shows model-free RL, where we act in the environment based on a learned policy or value function (arrow e), and use the acquired data to update the value or policy approximation (arrow f). This book focuses on the integration of both fields, shown in the bottom of the figure. This may in principle use any of the arrows a-g, as we will see throughout this dissertation.

Model-based RL has shown impressive data efficiency results in recent years (Deisenroth and Rasmussen, 2011; Levine and Koltun, 2013). Moreover, it recently surpassed human expert performance in two-player board games like Chess, Go and Shogi (Silver et al., 2018, 2017c), while planning-inspired algorithms were also successful in Atari 2600 (Schrittwieser et al., 2019), another well-known AI testbed. As such, model-based RL has established itself as an important frontier in AI research.

1.3 RESEARCH QUESTIONS

Although model-based RL has been studied for at least three decades (Sutton, 1990), and has shown important empirical success (Schrittwieser et al., 2019; Silver et al., 2017c), literature lacks a fundamental study of the relation between and the combination of both fields. We detail these two deficits below.

---

traces from this model to directly update the global solution. These methods therefore never build a local solution (we can not consider such a single sample value estimate a local solution, since otherwise even model-free RL methods would be planning). Second, note that 'planning over a learned model', without any global solution approximation, is not considered model-based RL in this definition. We further detail this distinction in Chapter 4.

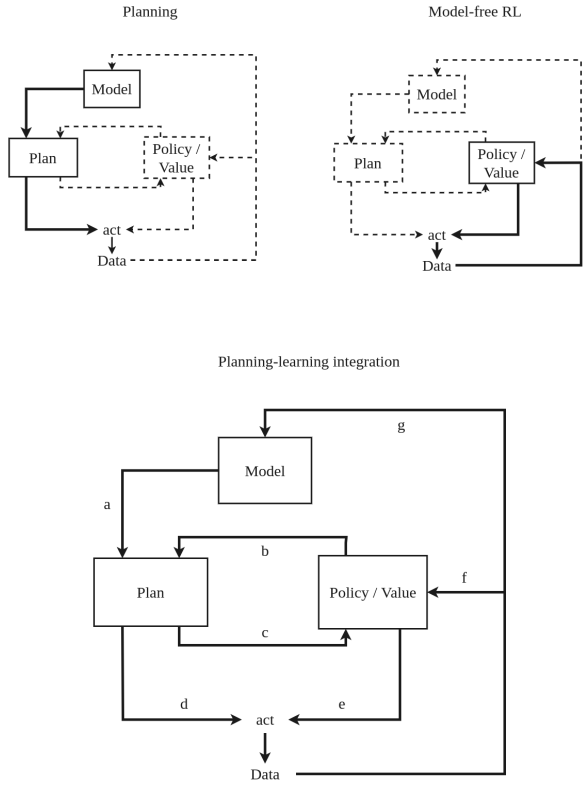


Figure 1.1: Conceptual illustration of the possible interactions between a model, a planning procedure, and policy/value value learning. **Top**: Illustration of the separate research fields of planning (left) and model-free reinforcement learning (right). Bold lines indicate which connections are actually used. **Bottom**: Illustration of planning-learning integration. All connections (identified with a-g) *can* be used. Labels: a) Planning over a model, b) Directing planning based on information in a learned policy/value function, c) Learning a policy/value function based on planning output, d) Acting in the real environment based on the planning output, e) Acting in the real environment based on a learned policy or value, f) Learning a policy/value based on real environment data, g) Learning a dynamics model from real environment data.

First of all, both planning and learning solve exactly the same MDP optimization problem. However, both research fields largely have their own research communities, and their methods are usually presented separately. For example, a classic AI textbook like Russell and Norvig (2016) devotes multiple chapters to both planning and reinforcement learning, but treats them as different topics, without discussion of their commonalities. Sutton and Barto (2018) did cover the relation between RL and planning, but only focuses on the type of back-up, which is only one aspect of planning and RL algorithms. This leaves a gap between both fields, as if they are fundamentally different. Since planning and RL deal with exactly the same problem, this dissertation hypothesizes that they actually share the same algorithmic space.

As a second deficit, literature also lacks a structured view on the way planning and learning can be combined. There is a plethora of model-based RL papers, which usually focus on a different subproblem of model-based RL, or introduce a variant of a known model-based RL approach. However, there is no literature that systematically structures the ways to combine planning and learning, and identifies which topics deserve more attention.

In short, planning and learning are two key fields in artificial intelligence, but we still lack a systematic bridge between both fields. This brings us to the following two research questions (each with a conceptual and empirical subquestion):

1. How are planning and learning related?
    - *Conceptual*: Do planning and learning share a common algorithmic space, and what does it consist of?
    - *Empirical*: How may we design a new algorithm in one field by taking inspiration from the other field?
2. How can planning and learning be combined?
    - *Conceptual*: How can we conceptually structure the space of algorithms that combine planning and learning?
    - *Empirical*: How may we design a new algorithm that combines planning and learning?

There appears to be a tension between these two research questions. The first question hypothesizes that both fields actually do the same thing, while the second question wants to combine both. The reader may ask: why would we want to combine two approaches that are actually the same? We will further detail this issue throughout the

Table 1.2: Thesis structure. We discuss two research quesions, each on a conceptual and empirical level. The thesis is structured in two halves: the first half (Ch. 3-4) provides a conceptual discussion, while the second half (Ch. 5-8) presents empirical illustration of both questions.

Research question	**Conceptual answer**	**Empirical answer**
How are planning and learning related?	Ch. 3	Ch. 8
How can planning and learning be combined?	Ch. 4	Ch. 5-7

book, but a short explanation is necessary here. The first question looks *within* a planning cycle, and *within* a reinforcement learning episode, identifying that their inner algorithmic decisions are essentially the same. In Figure 1.1, it hypothesizes that the algorithmic choices in the top-left and top-right graph are actually the same. The second research question looks *over* an entire planning cycle, emphasizing how planning may be integrated in a learning loop. For example, planning over a learned model may reduce the required number of samples in the real world, and the local tabular representation of planning may stabilize the global policy or value approximation of reinforcement learning. In Figure 1.1, the second research question therefore deals with all the arrows in the bottom graph, i.e., the possible connections to combine planning and learning.

1.4 THESIS STRUCTURE

Both research questions have a conceptual and an empirical subquestion. We therefore decide to split the book in two parts (Table 1.2). The first half provides a conceptual/ theoretical discussion of both questions, while the second half provides an empirical study of both. We choose this structure to keep the conceptual part self contained. In our view, the most important contribution of the dissertation is the conceptual part, which systematically structures the planning-learning field, and integrates their underlying disciplines. The empirical chapters that follow do have their standalone value: they have been published, and answer specific research questions in the field. However, in the context of the whole book, they also serve an illustratory purpose: they provide examples of novel combinations of planning and learning, and ways in which the field may be advanced.

It is useful to also put the thesis structure in the context of the associated PhD project. In this book, the theoretical part comes before the empirical part. However, in the chronology of the PhD project, we actually first conducted the empirical research. In some sense, we first needed to perform the empirical experiments to gain enough insight into the field to answer the questions on a conceptual level. Therefore, in a chronological sense, the thesis actually developed from back to front. We nevertheless invite the reader to still read the book from front to back, and thereby interpret the empirical chapters as an illustration to the conceptual part.

CHAPTER STRUCTURE We will shortly introduce the actual content of the chapters in this thesis. When applicable, we will also point to the relevant arrows in Figure 1.1 which the specific chapters discusses.

To keep the book self-contained, we start Chapter 2 by providing a short general introduction to MDPs, planning, and reinforcement learning. This serves as a background for readers with less experience in the field. The conceptual part of the book is covered in Chapters 3 and 4:

- Chapter 3 introduces a Framework for Reinforcement Learning and Planning (FRAP), which disentangles the common algorithmic space of planning and learning into its key underlying dimensions. As such, it illustrates that both fields share exactly the same methodology, and provides a common language for both fields to communicate their methods. This answers the conceptual part of research question 1. In Figure 1.1, it shows that planning (top-left) and (model-free) RL (top-right) actually deal with exactly the same algorithmic decisions.

- Chapter 4 provides a conceptual answer on the second research question. It presents a survey of model-based reinforcement learning, discussing the various ways to learn a model, the essential decisions to integrate planning and learning, and its potential benefits. As such, it provides a conceptual overview of the ways planning and learning can be combined. It discusses all the arrows in the bottom graph of Fig. 1.1 (the figure actually originates from this paper).

The empirical half of this thesis consists of Chapters 5-8. The first three chapters focus on the second research questions, presenting novel ways to combine planning and learning. In the last empirical chapter, we

also illustrate how the commonalities between planning and learning may provide mutual inspiration for new algorithms.

- Chapter 5 presents a novel model learning method, which is a key preliminary for model-based RL. In particular, we show how conditional variational inference in neural networks can be leveraged to flexibly learn transition functions in stochastic environments. In Fig. 1.1, this method deals with arrow g.
- Chapter 6 presents a new planning-learning integration, where we extend the successful AlphaZero (Silver et al., 2018) algorithm to deal with continuous action spaces, like frequently encountered in robotics tasks. In Fig. 1.1, this deals with a novel form of arrows a, b, c and d.
- Chapter 7 identifies an important new trade-off in planning-learning integration: how long should we plan before we act? This is a relatively little studied topic, but turns out to be of importance for the final performance. We show that an intelligent agent should neither plan too long nor too short, which can also be related to work from cognitive psychology on dual process theory (Evans, 1984; Kahneman, 2011). In Fig. 1.1, this studies the relative duration of planning (arrow a, b and plan box) versus acting and learning (arrows d and e).
- Finally, Chapter 8 presents MCTS-T+, an extension of the popular Monte Carlo Tree Search (MCTS) algorithm. MCTS-T+ uses ideas from exploration research in RL to improve the standard MCTS algorithm. As such, it provides an empirical illustration of the first research question, showing that both fields deal with the same algorithmic space, and can therefore profit from the solution that research in the other field has already come up with. In Fig. 1.1, this designs a new planning method (arrow a and Planning box) by taking inspiration from model-free RL (arrows e and f).

Altogether, the book provides both a conceptual and empirical study of the intersection of planning and learning. The key message of this dissertation is that planning and learning solve the same problem, have the same underlying algorithmic dimensions, but make some crucially different assumptions which turn out to be mutually beneficial. A good illustration of this last point is that optimal performance in a task appears to require both planning and learning, in a well-balanced

manner (Ch. 7). We will provide a more extensive discussion of our findings, and its implications for future work, in the Discussion (Ch. 9).

1.5 NOTATION

The chapters in this thesis were first published, or are in submission, as individual research papers. Therefore, notation can vary between chapters. Notation within a chapter is always consistent, and we therefore advice the reader to look within the same chapter for the meaning of an unclear symbol. However, in general the above remarks should not be much of an issue. We generally follow the notation conventions in the reinforcement learning, machine learning, and planning literature. We will shortly mention a few common notation conventions throughout this dissertation.

We reserve s for state, a for action, r for reward, t for timestep, $\mathcal{T}(s'|s,a)$ for dynamics function, and $\mathcal{R}(s,a,s')$ for reward function. Policies are denoted by $\pi(a|s)$, value functions by $V(s)$ and action-value functions by $Q(s,a)$. Probability distribution are denoted by $p(\cdot)$, or $p(\cdot|\cdot)$ when conditional. Counts are typically denoted by n. When we learn a parametrized function, the parameters are usually denoted by θ or ϕ. To indicate that a function is parametrized, we use subscripts, i.e., $f_\theta(x)$ is a function that takes in data x and has parameters θ.

Good statistical practice writes random variables with capital letters (X) and their realization with small letters (e.g., a sample x_i). RL conventions have always been less strict, and RL researchers usually write small letters, where the context should tell whether we deal with a random variable. For consistency with literature we also stick with these conventions. The same applies to scalar (x) versus vector ($\boldsymbol{x}$) versus matrix ($\boldsymbol{X}$) notation: we only explicitize these distinctions when they matter for understanding the proposed method.

2 BACKGROUND

ABSTRACT

This chapter provides a broad introduction to sequential decision making, and thereby to the remainder of the content of this dissertation. We first formally define intelligent sequential decision making in the form of a Markov Decision Process optimization. We then discuss the key difference between two approaches to this problem (planning and reinforcement learning), in the form of the type of access we get to the MDP dynamics function. Then, we provide short introductions of the individual research fields of planning and (model-free) reinforcement learning, and discuss their combination in the form of model-based reinforcement learning. Altogether, the chapter provides essential preliminaries to the remainder of this thesis.

In sequential decision-making, formalized as Markov Decision Process optimization, we are interested in the following problem: given a (sequence of) state(s), what next action is best to choose, based on the criterion of highest cumulative pay-off in the future. More formally, we aim for *context-dependent action prioritization based on a (discounted) cumulative reward criterion*. This is a core challenge in artificial intelligence research, as it contains the key elements of the world: there is sensory information about the environment (states), we can influence that environment through actions, and there is some notion of what is preferable, now and in the future. The formulation can deal with a wide variety of well-known problem instances, like path planning, robotic manipulation, game playing and autonomous driving.

This chapter will first formally introduce the MDP optimization problem (Sec. 4.2). Then, we shortly discuss the different types of access to the MDP dynamics, which formed the initial distinction between planning and learning (Sec. 2.2). Afterwards, we present a broad introduction to the planning (Sec. 2.3), reinforcement learning (Sec. 2.4), and model-based reinforcement learning (Sec. 2.5) fields. Together, the chapter provides a broad introduction to the problem type and relevant research fields.

2.1 MARKOV DECISION PROCESS

The formal definition of a *Markov Decision Process* (MDP) (Puterman, 2014) is the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_0), \gamma\}$. The environment consists of a *transition function* $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$ and a *reward function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. At each timestep t we observe some state $s_t \in \mathcal{S}$ and pick an action $a_t \in \mathcal{A}$. Then, the environment returns a next state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ and associated scalar reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The first state is sampled from the initial state distribution $p(s_0)$. Finally, $\gamma \in [0, 1]$ denotes a discount parameter.

The agent acts in the environment according to a *policy* $\pi : \mathcal{S} \to p(\mathcal{A})$. In the search community, a policy is also known as a *contingency plan* or *strategy* (Russell and Norvig, 2016). By repeatedly selecting actions and transitioning to a next state, we can sample a *trace* through the environment.

The *cumulative return* of trace through the environment is denoted by:

$$J_t = \sum_{k=0}^{K} \gamma^k \cdot r_{t+k}, \tag{2.1}$$

for a trace of length K. For $K < \infty$ we call this the *finite-horizon* return, for $K = \infty$ it is the *infinite-horizon* return. In the latter case, we either continue the episode until we encounter a *terminal state*, or otherwise we continue forever. Termination happens in specific states in which, by the MDP definition, there are no available actions.[1]

When we assume an infinite-horizon return ($K = \infty$), we theoretically require $\gamma < 1$ to ensure that the above cumulative reward stays finite. However, in practice this is often not a big issue. Throughout this thesis, we assume the infinite-horizon setting, i.e., we will use:

$$J_t = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k}, \tag{2.2}$$

Define the action-value function $Q^\pi(s, a)$ as the expectation of the cumulative return given a certain policy π:

$$Q^\pi(s, a) \doteq \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{K} \gamma^k r_{t+k} \middle| s_t = s, a_t = a \right] \tag{2.3}$$

1 We may also define a terminal state as a state in which all actions lead back to the state itself, and which all have a reward of 0. As such, we can never achieve any additional reward from the terminal state.

This equation can be written in a recursive form, better known as the *Bellman equation*:

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)} \left[\mathcal{R}(s,a,s') + \gamma \, \mathbb{E}_{a' \sim \pi(\cdot|s')} \left[Q^{\pi}(s',a') \right] \right] \tag{2.4}$$

Our goal is to find a policy π that maximizes our expected return $Q^{\pi}(s,a)$:

$$\pi^{\star} = \arg\max_{\pi} Q^{\pi}(s,a) = \arg\max_{\pi} \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{K} \gamma^k r_{t+k} \middle| s_t = s, a_t = a \right] \tag{2.5}$$

There is *at least* one optimal policy, denoted by $\pi^{\star}$, which is better or equal than all other policies π (Sutton and Barto, 2018). In the planning and search literature, the above problem is typically formulated as a cost *minimization* problem (Russell and Norvig, 2016). That formulation is interchangeable with our presentation by negating the reward function. The formulation also contains *stochastic shortest path* problems (Bertsekas and Tsitsiklis, 1991), which are MDP formulations with absorbing states at goal states, where we attempt to reach the goal with a little cost as possible.

2.2 REVERSIBLE VERSUS IRREVERSIBLE ACCESS TO THE MDP DYNAMICS

We already discussed the possible separations between planning and learning in Sec. 1.1: 1) based on reversible or irreversible access to the MDP dynamics, and 2) based on the use of a local or global solution. In this section, we will further clarify the first issue, i.e., what access to the MDP (dynamics $\mathcal{T}$ and reward function $\mathcal{R}$) are we provided with? We identify three ways in which we can get access to the MDP:

- *Reversible analytic* environments specify the entire probability distribution $\mathcal{T}(s'|s,a)$. In Figure 2.1 top-left, we see an example with three possible next states, where the probability of each state is fully explicitized. Such access allows for exact evaluation of the Bellman equation.
- *Reversible sample* environments provide a single sample from $s' \sim \mathcal{T}(\cdot|s,a)$, but do not give access to the underlying probabilities.

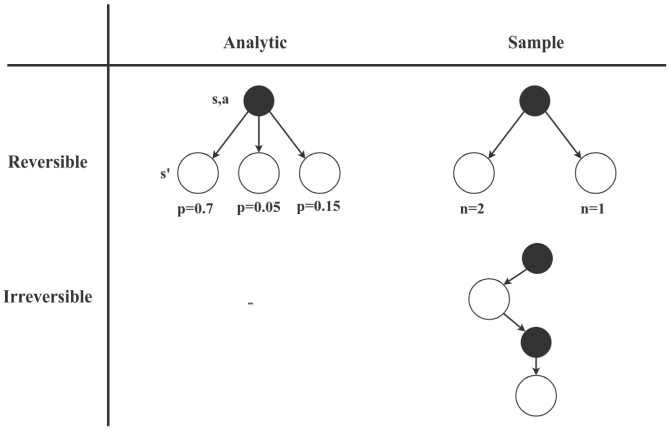


Figure 2.1: Types of access to the environment dynamics. Columns: On each trial, we may either get access to the exact transition probabilities of each possible transition (analytic or descriptive model), or we may only get a sampled next state (sample or generative model). Rows: Additionally, we may either be able to revert the model and make another trial from the same state (reversible), or we may need to continue from the resulting state (irreversible). Planning algorithms assume a reversible environment, RL algorithms assume an irreversible environment. We could theoretically think of an irreversible analytic environment, in which we do see the probabilities of each transition but can only continue from one drawn realization, but we are unaware of such a model in practice.

In Figure 2.1, top-right, we sampled the same state-action three times, which gave two times the first and one time the third next state.

- *Irreversible sample* environments also provide a sample, but introduce another restriction: we need to keep sampling forward. In other words, we cannot consider the same state twice directly after each other. If we want to get back, then we will have to pick the correct actions to bring us back to the specific state. The key example of an irreversible sampler is the real-world, in which we cannot revert time. For many real world problems it is hard to specify an analytic or reversible sample model, but we can always get irreversible sample data by interacting with the real world.

These models, based on two underlying distinctions, are summarized in Figure 2.1. Regarding the terminology, a model from which we can

(only) sample is often referred to as a *generative model*. However, in supervised learning (where we aim to predict y from x), generative models are often opposed to *discriminative* models (Jaakkola and Haussler, 1999). Discriminative models learn the conditional probability distribution $p(y|x)$, while generative models learn $p(x, y)$. Since we here want to make the contrast with 'analytic' models, which have full access to the density, we chose to use the term 'sample' model.

Note that there is an ordering in the above access types. We can always decide to sample from an analytic model, and we can always restrict ourselves to never revert the environment. Therefore, the reversible analytic model gives us most information and freedom. On the other hand, sample models are usually easier to obtain, and irreversible sampling is of course an important property of the real world, in which we ultimately want to apply learning.

The initial difference between planning and RL was based on the above assumption, i.e., RL assumed irreversible access to the MDP dynamics ('unknown model'), while planning assumed reversible access ('known model'). As we already discussed, this assumption led to a second distinction, i.e., RL assumes a global solution, while planning assumes a local solution. From these distinctions, both fields have developed their own methods and preferences for solving the MDP optimization problem, which will be covered in the next section.

2.3 PLANNING

Planning (or *search*) is a large research field within artificial intelligence (Russell and Norvig, 2016). We defined planning as a class of MDP algorithms that 1) use a model, and 2) store a local problem solution, like a value or action recommendation. We shortly list some important planning approaches. This presentation is by no means exhaustive, but it does establish some common ground of algorithms we consider in our framework:

- **Dynamic programming** (DP) (Bellman, 1966; Howard, 1960): The key idea of Dynamic programming is to break the optimization problem into smaller subproblems given by the 1-step optimal Bellman operator. We then sweep through state-space, repeatedly solving the small subproblem which eventually solves for the optimal policy. DP is a bridging technique between both planning and reinforcement learning. However, the tabular implementation does not scale well to high-dimensional problems, since the size

of the required table grows exponentially in the dimensionality of the state space ('the curse of dimensionality'). To solve for this issue, Real-time Dynamic Programming (RTDP) (Barto, Bradtke, and Singh, 1995) only applies DP updates on traces sampled from some start state distribution.

- **Heuristic search**: These search approach built a forward tree from some start state. Initial research largely focused on uninformed search strategies, like breadth-first search (BFS) (Moore, 1959) and Dijkstra's shortest path algorithm (Dijkstra, 1959). These approaches track a *frontier*, which is the set of nodes that have themselves been visited, but whose successor states have not all been visited yet. Later approaches successfully incorporated heuristics, which are functions that provide an initial *optimistic* estimate of the return from a particular state. A well-known heuristic search algorithm is $A^\star$ (Hart, Nilsson, and Raphael, 1968). However, for many problems informative and admissible heuristics are not trivial to obtain. Extensive introductions to planning methods on MDPs, including heuristics, are provided by Geffner and Bonet (2013) and Kolobov (2012).

- **Sample-based search**: This group of search algorithms estimates state-action values based on statistical sampling methods. The simplest example is Monte Carlo search (MCS) (Tesauro and Galperin, 1997), where we sample n traces for each currently available action and use their mean return as an estimate of the value of that action. A successful extension of this paradigm is Monte Carlo Tree Search (Browne et al., 2012; Coulom, 2006; Kocsis and Szepesvári, 2006). While MCS only tracks statistics at the root of the tree, MCTS recursively applies the same principle at deeper levels of the tree search. Exploration and exploitation within the tree are typically based on variants of the upper confidence bounds (UCB) rule (Kocsis and Szepesvári, 2006). Pure MCTS for example showed early success in the game of Go (Gelly and Wang, 2006). MCTS originates in regret minimization (Auer, 2002), which attempts to select the optimal action as often as possible during the search. In contrast, *best-arm identification* (BAI) tries to identify the optimal root action at the end of the search (Kaufmann and Koolen, 2017), which allows for additional exploration during the search itself. Finally, in the robotics path planning community there is another successful branch of sample-based planning algorithms known as *rapidly-exploring random trees* (RRTs) (LaValle,

1998). While MCTS samples in action space to build the tree, RRTs sample in state space, which is only feasible if the state-space is not too large.

- **Gradient-based planning**: This planning approach is especially popular in the robotics and control community. If we have a differentiable dynamics models (either pre-known or learned from data), then we can directly obtain the derivative of the cumulative reward objective with respect to the policy parameters by differentiating through the dynamics function. An especially popular approach in this category applies when we have a linear dynamics model and a quadratic reward function. In that case, we can derive closed-form expressions for the optimal action, known as the linear-quadratic regulator (LQR) (Anderson and Moore, 2007; Kalman, 1960). While most practical problems have non-linear dynamics, this problem can be partly mitigated by iterative LQR (iLQR) (Todorov and Li, 2005), which repeatedly makes local linear approximations to the true dynamics. In RL literature, gradient-based planning is referred to as *value gradients* (Fairbank and Alonso, 2012).

- **Direct optimization**: We may also treat the planning challenge as a black-box optimization problem. This approach is especially popular in the robotics and control community, better known as *direct optimal control* (Bock and Plitt, 1984). In this approach we reformulate the objective as a non-linear programming problem, in which the dynamics typically enter as constraints on the solution. We then parametrize a trajectory (a local policy), and perform hill-climbing in this parameter space, for example based on finite-differencing. In the next section on RL, we will encounter similar ideas known as *policy search*.

Another direction of planning research that has been popularized in the last decade treats planning as probabilistic inference (Botvinick and Toussaint, 2012; Kappen, Gómez, and Opper, 2012; Toussaint, 2009), where we use message-passing like algorithms to infer which actions would lead to receiving a final reward. Note that we do leave out some planning fields that depart from the generic MDP specification. For example, *classical planning* (Ghallab et al., 1998) requires a propositional logic structure of the state space. Approaches in this field may plan based on delete relaxations, in which we temporarily ignore attributes in the state that should be removed, and only focus on solving for the

ones that should be added. These methods require additional structure on the state space, and are therefore not considered in this thesis.

In practice, many planning approaches cannot solve for the entire optimal policy from the start. Therefore, they frequently employ a form of *receding horizon control* (Mayne and Michalska, 1990; Thomas, 1975), also known as *model predictive control* (Richalet et al., 1978). In those cases, we plan for the optimal policy up to a certain depth, execute the optimal action, and then repeat the process from the next state. This approach moves forward in the domain on preliminary estimates of the value of each action, but does make execution practically feasible, and may still lead to good solutions.

2.4 MODEL-FREE REINFORCEMENT LEARNING

Reinforcement learning is a large research field within machine learning. As discussed in the introduction, we consider the defining assumption of reinforcement learning the use of global, learned representation of the solution (like a value or policy function). This section covers model-free RL, where we have irreversible access to the MDP dynamics, and directly learn from sampled data from this environment (similar to directly learning from real world interaction).

The planning literature (introduced above) is mostly organized in sub-disciplines, where each discipline focuses on its own set of assumptions or particular approach. In contrast, the RL community is less organized in subtopics, but has rather focused on a range of factors that can be altered in algorithms. This already hints at the possibility of a framework, which should disentangle such factors. We will here introduce some important concepts in RL literature:

- **Value and policy**: While many planning algorithms search for a local solution (e.g., a single trajectory, or only a solution for the current state), RL algorithms in principle approximate a solution for the entire state space. Since RL agents can only try an action once and then have to continue, we cannot really learn a local solution, since we do not know when we will be able to return to the current state. Solutions are usually stored in the form of a value function (from which the policy is implicitly derived) or a policy function. Some approaches learn both, where the value function aids in updating the policy, better known as *actor-critic* methods.

- **On-policy and off-policy bootstrapping**: A crucial idea in RL literature is *bootstrapping*, where we plug in the learned estimate of the value of a state to improve the estimate of a state that precedes it. A key concept is the *temporal difference* error, which is the difference between our previous and new estimate of the value of a state (Sutton, 1988). When bootstrapping state-action values, there is an important distinction between *on-policy* learning, we we estimate the value of the policy that we actually follow, and *off-policy* learning, where we create a value estimate of another (usually greedy) policy. Cardinal examples of the on- and off-policy cases are SARSA (Rummery and Niranjan, 1994) and Q-learning (Watkins and Dayan, 1992), respectively.

- **Exploration**: Exploration is a fundamental theme in nearly all optimization research, where we typically store a (set of) current solution(s) and want to explore to a (set of) potentially better candidate solution(s) around the current solution (set). However, exploration is extra relevant in reinforcement learning, because we also need to collect our own data, which makes the process more brittle.

  Many RL exploration methods have focused on injecting some form of noise into the action space decision. Some methods, like ϵ-greedy and Boltzmann exploration, use random perturbation, while other approaches, like confidence bounds (Kaelbling, 1993) or Thompson sampling (Thompson, 1933), base exploration decisions on the remaining uncertainty of an action. While these methods explore in action space, we can also explore in policy parameter space (Plappert et al., 2017). There are other exploration approaches based on intrinsic motivation (Chentanez, Barto, and Singh, 2005), like curiosity (Schmidhuber, 1991a), or by planning ahead over an uncertain dynamics model (Guez, Silver, and Dayan, 2012).

- **Generalization**: Since RL tends to store global solutions, it is typically infeasible to store them in a table for problems with a higher dimensional state-space (due to the curse of dimensionality, as already mentioned in the section on Dynamic Programming). Therefore, the RL literature has largely focused on learning methods to *approximate* the solution. Note that such approximation is a supervised learning task itself, which frequently creates a

nested supervised learning optimization loop within the outer RL optimization.

A plethora of function approximation methods has been applied to RL, including tile coding, (Sutton, 1996), linear approximation (Bradtke and Barto, 1996), and a recent explosion of (deep) neural network (Goodfellow, Bengio, and Courville, 2016) applications to RL (Mnih et al., 2015). Recent surveys of deep RL methods are provided by François-Lavet et al. (2018) and Arulkumaran et al. (2017). Learning not only allows a global solution to be stored in memory (in approximate form), but, equally important, its generalization also provides a fundamental way to share information between similar states.

- **Direct policy optimization**: We may also approach MDP optimization as a direct optimization problem in policy parameter space. An important example are policy gradient methods (Sutton and Barto, 2018; Sutton et al., 2000; Williams, 1992), which provide an unbiased estimator of the gradient of the objective with respect to the policy parameters. We will discuss the policy gradient theorem in much greater detail in Sec. 3.2.7 of our framework. There has been much research on ways to stabilize policy gradients, for example based on trust region optimization methods (Schulman et al., 2015).

  Some gradient-free policy search methods only require the ability to evaluate the objective (the expected cumulative return). Example approaches include evolutionary strategies (ES) applied to the policy parameters (Moriarty, Schultz, and Grefenstette, 1999; Salimans et al., 2017; Whiteson and Stone, 2006), and the use of the cross-entropy method (CEM) (Mannor, Rubinstein, and Gat, 2003; Rubinstein and Kroese, 2013). These approaches treat the MDP as a true black box function which they only need to evaluate. Therefore, they use less MDP specific properties, and will also receive less emphasis in our framework.

There are many specific subtopics in RL research, like partial observability (Chrisman, 1992), hierarchy (Barto and Mahadevan, 2003), goal setting and generalization over different goals (Schaul et al., 2015), transfer between tasks (Taylor and Stone, 2009), inverse reinforcement learning (Abbeel and Ng, 2004), multi-agent learning (Busoniu, Babuska, and De Schutter, 2008), etc. While these topics are all important, our framework (Chapter 3) solely focuses on a single agent in a single

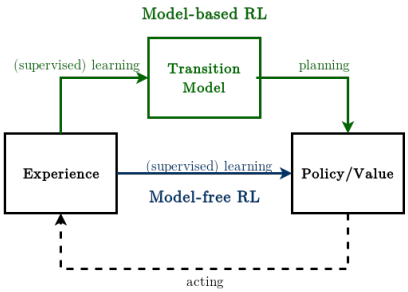


Figure 2.2: Model-based versus model-free reinforcement learning. In model-free RL (blue), we directly use experience (data) acquired from the environment to improve a value/policy. In model-based RL (green), we additionally use the sampled data to learn a model, which can then be used to update the value or policy through planning.

MDP optimization task. However, many of the above topics are complementary to our framework. For example, we may use meta-actions (hierarchical RL) to define new, more abstract MPDs, in which all of the principles of our framework are again applicable.

2.5 MODEL-BASED REINFORCEMENT LEARNING

In model-based reinforcement learning (Hester and Stone, 2012b; Moerland, Broekens, and Jonker, 2020b; Sutton, 1990), the two research fields of planning and reinforcement learning merge. The original idea of model-based RL was to start from an irreversible environment, and then: i) use sampled data to learn a dynamics model, and ii) use the learned model to improve a learned value or policy. This idea is illustrated in Figure 2.2.

However, more recently we have also seen a surge of techniques that start from a reversible model, but also use learning techniques for the value or policy. An example is AlphaGo Zero (Silver et al., 2017c). Since most researchers also consider this model-based RL, we defined model-based RL as: 'any MDP algorithm that 1) uses a model, and 2) stores a global, learned solution, like a value or policy function.'

There are two important steps in model-based RL. When our access to the ground-truth MDP is irreversible, we will first have to learn the model from sampled data. This is similar to the supervised learning setting, except for the challenge that we influence the data we actually

observe. Since our survey (Ch. 4) extensively discusses this topic, we will not further discuss this topic here. The second important step of model-based RL involves usage of the learned reversible model to improve a value or policy. We will list a few successful approaches to integrate planning in global function approximation:

- **Sampling additional data**: The classic idea of model-based RL was to use the model to sample additional data, which can then be used for standard model-free updates. This idea was first introduced in the well-known Dyna algorithm (Sutton, 1990).
- **Multi-step approximate dynamic programming**: More complex integrations use a form a multi-step approximate dynamic programming (Efroni et al., 2018, 2019). In this approach, we use the reversible model to make a multi-step planning back-up, which is then used to update a value or policy approximation at the root of the search. This approach has received much recent attention, for example in AlphaGo Zero (Silver et al., 2017c) and Guided Policy Search (Levine and Koltun, 2013).
- **Backward trials**: While most models have a forward view (which next states may result from a particular state-action pair), we can also learn a backward model (given a particular state, which state-action pairs could bring us there). A backward model allows us to spread new information more quickly over state-space, by identifying all the possible precursors of a changed state-action value estimate. This idea is better known as *prioritized sweeping* (PS) (Moore and Atkeson, 1993).
- **Value gradients**: When the function class of our learned dynamics model is differentiable, then we can apply gradient-based planning (already introduced in Sec. 2.3). In the RL literature, this approach is known as *value gradients* (Fairbank and Alonso, 2012). A successful example is PILCO (Deisenroth and Rasmussen, 2011), which learns a Gaussian Process (GP) transition model, and combines this with gradient-based planning to achieve good data efficiency in real-world robotics tasks.

We present a much more extensive discussion of the ways to integrate planning and learning in the survey in Chapter 4. For example, the survey also discusses implicit approaches to model-based RL, like MuZero (Schrittwieser et al., 2019), Value Iteration Networks (VIN) (Tamar et al., 2016) and TreeQN (Farquhar et al., 2018). As we will see,

these methods actually blur the line between model-free and model-based RL.

This concludes our short introduction of the MDP optimization problem, and the three solution approaches of planning, model-free RL, and model-based RL. For further details on the MDP optimization problem, we refer the reader to Puterman (2014). Further details on model-free and model-based reinforcement learning can be found in Sutton and Barto (2018), while an extensive discussion of planning methods is provided by Russell and Norvig (2016). In the next chapter, we will present a framework to disentangle the common factors underneath these methods.

Part II

CONCEPTUAL INTEGRATION OF PLANNING AND LEARNING

3

FRAP: A UNIFYING FRAMEWORK FOR REINFORCEMENT LEARNING AND PLANNING [1]

ABSTRACT

Sequential decision making, commonly formalized as Markov Decision Process optimization, is a key challenge in artificial intelligence. Two successful approaches to MDP optimization are planning and reinforcement learning. Both research fields largely have their own research communities. However, if both research fields solve the same problem, then we should be able to disentangle the common factors in their solution approaches. Therefore, this chapter presents a unifying framework for reinforcement learning and planning (FRAP), which identifies the underlying dimensions on which any planning or learning algorithm has to decide. At the end of the chapter, we compare - in a single table - a variety of well-known planning, model-free and model-based RL algorithms along the dimensions of our framework, illustrating the validity of the framework. Altogether, FRAP provides deeper insight into the algorithmic space of planning and reinforcement learning, and also suggests new approaches to integration of both fields.

3.1 INTRODUCTION

Sequential decision making is a key challenge in artificial intelligence research. The problem, commonly formalized as a Markov Decision Process (MDP) (Puterman, 2014), has been studied in different research fields. The two prime research directions are *reinforcement learning* (Sutton and Barto, 2018), a subfield of machine learning, and *planning* (also known as *search*), of which the discrete and continuous variants have been studied in the fields of artificial intelligence (Russell and Norvig, 2016) and control (Bertsekas, 1995), respectively. Planning and learning approaches differ with respect to a key assumption: is the dynamics model of the environment known (planning) or unknown (reinforcement learning).

Departing from this distinctive assumption, both research fields have largely developed their own methodology, in relatively separated com-

1 Chapter based on: Moerland TM, Broekens J, Jonker CM. A Framework for Reinforcement Learning and Planning. *In submission*.

munities. There has been cross-breeding as well, better known as 'model-based reinforcement learning' (surveyed in Chapter 4). However, a unifying view on both fields, including how their approaches overlap or differ, currently lacks in literature (see Section 3.4 for an extensive discussion of related work).

Therefore, this chapter introduces the Framework for Reinforcement learning and Planning (FRAP), which identifies the essential algorithmic decisions that any planning or RL algorithm has to make. We idenfity six main questions: 1) where to put our computational effort, 2) where to make our next trial, 3) how to estimate the cumulative return, 4) how to back-up, 5) how to represent the solution and 6) how to update the solution. As we will see, several of these questions have multiple subquestions. However, the crucial message of this chapter is that any RL or planning algorithm, from Q-learning (Watkins and Dayan, 1992) to A* (Hart, Nilsson, and Raphael, 1968), will have to make a decision on each of these dimensions. We illustrate this point at the end of the chapter, by formally comparing a variety of planning and RL papers on the dimensions of our framework.

The framework first of all provides a common language to categorize algorithms in both fields. Too often, we see researchers mention 'they use a policy gradient algorithm', while this only specifies the choice on one of the dimensions of our framework and leaves many other algorithmic choices unspecified. Second, the framework identifies new research directions, for example by taking inspiration from solutions in the other research field, or by identifying novel combinations of approaches that are still left untried. Finally, it can also serve an educational purpose, both for researchers and students, to get a more systematic understanding of the common factors in sequential decision-making problems.

Essential preliminaries on planning and reinforcement learning were already provided in the previous chapter. We will next introduce the framework (Section 3.2). Afterwards, we will compare various well-known planning and reinforcement learning algorithms along the dimensions of our framework (in Table 3.3), thereby illustrating the generality of FRAP. We conclude the chapter with Related Work (Sec. 3.4), Discussion (Sec. 3.5) and Summary (Sec. 3.6) sections.

3.2 FRAMEWORK FOR REINFORCEMENT LEARNING AND PLANNING

We now introduce the Framework for Reinforcement Learning and Planning (FRAP). One of the key messages of this chapter is that both planning and reinforcement learning make the exact same algorithmic choices to solve the MDP problem. For example, a MCTS search of 500 traces is conceptually not too different from 500 episodes of a model-free Q-learning agent in the same environment. In both cases, we repeatedly move forward in the environment to acquire new information, make back-ups to store this information, with the goal to make better informed decisions in the next trace/episode. The model-free RL agent is restricted in the order in which it can visit states, but otherwise, the methodology of exploration, back-ups, representation and updates is largely the same (which we will show later in this chapter).

We will center our framework around the concept of *trials* and *back-ups*. We will first introduce these in Section 3.2.1. Afterwards, we will introduce the dimensions of our framework:

- Where to put our computational effort? (Sec. 3.2.2)
- Where to make the next trial? (3.2.3)
- How to estimate the cumulative return? (3.2.4)
- How to back-up? (3.2.5)
- How to represent the solution? (3.2.6)
- How to update the solution? (3.2.7)

Table 3.1 is crucial, since it summarizes our entire framework, and can be used as a reference point throughout the sections.

3.2.1 *Trials and back-ups*

We will first conceptually define a trial and a back-up:

1. **Trial**: A trial consists of a single call to the environment.[2] We have to specify a certain state action pair (s, a), and the environment gives us either a sample from, or the entire distributions of,

2 Note that especially in the early RL literature,'trial' has also been used to refer to an entire episode. Here, we specifically use trial to refer to a single call to the environment, i.e., the MDP dynamics and reward function.

Table 3.1: Overview of dimensions in the Framework for Reinforcement learning and Planning (FRAP). The right column shows possible choices on each dimension and subconsideration. IM = Intrinsic Motivation.

Dimension	**Consideration**	**Choices**
1. Comp. effort (3.2.2)	- State set	All ↔ reachable ↔ relevant
2. Trial selection (3.2.3)	- Candidate set	Step-wise ↔ frontier
	- Exploration	Random ↔ Value-based ↔ State-based - For value: mean value, uncertainty, priors - For state: ordered, priors (shaping), novelty, knowledge IM, competence IM
	- Phases	One-phase ↔ two-phase
	- Reverse trials	Yes ↔ No
3. Return estim. (3.2.4)	- Sample depth	$1 \leftrightarrow n \leftrightarrow \infty$
	- Bootstrap func.	Learned ↔ heuristic ↔ none
4. Back-up (3.2.5)	- Back-up policy	On-policy ↔ off-policy
	- Policy expec.	Expected ↔ sample
	- Dynamics expec.	Expected ↔ sample
5. Representation (3.2.6)	- Function type	Value ↔ policy ↔ both (actor-critic) - For all: generalized ↔ not generalized
	- Function class	Tabular ↔ function approximation - For tabular: local ↔ global
6. Update (3.2.7)	- Loss	Squared, etc. (for value) ↔ (det.) policy gradient, value gradient, cross-entropy, etc. (for policy)
	- Update	Gradient-based ↔ gradient-free - For gradient-based, special cases: replace & average update

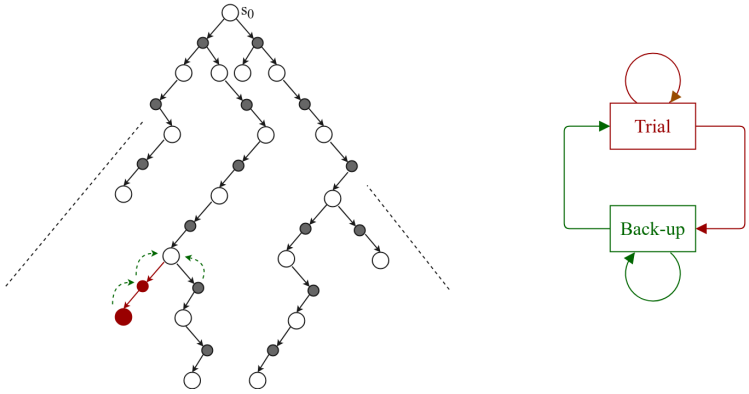


Figure 3.1: Trials and back-ups. Left: Grey nodes visualize a search tree, consisting of all the state-action pairs evaluated in the domain so far. In red we visualize the next trial, which picks a state-action pair an queries the environment for either a sample from, or the entire distributions of, $\mathcal{T}$ and $\mathcal{R}$. The green dotted arrows visualize a back-up, in which the newly acquired information is used to update our value estimates of the state(s) above it. Right: Key procedure in FRAP consists of iterated trials and back-ups.

$\mathcal{T}(s'|s,a)$ and $\mathcal{R}(s,a,s')$ (depending on what access we have to the environment, see Figure 2.1). In Figure 3.1 this is visualized in red.

2. **Back-up**: The second elementary operation is the 1-step back-up, which uses the information on the state-action pairs below it (for example obtained from the last trial) to update the information of the state-action pairs above it. In Figure 3.1 this is visualized in green. The back-up can involve any type of information, but frequently involves estimates of state-action values.

The central idea of nearly all MDP optimization algorithms is that the information in the back-up allows us to better choose the location of the next trial. Therefore, most algorithms iterate both procedures. However, we are not forced to alternate a trial and a back-up (Figure 3.1, right). We may for example first make a series of trials ('a roll-out') to go deep into the domain, and then make a series of back-ups to propagate the information all the way up to the root node.

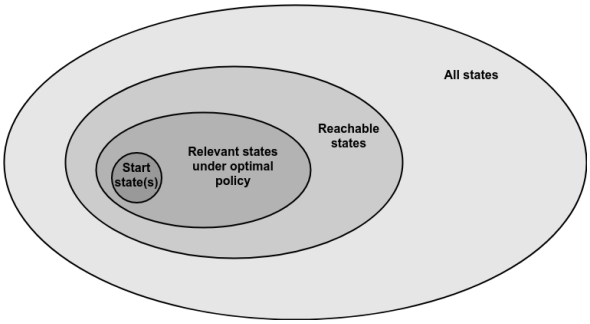


Figure 3.2: Four sets of states. The reachable state set, a subset of the entire state space, consist of states that are reachable from any start state under any policy. A subset of the reachable states are the relevant states, which are reachable from any start state under the optimal policy. The start states are by definition a subset of the relevant states.

3.2.2 *Where to put our computational effort?*

To make the MDP optimization tractable, the first question that any algorithm implicitly asks is: are there states that we can completely ignore? Fundamentally, we can identify four sets of states, as graphically illustrated in Figure 3.2:

1. **All states**: i.e., $\mathcal{S}$.
2. **Reachable states**: all states reachable from any start state under *any* policy.
3. **Relevant states**: all states reachable from any start state under the *optimal* policy[3].
4. **Start states**: all states with non-zero probability under $p(s_0)$.

Some algorithms find a solution for all states, the most noteworthy example being Dynamic Programming (DP). Such approaches tend to break down in larger problems, as the number of unique states grows exponentially in the dimensionality of the state space. As an illustration, imagine we apply DP to video game playing, where the input is a low-resolution 200x200 pixel greyscale image, with each pixel taking values

3 Intuitively, one may prefer to include a solution for states close to the optimal solution. However, conceptually, when we truly obtain the optimal solution, the only relevant states (in a stationary MDP) are those reachable under the optimal policy.

between 0 and 255. Then the state space consists of $256^{(200 \cdot 200)}$ unique states, a quantity without any meaningful interpretation. However, this state space contains all possible screen configurations, including enormous amounts of noise images that will never occur in the game.

Therefore, nearly all planning and RL methods start updating states from some start state, thereby only considering states that they can actually reach. Without additional information, this is the only practical way to identify reachable states. However, the reachable state set still tends to be large, and ultimately we are only really interested in the policy in those states that we will encounter under the optimal policy (the relevant states). As the optimal policy is not known in advance, nearly all algorithms beside Dynamic Programming start from the reachable set, and try to gradually narrow this down to the relevant state set. We will discuss approaches to gradually focus on the relevant set in the next section (on exploration).

Some specifications do provide additional information, for example in the form of explicit goal states. This frequently happens in path-planning problems, where we for example want to navigate to a certain destination. This is a form of prior knowledge on the form of the reward function, which peaks at the goal. In such cases, we can also include backwards planning from the goal state, which identifies the reachable state set from two directions. This principle was first introduced as *bidirectional search* (Pohl, 1969), and for example also part of some RRT approaches (LaValle, 1998).

3.2.3 *Where to make the next trial?*

A trial is the fundamental way to obtain new information about the environment. The crucial question then becomes: at which state-action pair should we make our next trial? Note that in RL (where we have an irreversible environment), the state-action pairs that we can access is usually restricted, but over a sequence of actions the same principles apply (i.e., where do we want to go to make more trials). There are two considerations we need to make for trial selection. First, we need to decide on a *candidate set* of state-action pairs that will be considered for the next trial (Sec. 3.2.3.1). Then, we need to actually decide which candidate from the set to select, which needs to incorporate some amount of *exploration* (Sec. 3.2.3.2). At the end of the section, we also briefly touch upon two additional generic concepts in trial selection (phases and reverse trials, in Sec. 3.2.3.3 and 3.2.3.4, respectively).

3.2.3.1 *Candidate set selection*

The first step is to determine the set of state-action pairs that are candidates for the next trial (in the current iteration). There are two main approaches:

- **Step-wise**: The most frequent approach is to explore *step-wise* on traces from some start state. At each step in the trace, the candidate set consists of all available actions in the current state. After finishing a sequence of step-wise candidate sets, we typically reset to a start state, and repeat the same procedure. This is the standard approach for most RL approaches (Sutton and Barto, 2018) and also for many planning algorithms, such as MCTS (Browne et al., 2012). Note that methods that explore by perturbation in (policy) parameter space (Plappert et al., 2017) can be seen as a special form of step-wise perturbation, where the perturbation for all steps is already fixed at the beginning of the episode.
- **Frontier**: The second type of candidate set is a frontier, illustrated in Figure 3.3. A frontier (or open list) (Dijkstra, 1959) consists of the set of states at the border of the explored region, i.e. those states who have themselves been visited, but whose child states have not all been visited yet. In a search tree, the frontier consists of all leaf nodes, with duplicate states removed (only keeping the occurrence with the lowest cost to reach). The cardinal value-based frontier exploration algorithm is the heuristic search algorithm A* (Hart, Nilsson, and Raphael, 1968).

The key difference between step-wise and frontier candidate sets is the moment at which they start exploration (next section). Step-wise methods have a new candidate set at every step in the trace. In contrast, frontier methods only have a single candidate set per episode, fixing a new target at the horizon, and only starting exploration once they are on the frontier.

There a pros and cons for both step-wise and frontier-based candidate sets. A benefit of frontier exploration is that it will by definition explore a new node. By storing the edges of the region you have already visited, you are guaranteed to make a step into new territory. In contrast, step-wise exploration has a risk to repeatedly trial around in an already explored region of state space. This is especially pronounced in tasks with bottleneck states (a narrow passage which brings the agent to another region of the problem). As Ecoffet et al. (2019) mention, step-wise exploration methods already apply exploration pressure while

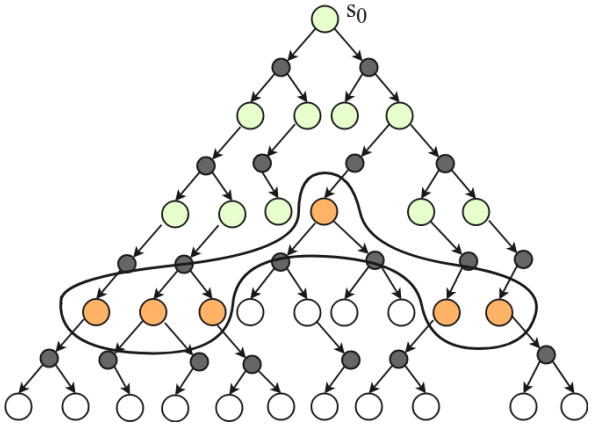


Figure 3.3: Illustration of the frontier. Green-shaded nodes have been completely visited, in a sense that either all their children have been visited, or they are terminal. Orange nodes are part the the search tree, but have unvisited child nodes left. Together the orange nodes constitute the frontier (black line), from which we want to continue exploring. White nodes are still unexplored.

getting back to the frontier, while we actually want to get back to a new region first, and only then explore. In the long run, (random) step-wise exploration methods will of course also hit the frontier, but this may take a long time of wandering around in known territory.

Frontier candidate sets also have their challenges. First, frontier exploration assumes that we can always get back to a particular frontier node, which is not guaranteed in stochastic domains (although we may also opt to *approximately* reach the same node (Péré et al., 2018). Moreover, in larger problems, the frontier may become very large, let alone all the paths towards it. In those cases, we can no longer store the frontier as a list, or the paths towards it as a tree. We then need to use representation learning for storing both the frontier (Ecoffet et al., 2019) and the paths towards it (Péré et al., 2018), which may generate instability, and make it hard to actually reach the frontier. Step-wise exploration methods do not have to deal with these issues.

3.2.3.2 *Exploration*

Once we have defined the candidate set, we need to decide which state-action pair in the set we will select for the next trail. The *exploitation*

decision is to greedily select the action with the highest value estimate. However, as discussed before, this will lead to suboptimal performance. We need to add *exploration* pressure to the greedy policy. We identify three main ways to achieve this: i) random perturbation, ii) value-based perturbation, and iii) state-based perturbation.

- **Random exploration**: In this category we simply inject random exploration noise to the greedy policy. The classic example is ϵ-greedy exploration (Sutton and Barto, 2018), which (in a step-wise candidate set) with small probability randomly selects one of the other actions, independently of its current value estimate or any other characteristics. In continuous action space the noise can for example be Gaussian. We may also inject the noise in (policy) parameter space (Plappert et al., 2017), which may help to correlate it over timesteps.

  A benefit of random exploration approaches is that they can guarantee to retain positive exploration pressure throughout learning, and may therefore escape a local optimum when given (a lot of) time. However, they have serious drawbacks as well. Random exploration is undirected, which may lead to *jittering* behaviour, where we undo an exploratory step in the next step (Osband et al., 2016). Moreover, there is no good measure of progress (when should exploration stop), and these methods therefore typically require tedious hyperparameter tuning.

- **Value-based exploration**: A second approach is to use value-based information to better direct the perturbation. There are several approaches:

  – *Mean action values*: We may potentially improve over random exploration by incorporating the mean estimates of all the available actions. The general idea is that actions with a higher value estimate also deserve more exploration pressure. In discrete action space, the cardinal example of this approach is Boltzmann exploration (Sutton and Barto, 2018):

$$\pi(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{b\in\mathcal{A}} \exp(Q(s,b)/\tau)}, \tag{3.1}$$

  where $\tau \in \mathbb{R}$ denotes a temperature parameter that scales exploration pressure.

For continuous action spaces, we may achieve a similar effect through entropy regularization (Mnih et al., 2016; Peters, Mulling, and Altun, 2010). These methods usually optimize an adjusted reward function of the form:

$$r(s_t, a_t, s_{t+1}) + \alpha \cdot H(\pi(\cdot|s_t)), \tag{3.2}$$

where $H(\cdot)$ denotes the entropy of a distribution, and $\alpha \in \mathbb{R}$ is a hyperparameter that scales exploration pressure. The entropy term prevents the policy from converging to a narrow distribution, unless a narrow policy distribution can achieve large gains in expected cumulative return. Thereby, it applies a similar principle as Boltzmann exploration, gradually weighting exploration based on the returns of competing actions.

Compared to random perturbation, the benefit of this approach is that it gradually starts to prefer actions with better returns. On the downside, it does not track any measure of progress (or remaining uncertainty), and therefore cannot assess whether learning has converged, or whether we need additional information. It also depends on the relative scale of the rewards, and can therefore involve tedious tuning of hyperparameters.

– *Action value uncertainty*: A popular approach to exploration uses the remaining uncertainty in the value estimates of the available actions. With high uncertainty around our estimate there is still reason to explore, while reducing uncertainty should gradually shift our policy towards exploitation. A popular uncertainty-based approach are upper confidence bound (UCB) methods (Auer, 2002; Kaelbling, 1993; Kocsis and Szepesvári, 2006; Silver et al., 2017c), which for example explore like:

$$\pi(a|s) = Q(s,a) + c \cdot \sqrt{\frac{\ln(n(s))}{n(s,a)}}, \tag{3.3}$$

where $n(\cdot)$ denotes the number of visits to a state or state-action pair, and $c \in \mathbb{R}$ is a hyperparameter that scales exploration. A popular Bayesian approach to select actions under

uncertainty is Thompson sampling (Thompson, 1933). Again, we may want to correlate noise over timesteps, for example by sampling from the value function posterior once at the beginning of a new episode (Osband et al., 2016).

We also consider *pruning* an uncertainty-based method. In certain scenarios, we can completely eliminate an action from the candidate set because we are absolutely certain that it can never outperform an already visited action. This is a form of 'hard uncertainty'. It for example occurs in two-player games with a minimax (Edwards and Hart, 1961; Knuth and Moore, 1975) structure. Indeed, the *soft pruning* techniques developed in the search community in the early '80 (Berliner, 1981) can be regarded as early confidence bound methods.

Finally, note that due to the sequential nature of the MDP problem, value uncertainty in a MDP is more complicated than in the bandit setting. In particular, the remaining uncertainty is not only a function of the number of trials at a state-action pair, but also depends on the remaining uncertainty in the value estimates of the state-action pairs that follow it. See Dearden, Friedman, and Russell (1998), Moerland, Broekens, and Jonker (2017a, 2018b), and Osband et al. (2016) for a further discussion of this topic.

– *Priors*: In some cases we may have access to specific prior information about the value function, which then implicitly encodes exploration pressure. The prime example is an *admissible* heuristic. An admissible heuristic provides for every state an optimistic estimate of the cumulative return under the optimal policy. The closer the heuristic is to the true action value, the more prior information about exploration potential we get.

  The classic example of a good admissible heuristic is the Euclidean distance to the goal in a path planning task, which can for example be used in A* (Hart, Nilsson, and Raphael, 1968). An admissible heuristic actually provides informative exploration information, as it directly gives an estimate of the remaining value of a node, which may therefore become promising for exploration (or not). However, in most problems an admissible heuristic in not easy to obtain. See for example Kolobov (2012) for a more extensive discussion of planning on MDPs and possible heuristics.

- **State-based exploration**: The third main approach to exploration uses state-dependent properties to inject exploration noise, i.e., independently of the value of a particular state action. We again list the most important approaches:
  - *Ordered*: First of all, we may simply give every state-action a finite probability of selection. This is better known as a *sweep*. In Dynamic Programming (Bellman, 1966) the sweep is ordered based on the state-space structure, while in exhaustive search (Russell and Norvig, 2016) the sweep is ordered based on a tree structure. Note that a DP sweep is fully exploratory, since it visits every state-action pair in a fixed order, independently of greedy policies.

    We also consider *random starts* to be part of this category. Random starts, where our agents starts each new trial at a random state, is part of several classic RL convergence proofs (Barto, Bradtke, and Singh, 1995; Watkins and Dayan, 1992). It ensures that we visit every state eventually infinitely often. Although randomized, it is conceptually close to the DP sweeps, since it ensures that we visit every state-action infinitely often in the limit. We therefore consider it a state-based exploration method, with random ordering.
  - *Priors*: The state-based variant of prior information is better known as *shaping*. The best known example are *shaping rewards* (Ng, Harada, and Russell, 1999), which are additional rewards placed at states that are likely part of the optimal policy. Recent examples that include this approach are AlphaStar (Vinyals et al., 2019) and For The Win (Jaderberg et al., 2019), who use intermediate game scores as shaping rewards, and optimize the relative weight of each shaping reward in a meta-optimization loop.

    Another form of state-dependent priors that guide exploration are *expert demonstrations*, which help to initialize to a good policy. This approach was for example used in the first version of AlphaGo (Silver et al., 2016). While these examples use completely task-specific shaping, we can also find more generic shaping priors. For example, *objects* are generally salient in a task, and children are indeed able to discriminate objects in early infancy. Kulkarni et al. (2016) equip the RL agent with a pre-trained object recognizer, and subsequently

places shaping rewards at all detected objects in the scene, which is a more generic form of reward shaping.

– *Novelty*: As discussed before, uncertainty and novelty can be important primitives for exploration. While value-based uncertainty methods use the uncertainty around a value, there are also approaches that use the novelty of the state itself, independently of its value. An example is *optimistic value initialization* (Sutton and Barto, 2018), where we initialize every state-action estimate to a value higher than the maximum achievable return, which ensures that we initially prefer unvisited actions.

  A more formal approach to novelty is the *Probably Approximately Correct* in MDP (PAC-MDP) framework (Kakade et al., 2003). These approaches provide sample complexity guarantees for a RL algorithm, usually based on notions of novelty, ensuring that every reachable state-action pair gets visited enough times. A well-known example is R-max (Brafman and Tennenholtz, 2002), which assumes that every transition in the MDP has maximum reward until it has been visited at least n times. Note that such approaches are generally not computationally feasible in large, high-dimensional state spaces.

– *Knowledge-based intrinsic motivation*: A large group of state-based exploration approaches is knowledge-based intrinsic motivation (Chentanez, Barto, and Singh, 2005; Oudeyer, Kaplan, and Hafner, 2007). Novelty, as discussed above, is also part of this group, but knowledge-based IM contains a broader set of concepts. The general idea is to provide rewards for events that are intrinsically motivating to humans. Examples include curiosity, novelty, surprise, information gain, reduced model prediction error, etc. These are state dependent properties, independent of the external reward function. For example, depending on the interaction history of an agent, a certain transition can be surprising or not, the model prediction can be correct or completely off, etc. Knowledge-based IM approaches then provide an intrinsic reward for such events, based on the idea that good exploration requires us to decrease novelty, surprise and prediction error over the entire state-space. There is a plethora of different intrinsic motivation approaches (Achiam and Sastry,

2017; Bellemare et al., 2016; Dilokthanakul et al., 2019; Hester and Stone, 2012a; Houthooft et al., 2016; Lopes et al., 2012; Mohamed and Rezende, 2015; Pathak et al., 2017; Stadie, Levine, and Abbeel, 2015; Sun, Gomez, and Schmidhuber, 2011; Sutton, 1990)

– *Competence-based intrinsic motivation*: In RL, frontier-based exploration has been popularized under the name of competence-based intrinsic motivation (Oudeyer, Kaplan, and Hafner, 2007; Péré et al., 2018). Competence-based IM approaches try to explore by setting their own new goals at the border of their current abilities (i.e., their frontier). This approach typically involves three steps. The first step (goal space learning, for example based on variational auto encoders (Laversanne-Finot, Pere, and Oudeyer, 2018; Péré et al., 2018)) and third step (planning towards the sampled goal) are less relevant from an exploration perspective. The second step involves the exploration decision. We may for example select a new goal based on *learning progress* (Baranes and Oudeyer, 2013; Matiisen et al., 2017), selecting the goal which has shown the largest recent change in our ability to reach it. Otherwise, we can also train a generative model on the state that were of intermediate difficulty to reach, and sample a next goal from this model (Florensa et al., 2018). In any case, the prioritization is dependent on which states we managed to reach so far, and is therefore a form of state-based prioritization.

Note that the above groups are not mutually exclusive. For example, Ecoffet et al. (2019) introduces two methods to prioritize a frontier, one that estimates the amount of progress in the overall task (a value-based prior), and one that uses the visitation frequency of the state (a state-based novelty approach). In summary, we discussed two types of candidate sets (step-wise and frontier) and three approaches to exploration (random, value-based, state-based). Table 3.2 summarizes our discussion, by displaying common approaches on each of the possible combinations.

3.2.3.3 *One versus two phase exploration*

The straightforward implementation of the above ideas is to select one method and repeatedly apply it. This is what we call 'one phase exploration', where every step of trial selection uses the same method.

Table 3.2: Schematic overview of common trial selection methods. The columns display the *candidate set* selection method (Sec. 3.2.3.1), the rows display the way to inject *exploration* pressure to the greedy policy (Sec. 3.2.3.2). Each cell shows some illustrative example papers. IM = Intrinsic Motivation. PAC-MDP = Probably Approximately Correct in Markov Decision Process (Kakade et al., 2003).

	Step-wise	**Frontier**
Random	- *Random perturbation*, e.g., ϵ-greedy (Mnih et al., 2015) Gaussian noise	- *Random sampling on frontier*
Value-based	- *Mean value*: e.g., Boltzmann (Mnih et al., 2015), entropy regularization (Peters, Mulling, and Altun, 2010) - *Uncertainty*: e.g., confidence bounds (Kaelbling, 1993), posterior sampling (Thompson, 1933)	- *Priors*, e.g. A* (Hart, Nilsson, and Raphael, 1968)
State-based	- *Ordered*: e.g., DP (Bellman, 1966) - *Priors*: e.g., reward shaping (Ng, Harada, and Russell, 1999) - *Novelty*: e.g., optim.init. (Sutton and Barto, 2018), PAC-MDP (Brafman and Tennenholtz, 2002) - *Knowledge-based IM*, e.g., (Achiam and Sastry, 2017)	- *Competence-based IM*, e.g. (Péré et al., 2018)

However, some approaches extend this idea to two distinct phases. It is inspired by the way humans plan and act in the real world, where we typically first plan in our head, and then decide on an action in the real world. The two phases therefore are:

1. *Plan*: repeatedly plan ahead from the *same* state, which is the root of the plan.

2. *Real step*: decide on an action at the root, move forward to the next state, and repeat planning with the resulting state as the new root.

In the planning literature, this scheme is often referred to as *receding horizon control* (Mayne and Michalska, 1990), or *model predictive control* (Morari and Lee, 1999). However, in planning the real step is always greedy (we want to execute the best planned action), and the real step never contains any exploration. In model-free RL, we never have the planning step available, and therefore push all exploration in the real

step. However, in model-based RL, we approximate a value or policy over many episodes, and it does make sense to explore in both the plan and in the real step, usually in a different amount (where the real step is typically more greedy).

When we do model-based RL with in an irreversible environment, like the real world, then the two phases of exploration are inevitable. However, interestingly, some model-based RL methods with a known model still voluntarily use the two phase scheme, like AlphaZero (Silver et al., 2018). AlphaZero does not need to make a real step, since it could also try to solve for the entire solution of Go from the root. However, it nevertheless introduces 'real steps' (executed in the model). First, this heuristically prunes away parts of the search tree, since solving the entire problem from the root is simply not feasible (it reduces to exhaustive search). Second, a real step also cleans the search tree, which may have started to accumulate errors when we deal with learned transition or value functions. Real steps therefore seem a necessary aspect of exploration in larger problems.

We may ask ourselves whether Dyna (Sutton, 1990) uses one- or two-phase exploration? Between trials in the real environment, Dyna samples additional data from its learned, reversible model. Typically, it uses the same type of exploration policy and back-up for the additional samples as for data acquired from the real environment. Therefore, it is clearly one-phase in our definition. Multiple phases refers to the use of *different* exploration policies from the *same* state within one algorithm, but does not depend on the order in which we update states.

When a reversible model is not available but should be approximated from data, two-phase exploration is primarily studied as *Bayes-adaptive exploration* (Guez, Silver, and Dayan, 2012). In this approach, we first learn a Bayesian dynamics model from the available data. Then, we plan to solve for the optimal action, while we average out over all uncertainty in the dynamics model. We then execute this action, collect new real world data, and repeat the above procedure. This is a provably optimal approach to achieve high data efficiency in the real environment (Guez, Silver, and Dayan, 2012), but comes at the expense of high computational burden.

3.2.3.4 *Reverse trials*

Finally, there is a different approach to trial selection based on the idea of reverse trials. All previous approaches take a forward view on trials, utilizing information about the state-actions in the candidate

set obtained from previous trials. However, we can also identify a promising state-action pair for the next trial based on a change in the value of its child state. For this section, we will denote a child of (s, a) as (s', a'). If we reached (s', a') through another trace (not including (s, a)), and the estimate of (s', a') changed a lot, then it is likely that our estimate of (s, a) should be updated as well. In other words, if we learned that a certain state-action pair is good, then we can look back at all the state-action pairs that could bring us here, and update their estimates as well. This idea is better known as *prioritized sweeping* (Moore and Atkeson, 1993).

Prioritized sweeping is actually a special form of a candidate set, but since it is so conceptually different from the rest of the discussion (it requires a reverse model), we nevertheless discuss it separately. The key of prioritized sweeping is a *reverse* model $T^{-1}(s, a|s')$, which tells us which (s, a) can lead to s'. Given a change in some $Q(s', a')$, prioritized sweeping evaluates the *priorities* $\rho(s, a)$ of all possible precursors of s' based on the one-step temporal difference:

$$\rho(s,a) = T(s'|s,a) \cdot \left| R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right| \quad \forall \quad (s,a) \sim T^{-1}(s,a|s'). \tag{3.4}$$

Here, we use T and R for the learned transition and reward functions, although the principle equally applies to the ground-truth functions $\mathcal{T}$ and $\mathcal{R}$. When the priority $\rho(s, a)$ exceeds some small ϵ, we add it to the queue. We then update the state action pair on the top of the queue, and repeat the above procedure for a fixed budget, after which we make a new forward trial.

While forward trials try to figure out where good pay-off may be present further ahead in the MDP, backward trials try to spread the information about an obtained reward as quickly as possible over the nearby states in state space in reverse order. This is graphically illustrated in Figure 3.4. Note the difference between multi-step methods, which quickly propagate rewards along the same forward trace, and prioritized sweeping, which spreads to all possible precursors.

3.2.4 *How to estimate the cumulative return?*

Once we have selected a trial, we obtain a sampled next state (or a distribution over possible next states) and its associated reward. However,

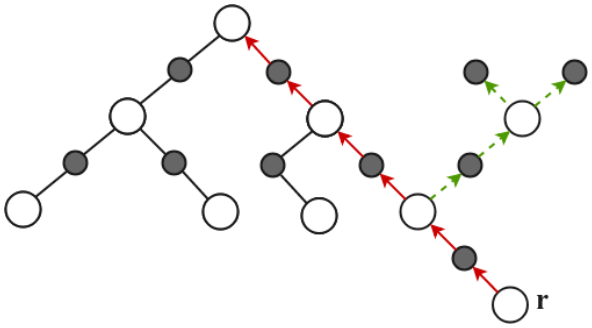


Figure 3.4: Prioritized sweeping. Regular back-ups are applied in reverse direction of the forward trials (red solid arrows). Prioritized sweeping acknowledges that new reward/value information may also affect other states that lead to a specific outcome. By learning a reverse model, $\hat{T}^{-1}(s, a|s')$, we may identify states in the reverse direction that are candidates for updating (green dashed arrows). The visualization shows that prioritized sweeping can be interpreted as building a new tree in the reverse direction, to spread the obtained reward information more quickly.

we are not interested in only the single reward of the transition, but actually in the *cumulative return*. The quantity that we need is actually visible in the one-step Bellman equation:

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{T}} \Big[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi} [Q(s_{t+1}, a_{t+1})] \Big]. \tag{3.5}$$

In the next section (on back-ups) we will discuss how to deal with the two expectations in Eq. 3.5. However, we will first discuss how to get an estimate of $Q(s_{t+1}, a_{t+1})$, i.e., the remaining cumulative reward after the trial. Likely, there is a large subtree below (s_{t+1}, a_{t+1}), which we can not fully enumerate.

The general form of the cumulative reward estimate takes the following form:

$$\hat{Q}_{\text{K-step}}(s_t, a_t) = \sum_{k=0}^{K-1} \gamma^k \cdot r_{t+k} + \gamma^K B(s_{t+K}), \tag{3.6}$$

where $K \in \{1, 2, 3, .., \infty\}$ denotes the *sample depth* and $B(\cdot)$ is a *bootstrap function*. These are the two key considerations of cumulative reward estimation, which we discuss below.

3.2.4.1 *Sample depth*

We first need to determine the sample depth n.

- $\boldsymbol{K = \infty}$: A quick way to get an estimate of the cumulative return after the first reward r_t is to sample a deep sequence of trials, and add all the rewards in the trace. In this case ($K \to \infty$, better known as a *Monte Carlo roll-out*) we do not bootstrap. Although a Monte Carlo roll-out gives an unbiased estimate of the value of the entire remaining subtree, it does have high variance, as we sampled only one realization of all the possible traces. Monte Carlo targets are for example commonly used in MCTS (Browne et al., 2012).

- $\boldsymbol{K = 1}$: On the other extreme we directly bootstrap after the trial. One-step targets have low variance but are biased, since the bootstrap estimate can have bias. The bootstrapping function will be discussed in the next section. Well-known algorithms that bootstrap after a single step are for example Q-learning (Watkins and Dayan, 1992) and A* (Hart, Nilsson, and Raphael, 1968).

- $\boldsymbol{K = n}$: We can also use an intermediate value for K, which is known as an $n - step$ target, for $1 < n < \infty$.

- **Reweighted**: We can also combine/reweight targets of different depths. Examples include eligibility traces (Schulman et al., 2016; Sutton and Barto, 2018) and more sophisticated reweighting schemes based on importance sampling (Munos et al., 2016).

3.2.4.2 *Bootstrap function*

When we stop sampling, we can plug in a fast estimate of the value of the remaining subtree, denoted by $B(\cdot)$ in Eq. 3.6. This idea is called bootstrapping. There are two main functions to bootstrap from:

- **Learned value function**: We can learn the function to bootstrap from. The ideal candidate is the state value function $V(s)$ or state-action value function $Q(s, a)$. These value function may also serve as the solution representation (see Sec. 3.2.6), in which case they serve two purposes. But also when we represent the solution with a policy, we may still want to learn a value function to bootstrap from.

- **Heuristic**: The second bootstrap approach uses a heuristic (value) function (Pearl, 1984), which is a form of prior information. An

admissible heuristic $H(s)$ or $H(s,a)$ gives an optimistic estimate of the cumulative return from a particular state or state-action pair. In many tasks it is hard to obtain a good admissible heuristic, since it should always be optimistic, but should not overestimate the return by too much, as otherwise it is of little benefit. In some planning settings we can obtain a good heuristic by first solving a simplified version of the problem, for example by making a stochastic problem deterministic (Yoon, Fern, and Givan, 2007).

In summary, we need to choose both a sample depth and bootstrap function to obtain a cumulative reward estimate. Note that a Monte Carlo roll-out is actually a deep sequence of trials. We can of course form value estimates for other state-actions in the trace as well, but our framework focuses on one particular state-action pair that we want to update. Note that we can also make a combined value estimate, for example from two Monte Carlo roll-outs, or from a depth-d limited search. However, these methods simply repeatedly apply the above principle, for example at the leafs of the depth-limited search. How to combine multiple cumulative reward estimates is part of the next section, on the back-up.

3.2.5 *How to back-up?*

The trial at s_t, a_t gave us a reward r_t, a next state s_{t+1} (or distribution over next states), and an estimate of the cumulative return. We now wish to back-up this information to improve our estimate of the value at s_t, a_t. In Eq. 3.5, we still need to specify i) which policy to specify for the back-up, ii) how to deal with the expectation over the actions, and iii) how to deal with the expectation over the dynamics. We will discuss each of them.

3.2.5.1 *Back-up policy*

We can in principle specify any back-up policy $\pi^{\text{back}}(a|s)$, which may differ from the forward policy $\pi^{\text{for}}(a|s)$ which we used for trial selection. When $\pi^{\text{back}}(a|s)$ equals $\pi^{\text{for}}(a|s)$ we call the back-up *on-policy*. In all other cases, the back-up is *off-policy*. The cardinal example of an off-policy back-up is the *greedy* or *max* back-up policy, which greedily selects the best action. A benefit of greedy back-ups is that they learn the optimal policy, but they can be unstable in combination with function approximation and bootstrapping (Sutton and Barto, 2018). Some

authors study other forms of off-policy back-ups (Coulom, 2006; Keller, 2015), for example more greedy than the exploration policy, but less greedy than the max operator.

3.2.5.2 *Expectation over the actions*

Given the back-up policy, we can either make a *sample* or *expected* back-up. A sample back-up samples from the policy, and backs up the value behind this particular action. Sample back-ups are computationally cheap, but need multiple samples to converge to the true value. In contrast, expected back-ups exactly evaluate the expectation over the actions. Sample back-ups are for example used in SARSA (Rummery and Niranjan, 1994), while expected back-ups are used in Expected Sarsa (Van Seijen et al., 2009) and (off-policy) in Tree Backup (Precup, 2000).

3.2.5.3 *Expectation over the dynamics*

Similar to the expectation over the actions, there are two main ways to deal with the expectation over the dynamics: sample or expected. When the exact transition probabilities are available, then we can exactly evaluate the expectation. Otherwise, when we only have access to an irreversible environment or to a generative model (given or learned), we make a small step in the direction of the sampled value, which will converge to the true value in over multiple back-ups. Although sample-based back-ups provide less information, they can actually be more efficient when many next states have a very small probability (Sutton and Barto, 2018). A special case are deterministic dynamics functions, for which the expected and sample update are equivalent.

The three categories together give rise to several back-up types, as visualized in Figure 3.5. The vertical axis shows the back-up over the dynamics, while the horizontal axis shows the back-up policy and (nested) the way to deal with the action expectation. In the example, the off-policy back-up (right column) is illustrated by the greedy policy. For the off-policy greedy back-up, the sample and expected action methods are the same, so the right column shows only a single graph centered in the column. For completeness, we list the associated back-up equations below:

- Bellman back-up:

$$\hat{Q}(s,a) = \mathbb{E}_{s' \sim \mathcal{T}(s'|s,a)}[\mathcal{R}(s,a,s') + \gamma \cdot \mathbb{E}_{a' \sim \pi(a'|s')} Q(s',a')]$$

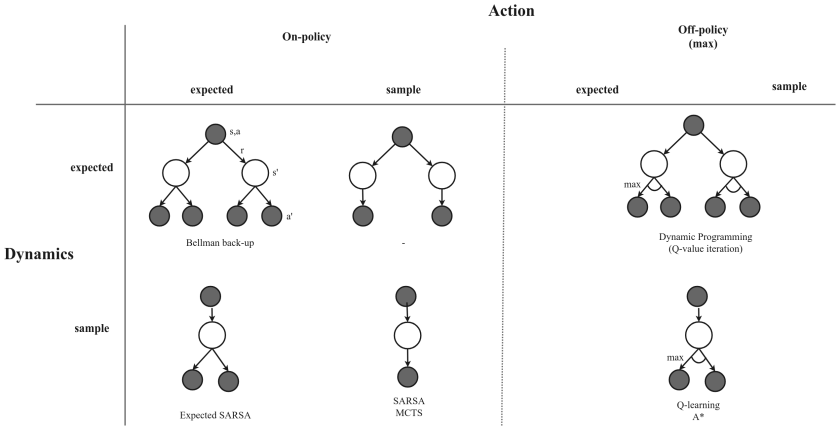


Figure 3.5: Variants of 1-step back-up. The associated equations are listed in the main text. Mentioned algorithms/back-ups include Value Iteration (Sutton and Barto, 2018), Bellman back-up (Bertsekas, 1995), Q-learning (Watkins and Dayan, 1992), Expected SARSA (Van Seijen et al., 2009), SARSA (Rummery and Niranjan, 1994), A* (Hart, Nilsson, and Raphael, 1968) and MCTS (Kocsis and Szepesvári, 2006).

- Q-value iteration:

$$\hat{Q}(s,a) = \mathbb{E}_{s' \sim \mathcal{T}(s'|s,a)}[\mathcal{R}(s,a,s') + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s',a')]$$

- Q-learning:

$$\hat{Q}(s,a) = \mathcal{R}(s,a,s') + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s',a'), \text{ for } s' \sim T(\cdot|s,a)$$

- SARSA:

$$\hat{Q}(s,a) = \mathcal{R}(s,a,s') + \gamma \cdot Q(s',a'), \text{ for } s' \sim T(\cdot|s,a), a' \sim \pi(\cdot|s')$$

- Expected SARSA:

$$\hat{Q}(s,a) = \mathcal{R}(s,a,s') + \gamma \cdot \mathbb{E}_{a' \sim \pi(a'|s')}[Q(s',a')], \text{ for } s \sim T(s,a,s')$$

We can now look back at the cumulative reward estimation methods from the previous section, and better interpret the Monte Carlo estimate. A MC roll-out is effectively a long sequence of trials, followed by sample-transition, sample-action, on-policy back-ups. So these trials indeed have their own specific back-ups to aid in the update of the root state-action pair under consideration.

3.2.6 *How to represent the solution?*

The back-up gave us an improved estimate of the value or policy at some (s, a). However, we have not discussed yet how the solution will actually be represented. There are two main considerations. First, we need to decide what function we will represent. Second, we need to decide how to represent it in memory.

3.2.6.1 *Function type*

There are several ways in which we can represent the solution:

- **Value**: A common solution form is a value function, typically in the form of a state-action values $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This function estimates the value of the current or optimal policy at all considered state-action pairs. For action selection, a value function representation requires a mapping from action values to selection probabilities, like an ϵ-greedy or Boltzmann policy.
- **Policy**: A policy $\pi : \mathcal{S} \to p(\mathcal{A})$ maps every state to a probability distribution over actions. A benefit of learning a policy is that we can directly act in the environment by sampling form the policy distribution. Therefore, this is usually the preferred approach for continuous action spaces, since a max over a continuous action value space requires a nested optimization before we can act.
- **Both**: Some approaches store both a value and a policy function, better known as *actor-critic* methods. The value function is typically used to aid the policy update (see Sec. 3.2.7).

GENERALIZED VALUE AND POLICY We may extend the above functions by also incorporating a goal that we attempt to reach, better known as *generalized* value or policy functions (Schaul et al., 2015). Generalized value and policy functions take as input the current state and some target/goal state that we would like to reach, and output the value or policy to reach that particular goal state. For example, $Q_g : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ would for a particular current state s_0, goal state s_g, and candidate action a, estimate the value $Q_g(s_0, s_g, a)$, under some reward function that increases when we get closer to s_g. The same principle applies to a generalized policy π_g. The main benefit of generalized value functions is our ability to return to any desired state in state space. The key underlying idea is that we may generalize in goal space, since

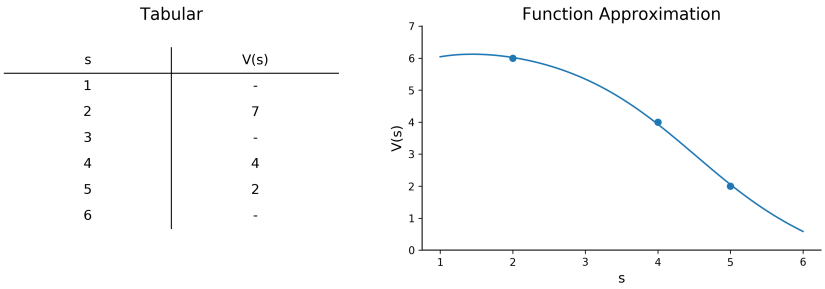


Figure 3.6: Representation of the solution. Example for a value function $V(s)$ on an one-dimensional, discretized state space. **Left**: Tabular representation. We have obtained estimates for states 2, 4 and 5, while 1, 3 and 6 have not been visited yet. **Right**: Function approximation. Dots indicate the observed data points, the solid line shows a neural network fit on these points. This function generalizes the information in the observations to other (nearby) states, like 1, 3 and 6.

nearby goals likely share much of their value functions and optimal policy to reach them. We further discuss the concept of generalization in the next section.

There are some others examples of solution representations. For example, some MCTS approaches make their real step decision based on the counts at the root, rather than the value estimates. Counts could be considered a separate function type as well. However, since it is a close derivative of the value, we treat it as a special case of a value function in our framework.

3.2.6.2 *Function class and generalization*

Once we determined which function to store, we have to decide how to actually represent it in memory. This topic is closely intertwined with the concept of *generalization*. At the top level, we can discriminate two groups of approaches: i) tabular methods, which do not generalize at all, and ii) function approximation methods, which do generalize. These two approaches are illustrated in Figure 3.6, and will be discussed in greater detail below:

- **Tabular**: The tabular representation, also known as *atomic*, *symbolic* or *list* representation, treats each input (e.g., state) as a unique element for which we store an individual estimate (e.g., value)

(Fig. 3.6). Tabular representations are the dominant approach in the planning literature, as the nodes in a tree are essentially tables. Note that in a tree, the same state may appear multiple times, like a table with duplicate entries. In the planning literature, a solution to this problem are *transposition* tables, which share information between multiple occurrences of the same state.

For tables, there is an additional important distinction based on the storage duration. On the one hand, *global* tables store the value or policy function for the entire state-action space. However, such a table usually does not fit in memory in large problems. On the other hand, we find *local* tables in planning methods like MCTS (Browne et al., 2012). The local table is temporarily built during the search, but after making a real step we throw away the part of the tree that is no longer relevant in this episode. Another example of local tables are trajectory optimization methods, as used in optimal control, which output a single trajectory (discretized in time).

- **Function approximation**: The second representation approach is function approximation, which builds on the concept of generalization. Generalization implies that similar inputs (states) to a function will in general also have approximately similar output (policy or value) predictions. Thereby, function approximation allows us to share information between near similar states, which a table can not do. A second benefit of function approximation is that due to the approximation we can store a global solution for large state spaces, for which a table would grow too large.

  There is a variety of function approximation methods in the machine learning literature. At a high-level we can discriminate parametric methods, like neural networks, and non-parametric methods, like k-nearest neighbours. We will focus on parametric methods here, since these have received most attention and shown most success recently. The most popular groups of parametric function approximation methods are deep neural networks (Goodfellow, Bengio, and Courville, 2016). For example, if we decide to learn a state-action value function, then we specify the function class $Q_\theta : \mathcal{S} \times \mathcal{A} \times \Theta \to \mathbb{R}$ with parameters $\theta \in \Theta$. Our aim is to chose the parameters θ in such a way that they accurately predict the state-action estimates obtained from the back-up (as discussed in the previous section).

> Note that generalization is actually a spectrum, with complete over-generalization on one extreme, and no generalization at all on the other end. Tabular methods are one extreme, since they do not generalize at all. But once we enter function approximation methods, we still need to balance the amount of generalization to the actual data, better known as balancing overfitting and underfitting.

Reinforcement learning methods have mostly focused on function approximation, while planning has mostly emphasized tabular representation. Tabular methods are easy to implement, stable (since an update to one state action pair does not affect other pairs), and provide good separation between neighboring states. However, their biggest limitation is the memory requirement, which scales exponentially in the dimensionality of the state and action space. Therefore, global solution tables are not feasible in large problems, while local tables cannot profit from offline learning of a global solution.

The benefits of function approximation (generalization, and lower memory requirements) were already mentioned above. Especially generalization can be crucial in large state spaces, where we seldom visit exactly the same state twice, but often encounter approximate similarity. A problem of function approximation is instability, since a local training step also affects the predictions of state-action pairs around it. This is less of a problem in supervised learning with a fixed training set, but since RL collects its own data, a deviation in the policy or value may cause the agent to never explore a certain area of the state-space again. *Replay databases* (Lin and Mitchell, 1992; Mnih et al., 2015) are a way to battle this instability, by reducing the correlation between training data points.

There are some preliminary indications that the combination of function approximation and local tabular methods may actually provide the best of both worlds (Moerland et al., 2020; Wang et al., 2019). These ideas are inspired by for example AlphaGo Zero (Silver et al., 2017c), Guided Policy Search (Levine and Koltun, 2013) and Dyna-2 (Silver, Sutton, and Müller, 2008), which nest a local tabular planning method in a learning loop. The hypothesis here is that local tabular planning smooths out errors in the global value approximation, while the global approximation provides the necessary information sharing and generalization for the planning method to be effective.

Finally, note that the representations also need to be initialized. The most common approaches are random (for function approximation) or

uniform (for tables) initialization. Optimistic initialization, where we initialize all state action value estimates above the maximum achievable return, actually adds an exploration aspect to the initialization, and was already discussed in Sec. 3.2.3.2.

3.2.7 *How to update the solution?*

The last step of our framework involves updating the solution representation (from Sec. 3.2.6) based on the backed-up value estimates (from Sec. 3.2.5). While we have mostly discussed value estimation so far, we will now suddenly see policy functions appear more often. The section is split up in two parts. First, we discuss *losses*, which define the way in which our solution can be improved based on the backed-up value. Second, we discuss *update rules*, which can either be gradient-based or gradient-free. The first part on losses only applies to the gradient-based updates.

3.2.7.1 *Loss*

A *loss* is the main approach in learning to specify the objective. We choose a loss in such a way that when we minimize it, our solution improves. Therefore, it is usually a function of both the solution ($Q_\theta(s,a)$ or $\pi_\theta(a|s)$) and the back-up estimate ($\hat{Q}(s,a)$). Below we discuss some common losses for both value and policy.

VALUE LOSS

- **Squared loss**: The most common loss for value function representations is the *mean squared error*. The squared error loss is

$$L(\theta|s_t,a_t) = \frac{1}{2}\Big(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)\Big)^2. \tag{3.7}$$

We may also use other losses than the squared loss, as long as minimization of the objective moves our solution closer to the new estimate. For example, Hamrick et al. (2020) recently successfully used a cross-entropy loss between the softmax of the Q-values from the search and the softmax of the Q-values from the value approximation. However, the squared loss is by far most common. In the next section we will show that many common planning updates implicitly use the squared loss as well.

POLICY LOSS There are various ways in which we may specify a policy loss:

- **Policy gradient**: One way to update the policy from a backed-up value estimate is based on the *policy gradient theorem* (Sutton and Barto, 2018; Sutton et al., 2000; Williams, 1992). The theorem gives an unbiased estimator of the gradient of our overall objective (the cumulative reward achieved from the start state):

$$\nabla_\theta V(s_0) = \mathbb{E}_{\pi_\theta, \mathcal{T}}\Big[\sum_{t=0}^{\infty} Q(s_t, a_t) \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t)\Big], \tag{3.8}$$

  where the expectation runs over all traces induced by π_θ and $\mathcal{T}$. In practice, the above gradient implicitly specifies a loss. For example, when we use automatic differentiation software, we would implement the policy gradient by sampling traces and minimizing, at every visited state-action pair, the loss

$$L(\theta|s_t, a_t) = -\hat{Q}(s_t, a_t) \cdot \ln \pi_\theta(a_t|s_t) \tag{3.9}$$

  with respect to θ. This last equation clearly shows what the policy gradient equation actually does. It specifies a relation between value estimates $\hat{Q}(s_t, a_t)$ and the policy $\pi_\theta(a_t|s_t)$. If we minimize the above objective, we effectively ensure that actions with a high value also get a high policy probability assigned (since the policy needs to integrate to 1). Note that the policy gradients is not a first-order derivative, but rather a zero-order gradient that tells whether we should move the probability of a certain action up or down. The close relationship between value back-ups and policy gradients is also illustrated by Schulman, Chen, and Abbeel (2017).

- **Deterministic policy gradient**: Another popular way to improve a policy based on value estimates is based on deterministic policy gradients (Lillicrap et al., 2015; Silver et al., 2014). These approaches first train a value function based on the methods of the previous paragraph. When we ensure that the learned value function is differentiable with respect to the input action, then we can update the policy by differentiation through the policy action. The associated loss is simply

$$L(\theta|s_t, a_t) = -Q_\psi(s, \pi_\theta(a|s)), \tag{3.10}$$

where we introduced ψ for the value function parameters, to make clear that the loss is with respect to the policy parameters.

- **Value gradient**: When we have a differentiable reward, transition and policy function, then we can treat our back-up value as a single computational graph, which we can directly optimize (illustrated in Figure 3.7). This approach is typically applied to sampled traces, for example in PILCO (Deisenroth and Rasmussen, 2011). After sampling a trace, our loss is simply the negative cumulative return:

$$L(\theta|s_t, a_t, .., s_\infty) = -\hat{Q}(s, a) = -\sum_{t=0}^{\infty} r_t. \tag{3.11}$$

  We call this objective the value gradient loss. The associated update will be discussed in the next section. Note that the above objective uses an on-policy, sample-action, sample-dynamics back-up, with a sample depth of ∞ and no bootstrapping. However, with a differentiable value function we could also use bootstrapping,and differentiate through the value function as well.

- **Cross-entropy policy loss**: Again, we can in principle come up with any type of policy loss that increase the probability of action that have comparatively higher value estimates. For example, AlphaGo Zero (Silver et al., 2017c) makes a heuristic decision for the policy loss. Their MCTS planning procedure returns value estimates and visitation counts at the root of the search. The counts are closely related to the backed-up value estimates in the search, as nodes with higher value estimates get more visits. They propose to normalize the visitation counts to a probability distribution, and train the policy network on a cross-entropy loss with this distribution:

$$L(\theta|s_t) = -\sum_{a \in \mathcal{A}} \log \pi_\theta(a|s_t) \Big(\frac{n(s_t, a)}{\sum_b n(s_t, b)} \Big), \tag{3.12}$$

  where $n(s_t, a)$ denotes the number of visits to action a at the MCTS root s_t.

The last example illustrates that heuristically motivated losses can work well in practice. The list is by no means exhaustive, and could, for example, also incorporate approaches that cast sequential decision making as an inference problem (Botvinick and Toussaint, 2012; Kappen, Gómez, and Opper, 2012; Toussaint, 2009). The choice of a particular loss function may also depend on the setting. For example, value gradients work well in tasks with relatively smooth transition and reward functions, like robotics and control tasks, but have trouble in sparse reward tasks. In short, there is a variety of possible losses for both value and policy targets.

3.2.7.2 *Update rule*

The final step of our framework is to actually update our representation. We identify two main approaches: gradient-based (which uses one of the losses of the previous section) and gradient-free optimization.

GRADIENT-BASED UPDATES Most learning approach perform gradient-based optimization. The general idea of gradient-based optimization is to repeatedly update our parameters in the direction of the negative gradient of the loss with respect to the parameters:

$$\theta \leftarrow \theta - \alpha \cdot \frac{\partial L(\theta)}{\partial \theta}, \tag{3.13}$$

where $\alpha \in \mathbb{R}^+$ is a learning rate. We will illustrate some examples:

- *Value update on table*: For a tabular value representation, the θ are simply all the individual table entries $Q_\theta(s,a)$. The derivative of the squared loss (Eq. 3.7) then becomes

$$\frac{\partial L(\theta)}{\partial \theta} = 2 \cdot \frac{1}{2}\Big(Q_\theta(s,a) - \hat{Q}(s,a)\Big) = Q_\theta(s,a) - \hat{Q}(s,a). \tag{3.14}$$

  Plugging this into Eq. 3.13 and reorganizing terms gives the well-known tabular learning rule:

$$Q_\theta(s,a) \leftarrow (1-\alpha) \cdot Q_\theta(s,a) + \alpha \cdot \hat{Q}(s,a). \tag{3.15}$$

  Note again that this update rule is actually the gradient update of a squared error loss on a value table. This update also makes

intuitive sense: we move our table entry $Q_\theta(s,a)$ a bit in the direction of our new estimate $\hat{Q}(s,a)$. Therefore, for the tabular case, we want to keep $\alpha \in [0,1]$.

We shortly discuss two special cases of the tabular learning rule, which both frequently occur in the planning community:

- Replace update: The *replace update* completely replaces the table entry with the new back-up estimate. In Eq. 3.15, this happens when we set $\alpha = 1$. In that case, it reduces to

$$Q_\theta(s,a) \leftarrow \hat{Q}(s,a). \tag{3.16}$$

  This effectively overwrites the solution with the new estimate obtained from the back-up. We can only afford to do this when we have some guarantees that our new estimate will always improve over our previous estimate. This does specifically happen when we have prior information, like an admissible heuristic. The replace update is for example used in $A^\star$ (Hart, Nilsson, and Raphael, 1968) planning. When such information is available, replace updates can be much faster than learning updates, which are relatively slow to converge. For example, for route planning on a map (where the euclidean distance in a good admissible heuristic), we would always prefer A* over Q-learning (Watkins and Dayan, 1992).

- Averaging update: The *averaging update*, the second special case of the tabular learning update, ensures that our table entry will remain equal to the *the mean of all previous back-up estimates*. We introduce n to index the update iteration. Then the update rule at every iteration that tracks the average is

$$Q_\theta(s,a) \leftarrow \frac{n-1}{n} Q_\theta(s,a) + \frac{1}{n} \hat{Q}(s,a). \tag{3.17}$$

  Comparing the above to Eq. 3.15, we see that the averaging update is actually a learning update with $\alpha = \frac{1}{n}$. In other words, we make the learning rate a function of the iteration number. The averaging update is for example the standard approach in MCTS (Browne et al., 2012).

The benefit of the averaging update is that it quickly moves the table entry to a reasonable estimate. After the first iteration, the estimate directly equals the first back-up estimate (while a learning update takes many small steps to move our predictions towards the true estimate). On the downside, fixed learning rates do eventually wash out the effect of the initial estimates, which are typically less reliable. In contrast, averaging updates will always give the initial estimate as much contribution to the table entry as the most recent back-up estimate.

- *Value update with function approximation*: The same principles apply for gradient-based updates in the context of function approximation. If our function approximator is differentiable, then we can simply apply the chain rule to again find the derivative of the loss with respect to the parameters. For example, training a value function approximation on a squared loss (Eq. 3.7) would have a gradient of

$$\frac{\partial L(\theta|s,a)}{\partial \theta} = \Big(Q_\theta(s,a) - \hat{Q}(s,a)\Big) \cdot \frac{\partial Q_\theta(s,a)}{\partial \theta}, \tag{3.18}$$

  where $\frac{\partial Q_\theta(s,a)}{\partial \theta}$ are for example the derivatives in a neural network.

- *Policy update with function approximation*: The same chain rule principles apply to the policy gradient loss, and also to the deterministic policy gradient. For example, for the deterministic policy gradient we have:

$$\frac{\partial L(\theta|s,a)}{\partial \theta} = -\frac{\partial Q_\psi(s,a)}{\partial \theta} = -\frac{\partial Q_\psi(s,a)}{\partial a}\frac{\partial \pi_\theta(a|s)}{\partial \theta}, \tag{3.19}$$

  where we again write ψ for the value parameters to distinguish them from the policy parameters.

- *Policy update for value gradient*: Gradient-based planning, better known as value gradients (Heess et al., 2015), is a special case of a policy update. When we have a differentiable dynamics and reward model, and specify a differentiable policy, then we can actually directly differentiate the cumulative reward estimate with respect to the policy parameters (Figure 3.7).

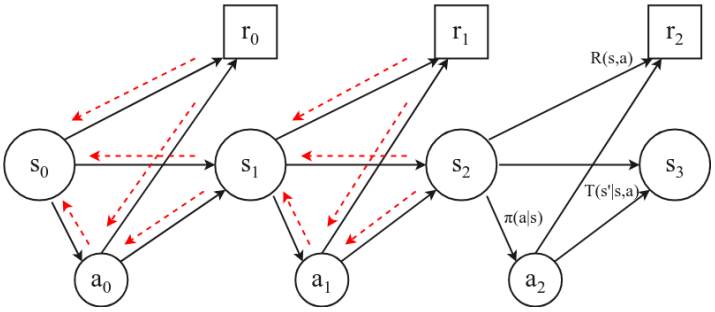


Figure 3.7: Illustration of value gradients. Black arrows show the forward specification of an MDP, with a reward function $\mathcal{R}(s,a)$, transition function $\mathcal{T}(s'|s,a)$, and our policy $\pi_\theta(a|s)$ to act in the MDP. If all of these functions are differentiable, then we can update the policy parameters θ by taking the gradient of the cumulative payoff $V(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$, with respect to these parameters, as indicated by the red dotted lines. This bears similarity to the way recurrent neural networks are trained with *backpropagation through time*.

We will here show the update equations for the gradient of the expected cumulative return $V(s) = \mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)|s_0 = s]$. To keep the update equations readable, we will for this equation abbreviate partial differentiation with subscripts, i.e., $V_s = \partial V(s)/\partial s$. The gradient of the sampled trace is given by the following set of recursive relations:

$$\begin{aligned} \hat{V}_\theta &= \mathcal{R}_a \pi_\theta + \gamma \hat{V}'_{s'} \mathcal{T}_a \pi_\theta + \gamma \hat{V}'_\theta, \quad \text{with} \\ \hat{V}_s &= \mathcal{R}_s + \mathcal{R}_a \pi_s + \gamma \hat{V}'_{s'} (\mathcal{T}_s + \mathcal{T}_a \pi_s). \end{aligned} \tag{3.20}$$

For every trace, the above gradient effectively sums over all paths in Figure 3.7. In practice we sample a single trace or finite set of traces to compute the gradients with respect to θ. Note the additional V'_θ term in the first equation, which appears since we need to sum the gradients with respect to θ at all timesteps.

Well-known examples of gradient-based planning are PILCO (Deisenroth and Rasmussen, 2011), which achieved high data-efficiency on real-world (small) robotic control tasks, and the linear-quadratic regulator (LQR) (Anderson and Moore, 2007; Todorov and Li, 2005). Gradient-based planning does rely on

smooth, differentiable dynamics functions, which makes it mostly applicable to continuous control tasks. Moreover, gradient propagation may suffer from vanishing and exploding gradients, as is also well-known for recurrent neural network (RNN) training.

There are two final remarks for gradient-based updates. First, all above methods have analytic gradients, but we may also use finite differencing to numerically approximate the gradient of our objective. This for example common in optimal control. Second, we have not defined yet how to choose the learning rate in Eq. 3.13. We neither want to progress too quickly nor too slowly. Most methods use a *line search* with manually tuned learning rate, but other approaches have been popularized in RL as well. A successful approach is to first determine a *trust region*, a region around the current solution in which we aim to search for the next solution, which is for example used in trust region policy optimization (TRPO) (Schulman et al., 2015) and proximal policy optimization (PPO) (Schulman et al., 2017) algorithms.

GRADIENT-FREE UPDATES We have extensively covered losses and learning-based updates. We will now also cover the competing approach, which uses gradient-free optimization. These approaches first specify a parametrized policy function. They then repeatedly: i) perturb the parameters in policy space, ii) evaluate the new solution by sampling traces, and iii) decide whether the perturbed solution should be retained. Example applications to MDP optimization include evolutionary strategies (Moriarty, Schultz, and Grefenstette, 1999; Salimans et al., 2017; Whiteson and Stone, 2006), simulated annealing (Atiya, Parlos, and Ingber, 2003) and the cross-entropy method (Mannor, Rubinstein, and Gat, 2003; Rubinstein and Kroese, 2013).

These methods largely bypass the other parts of our framework. They do not use any structural knowledge of the MDP, and never form local estimates of values for a particular state. Instead, they only require an evaluation function (sampling a set of traces), and treat the problem as a black-box optimization setting. There is extensive literature on gradient-free optimization methods, but these methods are not specific to planning and learning in MDPs, and therefore fall outside the scope of this framework.

This concludes our presentation of FRAP. The discussed dimensions, considerations per dimension, and choices per consideration were already summarized in Table 3.1. The next section will illustrate the

general applicability of FRAP, by analyzing a wide variety of planning and RL algorithms along the dimensions of the framework.

3.3 CONCEPTUAL COMPARISON OF WELL-KNOWN ALGORITHMS

The key point of FRAP is that planning and learning solve exactly the same problem, and therefore (implicitly) have to make decisions on all the dimensions mentioned in the framework. We illustrate this key idea in Table 3.3. The table shows, for a variety of well-known planning (blue), model-free RL (red) and model-based RL (green) algorithms, the choices each algorithm makes on the dimensions of FRAP.

The most important observation from the table is that it reads like a patchwork. On most dimensions, we see similar solution ideas appearing both within planning and reinforcement learning. For example, candidate selection is mostly performed step-wise, but there are both planning, model-free RL and model-based RL papers that use a frontier-based candidate set. For the back-up, MCTS uses an on-policy, sample action, sample transition approach, which is for example shared by SARSA. While these algorithms differ on other dimensions, for example the way they represent their solution, they are similar in their back-up method. Monte Carlo targets for the return estimation appear in all three classes, as do 1-step bootstrapping methods.

There seems to be consensus on few dimensions. For the computational effort dimensions, we do see that nearly all papers in the table except for Dynamic Programming focus on the reachable state set, by sampling forward from some start state distribution. This is indeed our best bet if we do not want to suffer from the curse of dimensionality (see Sec. 3.2.2).

One may wonder why policy gradient methods still use the value back-up dimension. Policy gradients are actually a form of a loss, which specify how the policy should change based on a new value estimate. But the value estimate should still be obtained, and any of the methods from Sections 3.2.4 and 3.2.5 still apply. For example, policy gradient methods can be combined with Monte Carlo estimates (Williams, 1992), but also with bootstrapping (Mnih et al., 2016).

Note that some approaches, such as PILCO (Deisenroth and Rasmussen, 2011) and policy gradients (Williams, 1992), completely rely on a stochastic policy to explore, without any additional exploration pressure. This is technically a form of optimistic initialization, since the start policy should broadly cover state space. There is no additional

exploration pressure, and for these methods it is crucial that the initial policy hits a non-zero reward region, since otherwise there will be no learning signal at all. Therefore, this approach seems less applicable to large state spaces.

As discussed in Sec. 3.2.7, the replace and average update types are special cases of the squared loss. Since the squared loss is never explicitly specified in these tabular updates, we have entered 'squared' between brackets in those cases. For Go-Explore (Ecoffet et al., 2019) we have only considered their initial exploration phase in the table, and omitted the second imitation learning phase in which they solidify their own policy into a neural network. Some smaller comments on the table are part of the table caption.

3.4 RELATED WORK

There is surprisingly little work on a systematic categorization of either planning or reinforcement learning algorithms. The two main examples are *trial-based heuristic tree search* (THTS) (Keller, 2015; Keller and Helmert, 2013), and the textbook classification of back-up width and depth by Sutton and Barto, 2018. We will discuss both.

THTS is closest to our work, specifying a framework to systematically categorize planning methods. It contains six dimensions, which we will each compare to our framework:

Table 3.3: (next two pages): Systematic overview of various learning, planning and model-based RL methods, broken up according to FRAP. See Table 3.1 for an overview of the components, as discussed throughout Section 3.2. Colour coding: blue = planning, red = model-free RL, green = model-based RL. Abbrevations of function approximation types: NN = neural network, GP = Gaussian Process, k-NN = k-nearest neighbour. Notes: †For Go-Explore (Ecoffet et al., 2019) we only describe their primary exploration approach. In a second stage, they solidify their policy with imitation learning. ○: Real-time DP leaves the sample depth for the back-up unspecified. In the table we show the vanilla choice for DP itself, a sample depth of 0. $ Péré et al. (2018) actually stores $s_0, s_g \rightarrow \theta$, i.e., a mapping from start and goal state to policy parameters, which themselves define another parametric policy.

Paper	Environment	Learned model	Comp. effort	Trial selection					
				Candidate Set	Exploration	Sub-category	Phases	Reverse Trials	Description
Dynamic Programming (Bellman, 1966)	Reversible analytic		All	State set	State	Ordered	1		Sweep
Depth-first exh. search (Russell and Norvig, 2016)	Reversible analytic		Reach.	Step	State	Ordered	1		Sweep
Heuristic search (e.g., A^* (Hart, Nilsson, and Raphael, 1968))	Reversible analytic		Reach.	Frontier	Value	Prior	1		Greedy on heuristic
MCTS (Browne et al., 2012)	Reversible sample		Reach.	Step	Value	Uncertainty	2		Upper confidence bound
Real-time DP (Barto, Bradtke, and Singh, 1995)	Reversible analytic		Reach.	Step	State	Ordered	1		Random starts
Q-learning (Watkins and Dayan, 1992)	Irreversible sample		Reach.	Step	State	Ordered	1		Random starts
SARSA + eligibility trace (Sutton and Barto, 2018)	Irreversible sample		Reach.	Step	Value	Mean values	1		e.g., Boltzmann
REINFORCE (Williams, 1992)	Irreversible sample		Reach.	Step	Random	-	1		Stochastic policy
DQN (Mnih et al., 2015)	Irreversible sample		Reach.	Step	Value	Random	1		ϵ-greedy
PPO (Schulman et al., 2017)	Irreversible sample		Reach.	Step	Value	Mean values	1		Stochastic policy with entropy regularization
DDPG (Lillicrap et al., 2015)	Irreversible sample		Reach.	Step	Random	-	1		Noise process (Ornstein-Uhlenbeck)
Go-Explore† (Ecoffet et al., 2019)	Irreversible sample		Reach.	Frontier	State+ val+rand	Novelty+ prior+random	1		Frontier prior.: visit freq. + heuristics. On frontier: random perturbation.
AlphaStar (Vinyals et al., 2019)	Irreversible sample		Reach.	Step	State+ Value	Prior+mean values	1		Imitation learning + shaping rewards + entropy regularization
Dyna-Q (Sutton, 1990)	Irreversible sample	✓	Reach.	Step	State+ Value	Knowledge+ mean values	1		Novelty bonus + Boltzmann
Prioritized sweeping (Moore and Atkeson, 1993)	Irreversible sample	✓	Reach.	Step	State	Novelty	1	✓	Visitation frequency + Reverse trials
PILCO (Deisenroth and Rasmussen, 2011)	Irreversible sample	✓	Reach.	Step	Random	-	2		Stochastic policy on initialization
AlphaGo (Silver et al., 2017c)	Reversible Sample		Reach.	Step	Value + random	Uncertainty	2		Upper confidence bound + noise
Knowledge, e.g., surprise (Achiam and Sastry, 2017)	Irreversible sample	✓	Reach.	Step	State	Knowledge	1		Intrinsic reward for surprise
Competence IM, e.g., (Péré et al., 2018)	Irreversible sample	✓	Reach.	Frontier	State	Competence	1		Sampling in learned goal space

Paper	Cumulative return		Back-up			Representation		Update	
	Sample depth	Bootstrap type	Back-up policy	Action expectation	Dynamics Expectation	Function type	Function class	Loss	Update type
Dynamic Programming (Bellman, 1966)	1	Learned	Off-policy	Max	Exp.	Value	Global table	(Squared)	Replace
Depth-first exh. search (Russell and Norvig, 2016)	∞	None	Off-policy	Max	Exp	Value	Global table	(Squared)	Replace
Heuristic search (e.g., $A^\star$ (Hart, Nilsson, and Raphael, 1968))	1	Heuristic	Off-policy	Max	Determ.	Value	Global table	(Squared)	Replace
MCTS (Browne et al., 2012)	∞	None	On-policy	Sample	Sample	Value	Local table	(Squared)	Average
Real-time DP (Barto, Bradtke, and Singh, 1995)	1°	Learned	Off-policy	Max	Exp.	Value	Global table	(Squared)	Replace
Q-learning (Watkins and Dayan, 1992)	1	Learned	Off-policy	Max	Sample	Value	Global table	Squared	Gradient
SARSA + eligibility trace (Sutton and Barto, 2018)	$1-n$ (eligibility)	Learned	On-policy	Sample	Sample	Value	Global table	Squared	Gradient
REINFORCE (Williams, 1992)	∞	None	On-policy	Sample	Sample	Policy	Func.approx. (NN)	Policy gradient	Gradient
DQN (Mnih et al., 2015)	1	Learned	Off-policy	Max	Sample	Value	Func.approx. (NN)	Squared	Gradient
PPO (Schulman et al., 2017)	$1-n$ (eligibility)	Learned	On-policy	Sample	Sample	Policy	Func.approx. (NN)	Policy gradient	Gradient (trust.reg.)
DDPG (Lillicrap et al., 2015)	1	Learned	Off-policy	Max	Sample	Policy+ value	Func.approx. (NN)	Determ. policy grad. + squared	Gradient
Go-Explore[†] (Ecoffet et al., 2019)	1	Heuristic	On-policy	Sample	Sample	Policy	Global table	(Squared)	Replace
AlphaStar (Vinyals et al., 2019)	1-n (importance weighted)	Learned	On-policy	Sample	Sample	Policy+ value	Func.approx. (NN)	Policy gradient + squared	Gradient
Dyna (Sutton, 1990)	1	Learned	On-policy	Sample	Sample	Value	Global table	Squared	Gradient
Prioritized sweeping (Moore and Atkeson, 1993)	1	Learned	Off-policy	Max	Exp.	Value	Global table	Squared	Gradient
PILCO (Deisenroth and Rasmussen, 2011)	∞	None	On-policy	Sample	Sample	Policy	Func.approx. (GP)	Value gradient	Gradient
AlphaGo (Silver et al., 2017c)	MCTS: 1-n Value: ∞	Learned	On-policy	Sample	Sample	Policy+ value	Func.approx. (NN)+ local table	Cross-entropy+ Squared	Average+ Gradient
Knowledge, e.g., surprise (Achiam and Sastry, 2017)	∞	None	On-policy	Sample	Sample	Policy	Func.approx. (NN)	Policy gradient	Gradient
Competence IM, e.g., (Péré et al., 2018)	∞	None	On-policy	Sample	Sample	Generalized policy[§]	Func.approx. (k-NN)	k-NN loss	Gradient-free

- *Initialization*: In THTS, 'initialization' refers to the value a new node in the tree gets assigned when it is generated. The framework describes one possible approach, which is initializing the value with a heuristic. In our framework, this idea is captured as one of the options in the 'bootstrap' consideration of the cumulative return estimation dimension (Sec. 3.2.4.2), where we also discuss other methods.

- *Outcome selection*: In THTS, 'outcome selection' refers to the way we generate the next state or state distribution depending on the action. It describes one option: 'Monte Carlo selection', which samples each next state according to its probability under $\mathcal{T}$. In our framework, this is equal to the sample-based dynamics back-up, discussed in Sec. 3.2.5.3). Note that in our framework we treat this consideration as a back-up choice. If we only sample one action, then we can only back-up a single action, while if we consider the probabilities of all next states, then we can also make an expected back-up. Forward and backward over the dynamics model are thereby directly linked.

- *Back-up*: In THTS the 'back-up' dimension covers possible choices like 'Monte Carlo back-up' (characterized by 'averaging' updates in our framework), 'Temporal Difference back-up' (characterized by a bootstrap depth of 1 in our framework), 'Selective back-ups' (characterized by a 'off-policy' back-up in our framework), etc. To our view, the back-up dimension of THTS actually groups together multiple considerations of the cumulative reward, back-up and update dimensions of our framework. FRAP does properly disentangle these aspects.

- *Trial length*: This dimension describes in THTS by how many trials we expand the search graph. This is clearly related to the 'sample depth' dimension of cumulative reward estimation (Sec. 3.2.4.1) in our framework. However, there is an important additional difference. THTS only counts the graph expansions, which for example for MCTS gives a sample depth of 1 per iteration (ignoring the roll-out). We disagree, as the roll-out is actually a sequence of new trials. We also make back-ups along the roll-out path, but we simply do not use these intermediate estimates to update our representation, which is a separate dimension in our framework.

- *Action selection*: This dimension in THTS contains the categories 'Greedy', 'Uniform', 'ϵ-greedy', 'Boltzmann' and 'Upper Confi-

dence Bound'. In our framework, this equals part of the exploration dimension (Sec. 3.2.3.2). However, THTS does not further substructure this dimensions like we do, and thereby fails to incorporate a variety of other methods like intrinsic motivation, frontier-based candidate sets, one- versus two-phase exploration, and reverse trials.

- *Recommendation function*: The final dimension in THTS is the recommendation function, which takes in a search graph and returns a probability distribution over the actions at the root. This category is specific to the online search setting, when we are only interested in the policy at the root. Instead, FRAP contains an entire dimension for solution representation. The above recommendation is a form of a local policy table in our framework, from which we can read the recommendation decision. But FRAP also includes global representations and various kinds of function approximation methods.

THTS was an important inspiration for the current framework, by proposing that there is a common underlying algorithmic space beneath all MDP search algorithms. However, FRAP extends THTS in many ways, by including the entire spectrum of learning methods (and all its associated RL literature), and by adding and splitting several dimensions to overcome the overlap and confusion of some dimensions of THTS.

Sutton and Barto (2018) also discuss a categorization of planning and learning based on the width and depth of the back-ups. Together these lead to four extremes: exhaustive search (full breadth and depth), Dynamic Programming (full breadth, single depth), Monte Carlo estimation (single breadth, full depth), and temporal difference learning (single breadth, single depth). The depth is clearly represented by the sample depth of the cumulative reward estimation in our framework. The breadth is in FRAP split up in the expectation over the actions and dynamics in the back-up. Note that FRAP considers breadth a back-up dimension, and therefore considers exhaustive search as a long ordered series of 1-step back-ups. Instead, Sutton and Barto (2018) consider exhaustive search as a single, large, broad and deep back-up. Both views can exist next to one another. Our view better fits a systematic framework that disentangles the elementary operations in search and RL, but the view of Sutton and Barto (2018) is conceptually insightful as well, when we think of an entire planning iteration as creating one new value target.

3.5 DISCUSSION

This chapter introduced the framework for reinforcement learning and planning (FRAP), as a systematic approach to categorize and compare planning and reinforcement learning approaches. We will now put our work in a broader perspective, and identify possible implications for future work.

First of all, we did not include *stopping criteria* in our framework. Nearly all algorithms empirically stop based on a fixed hyperparameter, or based on manual intervention by a human logging the performance. While some algorithms do have convergence guarantees, like DP (Bellman, 1966), MCTS (Browne et al., 2012), A* (Hart, Nilsson, and Raphael, 1968) and many RL algorithms with GLIE (greedy in the limit with infinite exploration) assumptions, it is typically infeasible to assess convergence during execution. The only algorithms that do assess convergence need to either make sweeps through the entire state space (like dynamic programming and exhaustive search), which is the only way to guarantee that we have at a certain moment visited all states frequently enough, or require an admissible heuristic, which ensures that we can stop expanding before visiting all states (Hart, Nilsson, and Raphael, 1968).

TRACTABILITY OF MDP OPTIMIZATION The framework also allows us to zoom out and identify the fundamental ways in which a MDP search can be made tractable. The MDP problem essentially specifies an infinitely deep MAX-EXP tree which we can never fully enumerate.[4] On the most fundamental level, without any consideration of two-phase exploration and a real environment versus planning model, there are only four ways in which we can somehow reduce the size of the true underlying MDP tree:

1. *Reachable states*: focus on reachable states instead of all states (Sec. 3.2.2).
2. *Exploration-exploitation*: gradually focus from reachable to relevant states, i.e., reduce the breadth of the problem through exploration-exploitation balancing (Sec. 3.2.3).
3. *Generalization*: share relevance information of one state to other appearances of (approximately) the same state (Sec. 3.2.6).

4 Note that we focus on infinite-horizon returns, see Chapter 2

4. *Priors*: We do not really consider this a fundamental solution approach, as it requires task specific information. It is however a way to solve an otherwise intractable MDP.

A fifth way to make the problem tractable, which was not discussed in our framework, involves compressing the MDP itself, for example through temporal abstraction (better known as hierarchical RL (Barto and Mahadevan, 2003)) This may define a temporally abstract MDP, in which it is easier to solve for the solution. However, this topic falls outside the scope of this framework.

DIFFERENCES BETWEEN RESEARCH FIELDS One question that arises from FRAP is: what are the true differences between planning and reinforcement learning? The defining difference was already discussed in Section 2.2. Learning algorithms assume an irreversible environment ('unknown model'), while planning algorithm assume a reversible environment ('known model'). Therefore, planning algorithms can repeatedly plan forward from the same state, which RL algorithms cannot. On a conceptual level, the difference is mostly about *the order* in which we do the updates. Once again, 100 traces of MCTS or 100 traces of Q-learning conceptually do the same: they walk forward, acquire information, make back-ups, and update a (local) representation, all to better inform next traces/episodes.

All other differences except for the visitation order seem to be based on convention rather than necessity. We will provide some examples of common conventions in both fields. For example, planning algorithms tend to use (local) tabular representations, in the form of a (discrete) search tree. In contrast, RL algorithms tend to use global representations of the solution, and have put much more emphasis on function approximation. The planning community has put more focus on the use of bootstrapping from heuristics, while the RL community has focused on bootstrapping from learned value functions. Uncertainty-based exploration has been successful in planning approaches like MCTS, but has also appeared in RL research (Kaelbling, 1993). Frontiers originate in research on planning, but competence-based intrinsic motivation now applies similar principles in RL. Sample-based back-ups mostly originate in RL research, where we interact with an irreversible environment and have to rely on sample action, sample dynamics back-ups. However, exactly the same back-up has also become popular in the planning approach of MCTS. In short, both fields have emphasized their own elements of the overall problem, but have at the same time

invented similar solutions and approaches, which blurs the algorithmic line between both fields.

FUTURE WORK The framework may also help to identify potential directions for future research. We will provide some potential directions here. First of all, the strongest boundary in Table 3.3 is the type of representation used. Planning methods use local representations, while reinforcement learning methods use global representations. Clearly, these two types of representations can be combined, where we temporarily store local solutions, which we after a while use to update a global solution. These approaches of course fall in the model-based RL spectrum, which has been known for long (Sutton, 1990).

However, recent work, like AlphaGo Zero (Silver et al., 2017c) and Guided Policy Search (Levine and Koltun, 2013), has shown that we have far from completely understood the potential of integrated planning and learning approaches yet. We will further investigate this topic in Chapter 7 as well, but in general believe that the combination of local, atomic and global, generalizing representations still has much potential to offer.

As a second example, we hypothesize that 'frontier' approaches deserve extra attention the RL community. While frontiers originate in planning, they have no use in model-free RL. However, model-based RL can definitely profit from frontiers for exploration, for example in the direction of competence-based intrinsic motivation. Go-Explore (Ecoffet et al., 2019), which uses a frontier-like approach in an RL agent, is an interesting recent example in this direction as well.

Another example of a future research direction could be reverse trials and prioritized sweeping. This time the direction of influence is reversed, since this topic originates in RL and has received little attention in planning. Reverse trials may help spread information over the state-space much faster, and can be seen as a form of planning in the backwards direction. Potentially, we could even apply successful planning algorithms, like MCTS, in the backwards direction. Given the success of prioritized sweeping in tabular models, and some interesting recent applications in high-dimensional problems (Agostinelli et al., 2019; Corneil, Gerstner, and Brea, 2018; Edwards, Downs, and Davidson, 2018), this should be a promising direction for future research.

Another potential idea to extract from Table 3.3 is the combination of state-based and value-based exploration methods. Most algorithms chose one of both, e.g., they either explore based on value function uncertainty, or they explore based on state characteristics like intrinsic

motivation. Few papers have combined these approaches, while there is no practical limitation to do this, as for example shown by Ecoffet et al. (2019).

We also see that both fields have been dominated by gradient-based updates (since we showed that tabular planning updates can actually be cast as a form of gradient-based updating). While gradient-based optimization has been the driving force behind much of the progress in machine learning in the last decade, recent papers, for example by Jaderberg et al. (2019), show that the combination of gradient-based and gradient-free updating can give strong results as well.

In a similar direction, we also believe that 'episodic memory' approaches (Pritzel et al., 2017), where we do not even update but simply store raw traces, could get more attention. These may for example help in the context of exploration, to ensure that we can get back to a particular state. This is an example of a different form of tabular representation with a non-gradient based update (or actually an absent update, since we store the raw trace).

We also see that, especially in the reinforcement learning direction, there is quite some variation in the types of loss functions used, although most papers use the ones that we explicitly mentioned in this paper. However, for example, Hamrick et al. (2020) recently proposed a new type of loss to train a discrete policy network against value estimates, based on a cross-entropy loss with the softmax of the value estimates. This is just an example, but there could be more types of loss function to better integrate planning results into global approximations.

Altogether, these are some examples of possible interpretations of Table 3.3, while there could be more depending on your own background. We will experimentally show another example of a possible connection between both fields in Chapter 8. Nevertheless, besides these directions of future work, we believe the main contribution of the paper is the overview and structure to both fields it may provide.

3.6 CONCLUSION

This concludes the description of our framework for reinforcement learning and planning (FRAP). We briefly summarize the main ideas:

- We can disentangle planning algorithms, like $A^\star$, and RL algorithms, like Q-learning, into one underlying framework. Any algorithm that solves a MDP optimization (implicitly) makes decisions on: i) the considered state set, ii) trial selection and exploration,

iii) cumulative reward estimation, iv) value back-up, v) solution representation and vi) update of the solution. These dimensions, with their relevant considerations, are summarized in Table 3.1.

- A key conclusion of the framework is that the lines between planning and learning are actually blurry, and frequently based on convention rather than necessity. Both fields share the same underlying algorithmic space.
- MDP optimization can in principle be approached as a black-box optimization problem. However, our framework illustrates the various ways in MDP specific characteristics can be systematically incorporated in the solution approach.
- Altogether, the framework may serve several purposes: i) provide a common language for researchers in both planning and RL to categorize their solution approach, ii) inspire future research, for example through novel combinations of planning and learning, and iii) serve an educational purpose, for students, and for researchers from either planning or RL who consider working at the intersection of both fields.

4

MODEL-BASED REINFORCEMENT LEARNING: A SURVEY [1]

ABSTRACT

Sequential decision making, commonly formalized as Markov Decision Process (MDP) optimization, is a key challenge in artificial intelligence. Two key approaches to this problem are reinforcement learning (RL) and planning. This chapter presents a survey of the integration of both fields, better known as model-based reinforcement learning. Model-based RL has two main steps. First, we systematically cover approaches to dynamics model learning, including challenges like dealing with stochasticity, uncertainty, partial observability, and temporal abstraction. Second, we present a systematic categorization of planning-learning integration, including aspects like: where to start planning, what budgets to allocate to planning and real data collection, how to plan, and how to integrate planning in the learning and acting loop. After these two section, we also discuss implicit model-based RL as an end-to-end alternative for model learning and planning, and we cover the potential benefits of model-based RL, like enhanced data efficiency, targeted exploration, and improved stability. Throughout the survey, we also draw connections to several related RL fields, like hierarchical RL and transfer. Altogether, the survey presents a broad conceptual overview of planning-learning combinations for MDP optimization.

4.1 INTRODUCTION

Model-based reinforcement learning combines planning and learning in a single algorithm. We defined model-based RL as: 'any MDP approach that uses i) a model (known or learned) and ii) a global solution, like a learned value or policy function'. Model-based RL has shown great success (Deisenroth and Rasmussen, 2011; Levine and Koltun, 2013; Silver et al., 2017c), but, as mentioned before, literature lacks a systematic review of the field (although Hamrick et al. (2020) does provide a short review, see Sec. 4.7 for a detailed discussion of related work). This chapter aims to fill this gap, by presenting a broad overview of the possible combinations of planning and learning.

1 Chapter based on: Moerland TM, Broekens J, Jonker CM. Model-based Reinforcement Learning: A Survey. *In submission*.

The survey consist of four key sections. We first cover approaches to *dynamics model learning*, including important challenges like stochasticity, uncertainty, partial observability, non-stationarity, state abstraction, and temporal abstraction (Sec. 4.3). Then, we cover the integration of planning and learning, i.e., the ways we may use a (learned) dynamics model to solve for a (learned) policy (Sec. 4.4). Afterwards, Section 4.5 covers the implicit approach to model-based RL, as opposed to the explicit approaches of Sections 4.3 and 4.4. Finally, Sec. 4.6 covers the potential benefits of model-based reinforcement learning, such as data efficiency, targeted exploration, stability, transfer, safety and explainability.

Model-based RL is a fundamental approach to sequential decision making, and many other sub-fields in RL have a close connection to model-based RL. For example, *hierarchical reinforcement learning* (Barto and Mahadevan, 2003) can be approached in a model-free and model-based way. In the latter case, the higher-level action space defines a model with temporal abstraction. Model-based RL is also an important approach to *transfer learning* (Taylor and Stone, 2009) (through model transfer between tasks) and *targeted exploration* (Thrun, 1992). When applicable, the survey also presents short overviews of such related RL research directions.

The remainder of this chapter is organized as follows. We first discuss the main algorithmic categories within model-based RL (Sec. 4.2). Then, we present the main content of the survey, on model learning (Sec. 4.3), planning-learning integration (Sec. 4.4), implicit model-based RL (Sec. 4.5, and the benefits of model-based RL (Sec. 4.6). At the end of the survey, we also present Related Work (Sec. 4.7), Discussion (Sec. 4.8), and Summary (Sec. 4.9) sections.

4.2 CATEGORIES OF MODEL-BASED REINFORCEMENT LEARNING

We refer the reader to Chapter 2 for a formal introduction to the MDP problem, relevant notation, and an introduction to planning, reinforcement learning and model-based RL. We defined model-based RL as: 'any MDP approach that uses 1) a model (known or learned), i.e., reversible access to the MDP dynamics, and 2) a global solution, like a learned value or policy function'. The first model-based RL algorithm was developed by Sutton (1990), although similar ideas were proposed around the same time in the search community, in the form of Learning Real-Time A* (Korf, 1990). Note that the underlying principles of

Table 4.1: Categories of planning-learning integration. The top two rows show the two separate research fields of model-free RL and planning. The table shows three forms of explicit integration of planning and learning, depending on whether the model is learned and/or whether a global value or policy is learned.

	Model	**Learned model**	**Global value/policy**
Model-free RL			✓
Planning	✓		
Model-based RL with a learned model	✓	✓	✓
Model-based RL with a known model	✓		✓
Planning over a learned model	✓	✓	

integrated planning and learning at least date back to the Checkers programme by Samuel (1967).

It is important to note that, in the context of planning and learning integration, learning is actually an overloaded term, since it may happen at two locations: 1) to approximate a dynamics model, and 2) to approximate a value or policy function. This leads to the following three categories of explicit planning-learning integration (summarized in Table 4.1):

- *Model-based RL with a learned model*, where we both learn a model and learn a global value or policy. An example is Dyna (Sutton, 1991).
- *Model-based RL with a known model*, where we have a known model and use planning to learn a global value and/or policy. An example is AlphaGo Zero (Silver et al., 2017c).
- *Planning over a learned model*, where we learn a model and (locally) plan over it, without learning a global value or policy function. An example is Embed2Control (Watter et al., 2015).

Note that the last category is not considered model-based RL, since it does not learn a global solution to the problem. However, it is a form of planning-learning integration (and some researchers may actually consider it model-based RL), and we therefore will include this topic in the survey. Also, note that the line between replay databases (Lin, 1992) and model-based RL with a learned tabular model is very blurry

(Hasselt, Hessel, and Aslanides, 2019; Vanseijen and Sutton, 2015), which of course also applies to the relation with episodic memory (Pritzel et al., 2017).

It is important to distinguish the above categories, because they need to cope with different challenges. For example, approaches with a learned dynamics model typically need to account for uncertainty, while approaches with a known/given dynamics model can ignore this issue, an put stronger emphasis on asymptotic performance. We will extensively encounter these categories in Sec. 4.4 on planning-learning integration. We will now start our survey of the field, starting with the common first step of model-based RL: dynamics model learning.

4.3 DYNAMICS MODEL LEARNING

The first step of model-based RL usually involves learning the dynamics model from observed data. In the control literature, dynamics model learning is better known as *system identification* (Åström and Eykhoff, 1971; Ljung, 2001). We will first cover the general considerations of learning a one-step model (Sec. 4.3.1). Afterwards, we extensively cover the various challenges of model learning, and their possible solutions. These challenges are stochasticity (Sec. 4.3.2), uncertainty due to limited data (Sec. 4.3.3), partial observability (Sec. 4.3.4), non-stationarity (Sec. 4.3.5), multi-step prediction (4.3.6), state abstraction (Sec. 4.3.7) and temporal abstraction (Sec. 4.3.8). The reader may wish to skip some of these section if the particular challenge is not relevant to your research problem or task of interest.

4.3.1 *Basic considerations*

Model learning is essentially a supervised learning problem (Jordan and Rumelhart, 1992), and many topics from the supervised learning community apply here. We will first focus on a simple one-step model, and discuss the three main considerations: what type of model do we learn, what type of estimation method do we use, and in what region should our model be valid?

TYPE OF MODEL The first question is: what do we actually consider to be a model? We will here focus on dynamics models. A model of the reward function can usually be easily added by predicting an additional

scalar. Given a batch of one-step transition data $\{s_t, a_t, r_t, s_{t+1}\}$, there are three main types of dynamics function we might be interested in:

- *Forward model*: $(s_t, a_t) \rightarrow s_{t+1}$. This predicts the next state given a current state and chosen action. It is by far the most common type of model, and can be used for lookahead planning.
- *Backward/reverse model*: $s_{t+1} \rightarrow (s_t, a_t)$. This model predicts which states are the possible precursors of a particular state. Thereby, we can plan in the backwards direction, which is for example used in prioritized sweeping (Moore and Atkeson, 1993).
- *Inverse model*: $(s_t, s_{t+1}) \rightarrow a_t$. An inverse model predicts which action is needed to get from one state to another. It is for example used in RRT planning (LaValle, 1998). As we will later see, this function can also be useful as part of representation learning (Sec. 4.3.7).

Model-based RL has mostly focused on forward models, and these will also be the main focus of our discussion.

ESTIMATION METHOD We next need to determine what type of approximation method (supervised learning method) we will use. We discriminate between parametric and non-parametric methods, and between exact and approximate methods.

- *Parametric*: Parametric methods are the most popular approach for model approximation. Compared to non-parametric methods, a benefit of parametric methods is that their number of parameters is independent of the size of the observed dataset. There are two main subgroups:
  - *Exact*: A cardinal distinction in learning is between exact/tabular and approximate methods. For a discrete MDP (or a discretized version of a continuous MDP), a tabular method maintains a separate entry for every possible transition. For example, in a stochastic MDP (in which we need to learn a probability distribution, see next section) a *tabular maximum likelihood model* (Sutton, 1991) estimates the probability of each possible transition as

$$T(s'|s,a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}, \tag{4.1}$$

where T denotes the approximation of the true dynamics $\mathcal{T}$, and $n(s, a, s')$ denotes the number of times we observed s' after taking action a in state s. This approach effectively normalizes the observed transition counts. Tabular models were popular in initial model-based RL (Sutton, 1990). However, they do not scale to high-dimensional problems, as the size of the required table scales exponentially in the dimensionality of $\mathcal{S}$.

  - *Approximate*: We may also *approximate* the function, which will scale down the memory requirements, introduce generalization of information between similar states. Function approximation is therefore the preferred approach in higher-dimensional problems. We may in principle use any parametric approximation method to learn the model. Examples include linear regression (Parr et al., 2008; Sutton et al., 2008), Dynamic Bayesian networks (DBN) (Hester and Stone, 2012b), nearest neighbours (Jong and Stone, 2007), random forests (Hester and Stone, 2013), support vector regression (Müller et al., 1997) and neural networks (Narendra and Parthasarathy, 1990; Oh et al., 2015; Wahlström, Schön, and Deisenroth, 2015; Werbos, 1989). Especially (deep) neural networks have become very popular in the last decade, for function approximation in general (Goodfellow, Bengio, and Courville, 2016), and therefore also for dynamics approximation. Compared to the other methods, neural networks especially scale (computationally) well to high-dimensional inputs, while being able to flexibly approximate non-linear functions. Nevertheless, other approximation methods still have their use as well.

- *Non-parametric*: The other main supervised learning approach is non-parametric approximation. The main property of non-parametric methods is that they directly store and use the data to represent the model.

  - *Exact*: Replay buffers (Lin, 1992) can actually be regarded as non-parametric versions of tabular methods. While a table has parameters (all table entries) and thereby a fixed size determined by the MDP dynamics, a replay buffer can theoretically continue to store all data, although we of course cap the size in practice.

  - *Approximate*: We may also apply non-parametric methods when we want to be able to generalize information to similar states. For example, Gaussian processes (Deisenroth and Rasmussen, 2011; Wang, Hertzmann, and Fleet, 2006) have been a popular non-parametric approach. Gaussian processes can also provide good uncertainty estimates, which we will further discuss in Sec. 4.3.3.

  The computational complexity of non-parametric methods depends on the size of the dataset, which makes them less applicable to high-dimensional problems, where we usually require more data.

Throughout this work, we sometimes refer to the term 'function approximation'. We then imply all *non-tabular* (non-exact) methods, i.e., all methods that generalize information between states.

REGION IN WHICH THE MODEL IS VALID The third important consideration is the region of state space in which we aim to make the model valid:

- *Global*: These models approximate the dynamics over the entire state space. This is the main approach of most model learning methods. It can be challenging to generalize well over the entire state space, but it is the main way to store all information from previous observations.

- *Local*: The other approach is to only locally approximate the dynamics, and each time discard the local model after planning over it. This approach is especially popular in the control community, where they frequently fit local linear approximations of the dynamics around some current state (Atkeson, Moore, and Schaal, 1997; Bagnell and Schneider, 2001; Levine and Abbeel, 2014). A local model restricts the input domain in which the model should be valid, and is also fitted to a restricted set of data. A benefit of local models is that we may use a more restricted function approximation class (like linear), and potentially have less instability compared to global approximation. On the downside, we continuously have to estimate new models, and do not continue to learn from all collected data (since it is infeasible to store all previous datapoints).

The distinction between global and local is equally relevant for representation of a value or policy function, as we will see in Sections 4.4 and 4.6.

This concludes our discussion of the three basic considerations of model learning. In practice, most model learning focuses on a particular combination of these: a forward model, with parametric function approximation, and global coverage. We will now discuss the more advanced challenges of model learning, in which this setting will also get most attention.

4.3.2 *Stochasticity*

In a stochastic MDP the transition function specifies a distribution over the possible next states, instead of returning a single next state (Figure 4.1, left). In those cases, we should also specify a model that can approximate entire distributions. Otherwise, when we for example train a deterministic neural network $f_\phi(s, a)$ on a mean-squared error loss (e.g., Oh et al. (2015)), then the network will actually learn to predict the conditional mean of the next state distribution (Moerland, Broekens, and Jonker, 2017b). This problem is illustrated in Figure 4.1, right.

We can either approximate the entire next state distribution (descriptive models), or approximate a model from which we can only draw samples (generative model). Descriptive models are mostly feasible in small state spaces. Examples include tabular models, Gaussian models (Deisenroth and Rasmussen, 2011) and Gaussian mixture models (Khansari-Zadeh and Billard, 2011), where the mixture contribution typically involved *expectation-maximization* (EM) style inference (Ghahramani and Roweis, 1999). However, these methods do not scale well to high-dimensional state spaces.

In high-dimensional problems, there has been much recent effort on generative models based on neural network approximation (deep generative models). One approach is to use variational inference (VI) to estimate dynamics models (Babaeizadeh et al., 2017; Buesing et al., 2018; Depeweg et al., 2016; Moerland, Broekens, and Jonker, 2017b). Competing approach include generative adversarial networks (GANs), autoregressive full-likelihood models, and flow-based density models, which were applied to sequence modeling by Yu et al. (2017), Kalchbrenner et al. (2017) and Ziegler and Rush (2019), respectively. Detailed discussion of these methods falls outside the scope of this survey, but

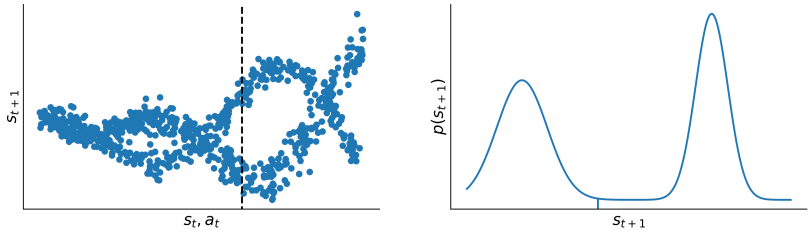


Figure 4.1: Illustration of stochastic transition dynamics. **Left**: 500 samples from an example transition function $\mathcal{T}(s'|s,a)$. The vertical dashed line indicates the cross-section distribution on the right. **Right**: distribution of s_{t+1} for a particular s, a. We observe a multimodal distribution. The conditional mean of this distribution, which would be predicted by mean squared error (MSE) training, is shown as a vertical line.

there is no clear consensus yet which deep generative modeling approach works best.

4.3.3 *Uncertainty*

A crucial challenge of model-based learning is dealing with uncertainty due to limited data. Uncertainty due to limited data (also known as *epistemic* uncertainty) clearly differs from the previously discussed stochasticity (also known as *aleatoric* uncertainty) (Der Kiureghian and Ditlevsen, 2009), in the sense that uncertainty can be reduced by observing more data, while stochasticity can never be reduced. We clearly want to be able to estimate the remaining uncertainty in our model estimate, to assess whether our plan is actually reliable. Uncertainty is even relevant in the absence of stochasticity, as illustrated in Figure 4.2.

We therefore want to estimate the uncertainty around our predictions. Then, when we plan over our model, we can detect when our predictions become less trustworthy. There are two principled approaches to uncertainty estimation in statistics: frequentist and Bayesian. A frequentist approach is for example the statistical bootstrap, applied to model estimation by Fröhlich, Theis, and Hasenauer (2014) and Chua et al. (2018). Bayesian RL methods were previously surveyed by Ghavamzadeh et al. (2015). Especially successful have been non-parametric Bayesian methods like Gaussian Processes (GPs), for example used for model estimation in PILCO (Deisenroth and Rasmussen, 2011). However, GPs

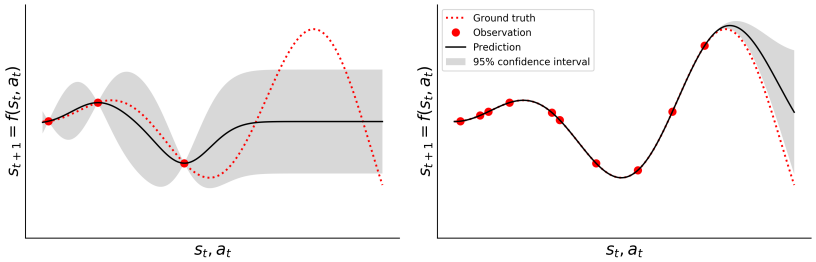


Figure 4.2: Illustration of uncertainty due to limited data. Red dotted line depicts an example ground truth transition function. **Left**: Gaussian Process fit after 3 observations. The predictions are clearly off in the right part of the figure, due to wrong extrapolation. The shaded area shows the 95% confidence interval, which does identify the remaining uncertainty, although not completely correct. **Right**: Gaussian Process fit after 10 observations. Predictions are much more certain now, mostly matching the true function. There is some remaining uncertainty on the far right of the curve.

scale (computationally) poorly to high-dimensional state spaces. Therefore, there has been much recent interest in Bayesian methods for neural network approximation of dynamics, for example based on variational dropout (Gal, McAllister, and Rasmussen, 2016) and variational inference (Depeweg et al., 2016). Note that uncertainty estimation is also an active research topic in the deep learning community itself, and advances in those fields will likely benefit model-based RL as well. While this section discussed uncertainty estimation, we will discuss how to deal with model uncertainty during planning in Sec. 4.4.

4.3.4 *Partial observability*

Partial observability occurs in an MDP when the current observation does not provide all information about the ground truth state of the MDP. Note the difference between partial observability and stochasticity. Stochasticity is fundamental noise in the transition of the ground truth state, and can not be mitigated. Instead, partial observability originates from a lack of information in the current observation, but can partially be mitigated by incorporating information from previous observations. For example, a first-person view agent can not see what is behind it

right now, but it can remember what it saw behind it a few observations ago, which mitigates the partial observability.

So how do we incorporate information from previous observations? There are four main approaches: i) windowing, ii) belief states, iii) recurrency and iv) external memory (Figure 4.3).

- *Windowing*: In the windowing approach we concatenate the n most recent observations and treat these together as the state (Lin and Mitchell, 1992). McCallum (1997) extensively studies how to adaptively adjust the window size. In some sense, this is the tabular solution to partial observability. Although effective in small problems, there are several important limitations. First of all, the size of the model grows linearly in n (the history length), which makes them less applicable in high-dimensional problems or with large n. More importantly, they do not generalize at all between similar histories, which makes it hard to apply them in high-dimensional problems as well (where we seldomly encounter exactly the same history twice).

- *Belief states*: Belief states explicitly partition the learned dynamics model in an *observation model* $p(o|s)$ and a *latent transition model* $\mathcal{T}(s'|s,a)$ (Chrisman, 1992). This structure reminds of the sequence modeling approach of *state-space models* (Bishop, 2006), like hidden Markov models (HMM). Estimation of model parameters is usually based on expectation-maximization (EM) schemes (Ghahramani and Hinton, 1996). There are also specific planning methods for belief state models, known as POMDP planners (Kurniawati, Hsu, and Lee, 2008; Silver and Veness, 2010; Spaan and Vlassis, 2004). However, belief state models usually require prior knowledge on the belief state structure, and have trouble scaling to high-dimensional problems (since the expectation step becomes intractable).

- *Recurrency*: The most popular solution to partial observability is probably the use of *recurrent* neural networks, first applied to dynamics learning in Lin (1993) and Parlos, Chong, and Atiya (1994). A variety of papers have studied RNNs in high-dimensional settings in recent years (Chiappa et al., 2017; Gemici et al., 2017; Ha and Schmidhuber, 2018). Since the transition parameters of the RNN are shared between all timesteps, the model size is independent of the history length, which is one the main benefits of RNNs.

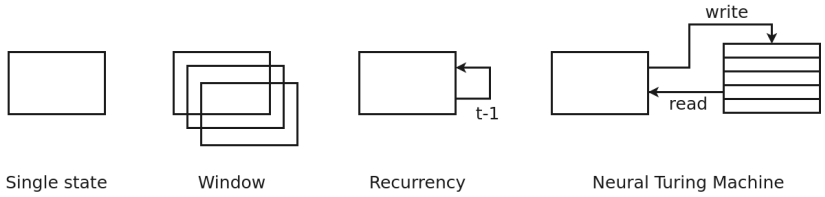


Figure 4.3: Example approaches to partial observability. The window approach concatenates the most recent n frames and treats this as a new state. The recurrent approach learns a recurrent mapping between timesteps to propagate information. The Neural Turing Machine uses an external memory to explicitly write away information and read it back when relevant, which is especially applicable to long-range dependencies.

They also neatly integrate with gradient-based training and high-dimensional state spaces. However, they do suffer from vanishing and exploding gradients to model long-range dependencies. This may be partly mitigated by long short-term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) or temporal skip connections (El Hihi and Bengio, 1996). Beck et al. (2020) recent proposed *aggregators*, which are more robust to long-range stochasticity in the observed sequences, as frequently present in RL tasks.

- *External memory*: The final approach to partial observability is the use of an external memory. Peshkin, Meuleau, and Kaelbling (1999) already gave the agent access to arbitrary bits in its state that could be flipped by the agent. Over time it learned to correctly flip these bits to memorize historical information. A more flexible extension of this idea are Neural Turing Machines (NTM) (Graves, Wayne, and Danihelka, 2014), which have read/write access to an external memory, and can be trained with gradient descent. Gemici et al. (2017) study NTMs in the context of model learning. External memory is especially useful for long-range dependencies, since we do not need to keep propagating information, but can simply recall it once it becomes relevant. The best way to store and recall information is however still an open area of research.

Partial observability is an inherent property of nearly all real-world tasks. When we ignore partial observability, our solution may completely fail. Therefore, many research papers that actually focus on some other question, still need to incorporate methodology to battle the

partial observability in the domain. Finally, note that the above partial observability methodology is equally applicable to a learned policy or value function.

4.3.5 *Non-stationarity*

Non-stationarity in a MDP occurs when the true transition and/or reward function change(s) over time. When the agent keeps trusting its previous model, without detecting the change, then its performance may deteriorate fast. Figure 4.4 illustrates the problem.

The main approach to non-stationarity are *partial models* (Doya et al., 2002). Partial models are an ensemble of stationary models, where the agents tries to detect regime switches and switches between models accordingly. Da Silva et al. (2006) detect a switch based on the prediction errors in transition and reward models. Nagabandi, Finn, and Levine (2018) makes a soft assignment based on a Dirichlet process. Jaulmes, Pineau, and Precup (2005) propose a simpler approach than partial models, by simply strongly decaying the contribution of older data (which is similar to a high learning rate). However, a high learning rate also introduces a lot of instability in training.

Transfer learning (Taylor and Stone, 2009) and meta-learning (Li et al., 2017) can actually be seen as special cases of non-stationarity, since we deal with optimization over a sequence of tasks. For example, Fu, Levine, and Abbeel (2016) aim to learn a generic neural network prior, which can be quickly adapted to new tasks. We further discuss transfer learning in Sec. 4.6.4.

4.3.6 *Multi-step Prediction*

After learning a model we intend to plan over it, which usually involves a multi-step look-ahead. The models we discussed so far made 1-step predictions of the next state. We can make multi-step predictions with such models by repeatedly feeding the prediction back into the learned model. However, since our learned model was never optimized to make long range predictions, accumulating errors may actually cause our multi-step predictions to diverge from the true dynamics. Several authors have identified this problem (Machado et al., 2018; Talvitie, 2014, 2017; Venkatraman, Hebert, and Bagnell, 2015).

There are two approaches to obtain better multi-step predictions: i) different loss functions and ii) separate dynamics functions for 1,2..n-

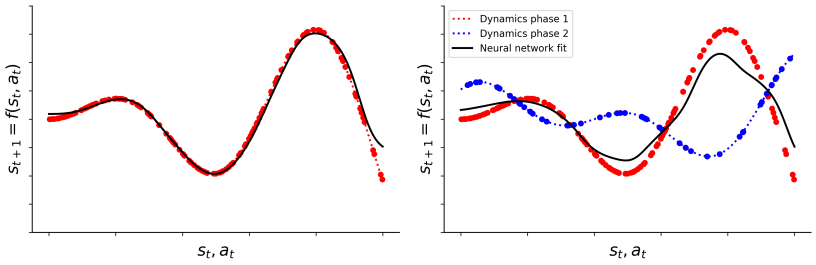


Figure 4.4: Illustration of non-stationarity. **Left**: First 150 data points sampled from initial dynamics. Black line shows the prediction of a neural network with 2 hidden layers of 50 units and tanh activations trained for 150 epochs. **Right**: Due to non-stationarity the dynamics changed to the blue curve, from which we sample an additional 50 points. The black curve shows the new neural network fit without detection of the dynamics change, i.e., treating all data as valid samples from the same transition distribution. We clearly see the network has trouble adapting to the new regime, as it still tries to fit to the old dynamics data points as well.

step predictions. In the first approach we simply include multi-step prediction losses in the overall training target (Abbeel and Ng, 2005; Chiappa et al., 2017; Hafner et al., 2019b; Ke et al., 2019). These models still make 1-step predictions, but during training they are unrolled for n steps and trained on a loss with the ground truth n-step observation. The second solution is to learn a specific dynamics model for every n-step prediction (Asadi et al., 2018). In that case, we learn for example a specific function $T^3(\hat{s}_{t+3}|s_t, a_t, a_{t+1}, a_{t+2})$, which makes a three step prediction conditioned on the current state and future action sequence. Some authors directly predict entire trajectories, which combines predictions of multiple depths (Mishra, Abbeel, and Mordatch, 2017). The second approach will likely have more parameters to train, but prevents the instability of feeding an intermediate prediction back into the model.

Some papers do not explicitly specify how many steps in the future to predict (Neitz et al., 2018), but for example automatically adjust this based on the certainty of the predicted state (Jayaraman et al., 2018). The topic of multi-step prediction also raises a question about performance measures. If our ultimate goal is multi-step planning, then one-step prediction errors are likely not a good measure of model performance.

4.3.7 *State abstraction*

Representation learning is a crucial topic in reinforcement learning and control (Lesort et al., 2018). Good representations are essential for good next state predictions, and equally important for good policy and value functions. Representation learning, also referred to as dimensionality reduction, is an important research field in machine learning itself, and many advances in state abstraction for model estimation build on results in the broader representation learning community.

Early application of representation learning in RL include (soft) state aggregation (Singh, Jaakkola, and Jordan, 1995) and principal component analysis (PCA) (Nouri and Littman, 2010). Mahadevan (2009) covers various approaches to learning basis functions in Markov Decision Processes. However, by far the most successful approach to representation learning in recent years have been deep neural networks, with a variety of example applications to model learning (Chiappa et al., 2017; Oh et al., 2015; Watter et al., 2015).

A (deep) neural network dynamics model is typically factorized in three parts: i) an *encoding* function $z_t = f_\phi^{\text{enc}}(s_t)$, which maps the observation to a latent representation z_t, ii) a *latent dynamics* function $z_{t+1} = f_\phi^{\text{trans}}(z_t, a_t)$, which transitions to the next latent state based on the chosen action, and iii) a *decoder* function $s_{t+1} = f_\phi^{\text{dec}}(z_{t+1})$, which maps the latent state back to the next state prediction. This structure, visualized in Figure 4.6 (item 4), reminds of an auto-encoder (with added latent dynamics), as frequently used for representation learning in the deep learning community.

There are three important additional themes for state representation learning in dynamics models: i) how do we ensure that we can plan at a latent level, ii) how may we better structure our models to emphasize objects and their physical interactions, and iii) how may we construct loss functions that retrieve more informative representations.

PLANNING AT A LATENT LEVEL We ideally want to be able to plan at a latent level. Since the representation space is usually smaller than the observation space, this may save much computational effort. However, we must ensure that the predicted next latent state lives in the same embedding space as the encoded current latent state. Otherwise, repeatedly feeding the latent prediction into the latent dynamics model will lead to predictions that diverge from the truth. One approach is to add an additional loss that enforces the next state prediction to

be close to the encoding of the true next state (Watter et al., 2015). An alternative are deep state-space models, like deep Kalman filters (Krishnan, Shalit, and Sontag, 2015) or deep variational Bayes filters (Karl et al., 2016). These require probabilistic inference of the latent space, but do automatically allow for latent level planning.

We may also put additional restrictions on the latent level dynamics that allow for specific planning routines. For example, (iterative) linear-quadratic regulator (LQR) (Todorov and Li, 2005) planning requires a linear dynamics function. Several authors (Fraccaro et al., 2017; Van Hoof et al., 2016; Watter et al., 2015) linearize their learned model on the latent level, and subsequently apply iLQR to solve for a policy (Van Hoof et al., 2016; Watter et al., 2015; Zhang et al., 2019). In this way, the learned representations may actually simplify planning, although it does require that the true dynamics can be linearly represented at latent level.

State abstraction is also related to *grey-box* system identification. In system identification (Åström and Eykhoff, 1971), the control term for model learning, we may discriminate 'black box' and 'grey box' approaches (Ljung, 2001). Black box methods, which do not assume any task-specific knowledge in their learning approach, are the main topic of Section 4.3. Grey box methods do partially embed task-specific knowledge in the model, and estimate remaining free parameters from data. The prior knowledge of grey box models is usually derived from the rules of physics. One may use the same idea to learn state abstractions. For example, in a robots task with visual observations, we may known the required (latent) transition model (i.e., f_ϕ^{trans} is known from physics), but not the encoding function from the visual observations (f_ϕ^{enc} is unknown). Wu et al. (2015) give an example of this approach, where the latent level dynamics are given by a known, differentiable physics engine, and we optimize for the encoding function from image observations.

OBJECTS A second popular approach to improve representations is by focusing on *objects* and their interactions. Infants are able to track objects at early infancy, and the ability to reason about object interaction is indeed considered a core aspect of human cognition (Spelke and Kinzler, 2007). In the context of RL, these ideas have been formulated as object-oriented MDPs (Diuk, Cohen, and Littman, 2008) and relational MDPs (Guestrin et al., 2003). Compared to models that predict raw pixels, such object-oriented models may better generalize to new, unseen

environments, since they disentangle the physics rules about objects and their interactions.

We face two important challenges to learn an object-oriented model: 1) how do we identify objects, and 2) how do we model interaction between objects at a latent level. Regarding the first questions, several methods have provided explicit object recognizers in advance (Fragkiadaki et al., 2015; Kansky et al., 2017), but other recent papers manage to learn them from the raw observations in a fully unsupervised way (Van Steenkiste et al., 2018; Watters et al., 2019; Xu et al., 2019). The interaction between objects is typically modeled like a *graph neural network*. In these networks, the nodes should capture object features (e.g., appearance, location, velocity) and the edge update functions predict the effect of an interaction between two objects (Van Steenkiste et al., 2018). There is a variety of recent successful examples in this direction, like Schema Networks (Kansky et al., 2017), Interaction Networks (Battaglia et al., 2016), Neural Physics Engine (Chang et al., 2016), Structured World Models (Kipf, Pol, and Welling, 2020) and COBRA (Watters et al., 2019). In short, object-oriented approaches tend to embed (graph) priors into the latent neural network structure that enforce the model to extract objects and their interactions. We refer the reader to Battaglia et al. (2018) for a broader discussion of relational world models.

BETTER LOSS FUNCTIONS Another way to achieve more informative representations is by constructing better loss functions. First of all, we may share the representation layers of the model with other prediction tasks, like predicting the reward function. The idea to share different prediction targets to speed-up representation learning is better known as an 'auxilliary loss' (Jaderberg et al., 2016).

We may also construct other losses for which we do not directly observe the raw target. For example, a popular approach is to predict the *relative* effect of actions: $s_{t+1} - s_t$ (Finn, Goodfellow, and Levine, 2016). Such background subtraction ensures that we focus on moving objects. An extension of this idea is *contingency awareness*, which describes the ability to discriminate between environment factors within and outside our control (Watson, 1966). We would also like to emphasize these controllable aspects in our representations. One way to achieve this is through an inverse dynamics loss, where we try to predict the action that achieves a certain transition: $(s, s') \rightarrow a$ (Pathak et al., 2017). This will focus on those parts of the state that the chosen action affects. Other approaches that emphasize controllable factors can be found in Choi et al. (2018), Sawada (2018), and Thomas et al. (2018).

There is another important research line that improves representations through *contrastive* losses. A contrastive loss is not based on a single data point, but on the similarity or dissimilarity with other observations. As an example, Sermanet et al. (2018) record the same action sequence from different viewpoints, and obtains a compact representation by enforcing similar states from different viewpoints to be close to eachother in embedding space. Ghosh, Gupta, and Levine (2018) add a loss based on the number of actions needed to travel between states, which enforces states that are dynamically close to be close in representation space as well. This is an interesting idea, since we use representation learning to actually make planning easier. Contrastive losses have also been constructed from the rules of physics in robotics tasks (Jonschkowski and Brock, 2015), have been applied to Atari models (Anand et al., 2019), and have combined with the above object-oriented approach (Kipf, Pol, and Welling, 2020).

Finally, there is an additional way to improve representations through *value equivalent models* (Grimm et al., 2020). These models are trained on their ability to predict a value or (optimal) action. We decide to cover this idea in Sec. 4.5 on implicit model-based RL, which covers methods that optimize elements of the model-based RL process for the ability to output an (optimal) action or value. In short, this section discussed the several ways in which the state representation learning of models may be improved, for example by embedding specific substructure in the networks (e.g., to extract objects and their interactions), or by constructing smarter loss functions.

4.3.8 *Temporal abstraction*

The MDP definition typically involves low-level, atomic actions executed at a high-frequency. This generates deep search trees with long-range credit assignment. However, many of these paths give the same end-state, and some end-states are more useful than others. The idea of temporal abstraction, better known as *hierarchical* reinforcement learning (Barto and Mahadevan, 2003; Hengst, 2017; Thrun and Schwartz, 1995), is to identify a high-level action space that extends over multiple timesteps (Figure 4.5). Temporal abstraction may reduce both the sample (Brunskill and Li, 2014) and computational complexity (Mann and Mannor, 2014) of solving the MDP.

There are a variety of frameworks to define abstract actions. One popular choice is the *options* framework (Sutton, Precup, and Singh,

1999). Options are a discrete set of high-level actions. Each option u has its own initiation set $I^u \in \mathcal{S}$ from which the option can be started, a sub-policy π^u for execution, and a state-dependent termination probability $\beta^u(s)$ for the option to end in a reached state. A popular competing approach are *goal-conditioned policy/value functions* (GCVF), also known as universal value function approximators (Schaul et al., 2015). These ideas originally date back to work on Feudal RL (Dayan and Hinton, 1993). GCVFs use a goal space $\mathcal{G}$ as the abstract action space. They learn a goal-conditioned value function $Q_g(s, a, g)$, which estimates the value of a in s if we attempt to reach g. We train such models on a *goal-parametrized reward function*, which for example rewards the agent for getter closer to g in Euclidean distance (Nachum et al., 2018). Afterwards, we can plan by chaining multiple subgoals.

Options and goal-conditioned value functions show conceptual differences. Most importantly, options have a separate sub-policy per option, while GCVFs attempt to generalize over goals/subpolicies. Moreover, options fix the initiation and termination set based on state information, while GCVFs can initiate and terminate everywhere. Note that GCVFs in some sense interpolate between one-step models (pick a really close goal) and model-free RL (directly impute the final goal in the GCVF), as for example shown by Pong et al. (2018).

DISCOVERY OF RELEVANT SUB-ROUTINES Whether we use options, GCVFs, or some other definition of abstract actions, the most important question is: how do we actually identify the *relevant* subroutines, i.e., relevant end-states for our options, or goal states for our GCVF. We summarize the most important approaches below:

- *Graph structure*: This approach identifies 'bottleneck' states as endpoints for the subroutines. A bottleneck is a state that connects two densely interconnected subgraphs in the MDP graph (Menache, Mannor, and Shimkin, 2002). Therefore, a bottleneck is a crucial state in order to reach another region of the MDP, and therefore a candidate subgoal. There are several ways to identify bottlenecks: McGovern and Barto (2001) identify bottlenecks from overlapping states in successful trials, Şimşek, Wolfe, and Barto (2005) run a graph partitioning algorithms on a reconstruction of the MDP graph, and Goel and Huber (2003) search for states with many predecessors, but whose successors do not have many predecessors. The bottleneck approach received much attention in smaller

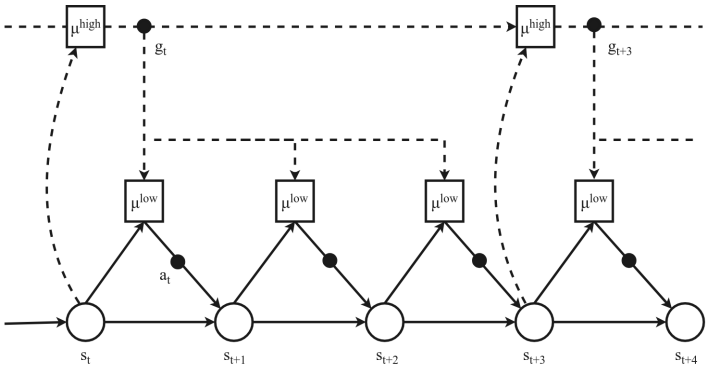


Figure 4.5: Conceptual illustration of a two-level hierarchy, partially based on Nachum et al. (2018). Standard low-level interaction is shown with solid lines, temporal abstraction is shown with dashed lines. The high-level controller picks a high-level action (goal) g_t according to π^{high}. After fixing g_t, the low level controller executes the relevant subpolicy, for example in the form of a goal-conditioned policy $\pi^{\text{low}}(s, g)$. The number of steps between high-level actions can be fixed or variable, depending on the framework. The illustration assumes full observability, in which case we only need to condition π^{high} on the current observation. We may also feed g back into the next high-level decision to enable temporal correlation between goals.

problems, but have received less attention in higher-dimensional problems.

- *State-space coverage*: Another idea is to spread the end-states of subroutines over the entire state-space, in order to reach good coverage. Most approaches first cluster the state space, and subsequently learn a dynamics model to move between the cluster centers (Lakshminarayanan et al., 2016; Machado, Bellemare, and Bowling, 2017; Mannor et al., 2004). Instead of the raw state space, we may also cluster in a compressed representation of it (Ghosh, Gupta, and Levine, 2018) (see previous section as well).

- *Compression (information-theoretic)*: We may also attempt to simply compress the space of possible end-points. This idea is close to the state space coverage ideas above. Achiam et al. (2018), Eysenbach et al. (2019), and Gregor, Rezende, and Wierstra (2016) associate the distribution of observed end-states with a noise

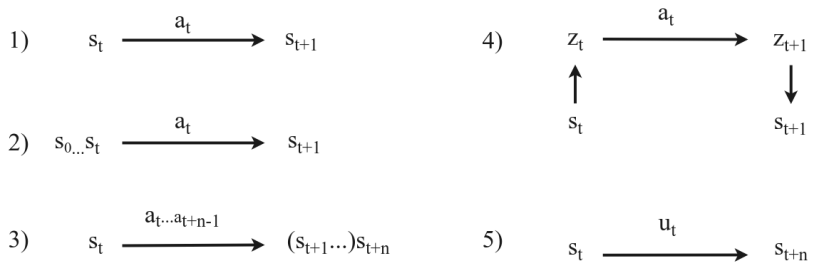


Figure 4.6: Overview of different types of mappings in model learning. **1)** Standard Markovian transition model $s_t, a_t \rightarrow s_{t+1}$. **2)** Partial observability (Section 4.3.4). We model $s_0...s_t, a_t \rightarrow s_{t+1}$, leveraging the state history to make an accurate prediction. **3)** Multi-step prediction (Section 4.3.6), where we model $s_t, a_t...a_{t+n-1} \rightarrow s_{t+n}$, to predict the n step effect of a sequence of actions. **4)** State abstraction (Section 4.3.7), where we compress the state into a compact representation z_t and model the transition in this latent space. **5)** Temporal/action abstraction (Section 4.3.8), better known as hierarchical reinforcement learning, where we learn an abstract action u_t that brings us to s_{t+n}. Temporal abstraction abstraction directly implies multi-step prediction, as otherwise the abstract action u_t is equal to the low level action a_t. All the above ideas (2-5) are orthogonal and can be combined.

distribution. After training, the noise distribution acts as a high-level action space from which we can sample. Various approaches also include additional information-theoretic regularization of this compression. For example, Gregor, Rezende, and Wierstra (2016) add the criterion that action sequences in the compressed space should make the resulting state well predictable ('empowerment'). Other examples are provided by Florensa, Duan, and Abbeel (2017), Fox, Moshkovitz, and Tishby (2016), and Hausman et al. (2018).

- *Reward relevancy*: The idea of this approach is that relevant subroutines will help incur extra reward, and they should therefore automatically emerge from a black-box optimization approach. These approaches embed the structure of subroutines into their algorithms, ensure that the overall model is differentiable, and run an end-to-end optimization. Examples are the Option-Critic (Bacon, Harb, and Precup, 2017; Riemer, Liu, and Tesauro, 2018)

and Feudal Networks (Vezhnevets et al., 2017), with more examples in Frans et al. (2018), Heess et al. (2016), Levy, Platt, and Saenko (2019), and Nachum et al. (2018). Daniel et al. (2016) and Fox et al. (2017) use probabilistic inference based on expectation-maximization, where the E-step infers which options are active, and the M-step maximizes with respect to the value. A challenge for end-to-end approaches is ensuring diversity, i.e., preventing that a single subroutine starts to solve the entire task (or that every subroutine terminates after one step).

- *Priors:* Finally, we may also use prior knowledge to identify useful subroutines. Sometimes, the prior knowledge is domain-specific, like pre-training on hand-coded sub-tasks (Heess et al., 2016; Tessler et al., 2017). Kulkarni et al. (2016) identify all objects in the scene as end-points, which may generalize over domains when combined with a generic object recognizer. Several papers also infer relevant subroutines from expert demonstrations (Fox et al., 2017; Hamidi et al., 2015; Konidaris et al., 2012), which is of course also a form of prior knowledge.

This concludes our discussion of temporal abstraction, and of model learning as a whole. As a summary, Figure 4.6 present a conceptual overview of four of the challenges we discussed (partial observability, multi-step prediction, state abstraction and temporal abstraction), and the type of connectivity that they require. As we have seen, there are a variety of challenges and issues in model learning. In the next section, we will discuss how this learned model may actually be used to act and learn in the environment.

4.4 INTEGRATION OF PLANNING AND LEARNING

The importance of models for intelligence has been long recognized in various research fields: machine learning (Bellman, 1966; Jordan and Rumelhart, 1992), neuroscience (Doll, Simon, and Daw, 2012; Tolman, 1948) and behavioural psychology (Craik, 1943; Doll, Simon, and Daw, 2012; Wolpert, Ghahramani, and Jordan, 1995). In this section we will discuss the integration of planning and learning to arrive at a policy $\pi(a|s)$, i.e., a local or global specification of action prioritization. We will specify a framework that disentangles the essential questions in the integration of planning and learning. The four main questions we need to answer are:

1. At which state do we start planning? (Sec. 4.4.1)
2. How much planning budget do we allocate for planning and real data collection? (Sec. 4.4.2)
3. How to plan? (Sec. 4.4.3)
4. How to integrate planning in the learning and acting loop? (Sec. 4.4.4)

These dimensions each have several important subconsiderations. The overall framework is summarized in Table 4.2, and will be discussed in the following sections.

4.4.1 *At which state to start planning?*

A model allows us to plan over it. The natural first question is: at which state shall we start planning? There are several options:

- *Random*: A straightforward approach is to randomly select states throughout state space. This is for example the approach of Dynamic Programming (Bellman, 1966), which selects all possible states in a sweep. The major drawback of this approach is that it does not scale to high dimensional problems, since the total number of states grows exponentially in the dimensionality of the state space. The problem is that we will likely update many states that are not even reachable from the start state.
- *Visited*: We may ensure that we only plan at reachable states by selecting previously visited states as starting points. This approach is for example chosen by Dyna (Sutton, 1990).
- *Prioritized*: Sometimes, we may be able to obtain an ordering over the reachable states, identifying their relevancy for a next planning update. A good example is Prioritized Sweeping (Moore and Atkeson, 1993), which identifies states that likely need updating. Prioritization is also popular in replay database, but these are not considered model-based RL.
- *Current*: Finally, a common approach is to only spend planning effort at the current state of the real environment. This puts much emphasis at finding a better solution or more information in the region where we are currently operating. Even model-based RL methods with a known model, like AlphaGo Zero (Silver

Table 4.2: Overview of dimensions of planning-learning integration. These considerations are discussed throughout Sec. 4.4. Table 4.3 summarizes several model-based RL algorithms on these dimensions.

Dimension	**Consideration**	**Choices**
1. Start state (4.4.1)	- Start state	Random ↔ visited ↔ prioritized ↔ current
2. Budget (4.4.2)	- Number of real steps before planning	1 ↔ n, episode, etc.
	- Effort per planning cycle	1 ↔ n ↔ convergence
3. Planning approach (4.4.3)	- Type	Discrete ↔ gradient-based
	- Direction	Forward ↔ Backward
	- Breadth	1 ↔ adaptive ↔ full
	- Depth	1 ↔ interm./adaptive ↔ full
	- Uncertainty	Data-close ↔ Uncertainty propagation (-Prop.method: parametric ↔ sample)
4. Integration in learning loop (4.4.4)	- Planning input from learned function	Yes (value/policy) ↔ No
	- Planning output for training targets	Yes (value/Policy) ↔ No
	- Planning output for action selection	Yes ↔ No

et al., 2017c), sometimes voluntarily introduce the notion of a real environment and current state. The real environment step introduces a form of pruning, as it ensures that we move forward at some point, obtaining information about deeper nodes (see Moerland, Broekens, and Jonker, 2020a as well).

4.4.2 *How much budget do we allocate for planning and real data collection?*

We next need to decide i) after how many real environment steps we start to plan, and ii) once we start planning, what budget we allocate? Together, these two questions determine an important trade-off in model-based RL.

WHEN TO START PLANNING? We first need to decide how many real steps we will make before a new planning cycle. Many approaches plan after every real environment step. For example, Dyna (Sutton, 1990) makes up to a hundred planning steps after every real step. Other approaches collect a larger set of data before they start to plan. For example, PILCO (Deisenroth and Rasmussen, 2011) collects data in entire episodes, and replans an entire solution after a set of new real transitions has been collected. The extreme end of this spectrum is *batch* reinforcement learning (Lange, Gabel, and Riedmiller, 2012), where we only get a single batch of transition data from a running system, and we need to come up with a new policy without being able to interact with the real environment. Some methods may both start with an initial batch of data to estimate the model, but also interact with the environment afterwards (Watter et al., 2015).

HOW MUCH TIME TO SPEND ON PLANNING? Once we decide to start planning, the second key question is: how much planning budget do we allocate. We define a planning cycle to consist of multiple planning iterations, where each iteration is defined by fixing a new planning start state. The total planning effort is then determined by two factors: i) how many times do we fix a new start state (i.e., start a new planning iteration), and ii) how much effort does each iteration get?

We will use Dyna (Sutton, 1990) and AlphaGo Zero (Silver et al., 2017c) as illustrative examples of these two questions. In between every real environment step, Dyna samples up to a 100 one-step transitions. This means we have 100 planning iterations, each of budget 1. In contrast, in between every real step AlphaGo Zero does a single MCTS

iteration, which consists of 1600 traces, each of approximate depth 200. Therefore, AlphaGo Zero performs 1 planning iteration, of budget $\sim 1600 * 200 = 320.000$. The total budget per planning cycle for Dyna and AlphaGo Zero are therefore 100 and ~ 320.000, respectively. Note that we measure planning budget as the number of model calls here, while the true planning effort of course also depends on the computational burden of the planning algorithm itself.

Some approaches, especially the ones that target high data efficiency (see Sec. 4.6.1) in the real environment, allow for a very high planning budget once they start planning. These methods for example plan until convergence on an optimal policy (given the remaining uncertainty) (Deisenroth and Rasmussen, 2011). We call this a *squeezing* approach, since we attempt to squeeze as much information out of the available transition data as possible. We further discuss this approach in Sec. 4.6.1.

ADAPTIVE TRADE-OFF Our choice on the above two dimensions essentially specifies a trade-off between planning and real data collection, with model-free RL (no planning effort) and exhaustive search (infinite planning effort) on both extremes. Most model-based RL approaches set the above two considerations to fixed (intermediate) values. However, humans clearly make a much more adaptive trade-off, where they adaptively decide a) when to start planning, and b) how much time to spend on that plan (i.e., the two considerations discussed above). This has indeed been an active topic of research in human psychology as well. Keramati, Dezfouli, and Piray (2011) present this as a speed/accuracy trade-off, where habitual, reactive behaviour (no planning) is fast but potentially inaccurate, while extensive planning is slow but more accurate. See Hamrick (2019) for a more detailed discussion. We also return to this topic in Sec. 4.6.3.

A few authors have investigated an adaptive trade-off between planning and acting in model-based RL. Pascanu et al. (2017) add a small penalty for every planning step to the overall objective, which ensures that planning should provide reward benefit. This approach is very task specific. Hamrick et al. (2017) learn a meta-controller over tasks that learns to select the planning budget per timestep. In contrast to these optimization-based approaches, Kalweit and Boedecker (2017) derive the ratio between real and planned data from the variance of the estimated Q-function. When the variance of the Q-function is high, they sample additional data from the model. This ensures that they only use ground-truth data near convergence, but accept noisier model-based

data in the beginning. Lu, Mordatch, and Abbeel (2019) propose a similar idea based on the epistemic uncertainty of the value function, by also increasing planning budgets when the uncertainty rises above a threshold. However, when we have a learned model, we probably do not want to plan too extensively in the beginning of training either (since the learned model is then almost random), so there are clear open research questions here.

4.4.3 *How to plan?*

The third crucial consideration is: how to actually plan? Of course, we do not aim to provide a full survey of planning methods here, and refer the reader to Moerland, Broekens, and Jonker (2020a) for a recent framework to categorize planning and RL methods. Instead, we focus on some crucial decisions we have to make for the integration, on a) the use of potential differentiability of the model, b) the direction of planning, c) the breadth and depth of the plan, and d) the way of dealing with uncertainty.

TYPE One important distinction between planning methods is whether they require differentiability of the model:

- *Discrete planning*: This is the main approach in the classic AI and reinforcement learning communities, where we make discrete back-ups which are stored in a tree, table or used as training targets to improve a value or policy function. We can in principle use any preferred planning method. Examples in the context of model-based RL include the use of probability-limited search (Lai, 2015), breadth-limited depth-limited search (François-Lavet et al., 2019), Monte Carlo search (Silver, Sutton, and Müller, 2008), Monte Carlo Tree Search (Anthony, Tian, and Barber, 2017; Jiang, Ekwedike, and Liu, 2018; Moerland et al., 2018a; Silver et al., 2017c), minimax-search (Baxter, Tridgell, and Weaver, 1999; Samuel, 1967), or a simple one-step search (Sutton, 1990). These methods do not require any differentiability of the model.
- *Differential planning*: The gradient-based approach requires a differentiable model. If the transition and reward models are differentiable, and we specify a differentiable policy, then we can directly take the gradient of the cumulative reward objective with respect to the policy parameters. While a real world environment or sim-

ulator is by definition not differentiable, our learned model of these dynamics (for example a neural network) usually is differentiable. Therefore, model-based RL can suddenly utilize differential planning methods, exploiting the differentiability of the learned model. Note that differentiable models may also be obtained from the rules of physics, for example in differentiable physics engines (Avila Belbute-Peres et al., 2018; Degrave, Hermans, Dambre, et al., 2019).

A popular example is the use of iterative linear quadratic regulator planning (Todorov and Li, 2005), which requires a linear model, and was for example used as a planner in Guided Policy Search (Levine and Koltun, 2013). In the RL community, the gradient-based planning approach is better known as *value gradients* (Fairbank and Alonso, 2012; Heess et al., 2015). Successful examples of model-based RL that use differential planning are PILCO (Deisenroth and Rasmussen, 2011), which differentiates through a Gaussian Process dynamics model, and Dreamer (Hafner et al., 2019a) and Temporal Segment Models (Mishra, Abbeel, and Mordatch, 2017), which differentiate through a (latent) neural network dynamics model.

Gradient-based planning is especially popular in the robotics and control community, since it requires relatively smooth underlying transition and reward functions. In those cases, it can be very effective. However, it is less applicable to discrete problems and sparse reward functions.

DIRECTION We also have to decide on the direction of planning (see also Sec. 4.3.1):

- *Forward*: Forward simulation (lookahead) is the standard approach in most planning and model-based RL approaches, and actually assumed as a default in all other paragraphs of this section. We therefore do not further discuss it.
- *Backward*: We may also learn a reverse model, which tells us which state-action pairs lead to a particular state ($s' \rightarrow s, a$). A reverse model may help spread information more quickly over the state space. This idea is better known as *Prioritized sweeping* (PS) (Moore and Atkeson, 1993). In PS, we track which state-action value estimates have changed a lot, and then use the reverse model to identify their possible precursors, since the estimates of these

state-actions are now likely to change as well. This essentially builds a search tree in the backward direction, where the planning algorithm follows the direction of largest change in value estimate.

Various papers have shown the benefit of prioritized sweeping with tabular models (Dearden, Friedman, and Andre, 1999; Moore and Atkeson, 1993; Wiering and Schmidhuber, 1998), which are trivial to invert. These ideas were extended to linear approximation (Sutton et al., 2012) and nearest-neighbour approximation of the dynamics (Jong and Stone, 2007) as well. More recently, backward models were studied in high-dimensional problems using neural network approximation, for example by Agostinelli et al. (2019), Edwards, Downs, and Davidson (2018), and Corneil, Gerstner, and Brea (2018).

BREADTH AND DEPTH A new planning iteration starts to lookahead from a certain start state. We then still need to decide on the the breadth and the depth of the lookahead. For model-free RL approaches, breadth is not really a consideration, since we can only try a single action in a state (a breadth of one). However, a model is by definition reversible, and we are now free to choose and adaptively balance the breadth and depth of the plan. We will list the possible choices for both breadth and depth, which are summarized in Figure 4.7.

For the breadth of the plan, there are three main choices:

- *Breadth = 1*: These methods only sample single transitions or individual traces from the model, and still apply model-free updates to them. Therefore, they still use a breadth of one. The cardinal example of this approach is Dyna (Sutton, 1990), which sampled additional one-step data for model-free Q-learning (Watkins and Dayan, 1992) updates. More recently, Kalweit and Boedecker (2017) applied the above principle to deep deterministic policy gradient (DDPG) updates, Kurutach et al. (2018) to trust region policy optimization (TRPO) updates and Gu et al. (2016) to normalized advantage function (NAF) updates.

- *Breadth = adaptive*: Many planning methods adaptively scale the breadth of planning. The problem is of course that we cannot afford to go full breadth and full depth, because exhaustive search is computationally infeasible. A method that adaptively scales the breadth of the search is for example Monte Carlo Tree Search (Browne et al., 2012), by means of the upper confidence bounds

formula. This ensures that we do go deeper in some arms, before going full wide at the levels above. This approach was for example applied in AlphaGo Zero (Silver et al., 2017c).

- *Breadth = full*: Finally, we may of course go full wide over the action space, before we consider searching on a level deeper. This is for example the approach of Dynamic Programming, which goes full wide with a depth of one. In the context of model-based RL, few methods have taken this approach.

For the depth of the plan, there are four choices:

- *Depth = 1*: We may of course stop after a depth one. For example, Dyna (Sutton, 1990) sampled transition of breadth one and depth one.
- *Depth = intermediate*: We may also specify an intermediate search depth. RL researchers have looked at balancing the depth of the back-up for long, since it trades off bias against variance (a shallow back-up has low variance, while a deep back-up is unbiased). In the context of Dyna, Holland, Talvitie, and Bowling (2018) explicitly studied the effect of deeper roll-outs, showing that traces longer than depth 1 give better learning performance. Of course, we should be careful that deeper traces do not depart from the region where the model is accurate.
- *Depth = adaptive*: Adaptive methods for depth go together with adaptive methods for breadth. For example, a MCTS tree does not have a single depth, but usually has a different depth for many of its leafs.
- *Depth = full*: This approach samples traces in the model until an episode terminates, or until a large horizon. PILCO and Deep PILCO for example sample deep traces from their models (Gal, McAllister, and Rasmussen, 2016).

This is of course a very shallow treatment of the crucial breadth versus depth balancing in planning, which has a close relation to exploration methods as well. However, the focus of this survey is the integration of planning and learning, not the actual planning method itself. From a model-based RL perspective, the crucial realization is that compared to model-free RL, we can suddenly use a breadth larger than one. Nevertheless, many model-based RL methods still choose to stick to a

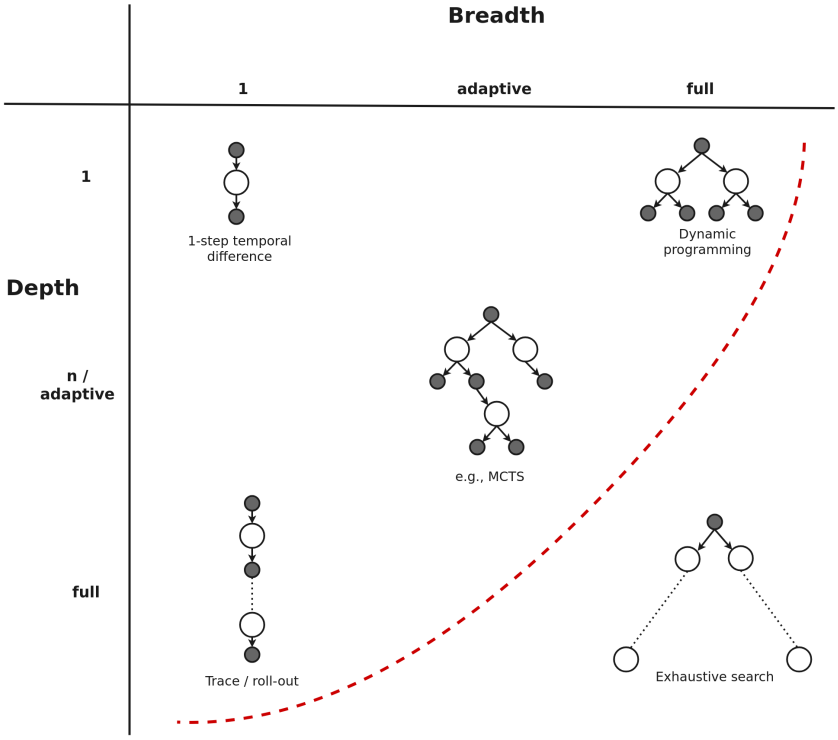


Figure 4.7: Breadth and depth of a single planning iteration. For every planning iteration, we need to decide on the breadth and depth of the lookahead. In practice, planning iterations usually reside somewhere left of the red dashed line, since we cannot afford a full breadth, full depth (exhaustive) search. Most planning methods, like MCTS, adaptively balance breadth and depth throughout the tree, where the breadth and depth differ throughout the tree. Figure is based on Sutton and Barto, 2018, who used it to categorize different types of back-ups. A single planning iteration, which we define by fixing a new root state, can indeed be seen as a large back-up.

breadth of one in their model samples, likely because this gives seamless integration with model-free updates. We further discuss this topic in Sec. 4.6.1.

DEALING WITH UNCERTAINTY When we plan over a learned model, we usually also need to deal with the uncertainty of a learned model. There are two main approaches:

- *Data-close planning*: The first approach is to ensure that the planning iterations stay close to regions where we have actually observed data. For example, Dyna (Sutton, 1990) samples start states at the location of previously visited states, and only samples one-step transitions, which ensures that we do not depart from the known region of state space. Other approaches, like Guided Policy Search (Levine and Abbeel, 2014), explicitly constraint the new plan to be close to the current policy (which generated the data for the model).

- *Uncertainty propagation*: We may also explicitly estimate model uncertainty, which allows us to robustly plan over long horizons. Once we depart too far from the observed data, model uncertainty will increase, predictions will start to spread out over state space, and the learning signal will naturally vanish. Estimation of model uncertainty was already discussed in Sec. 4.3.3. We will here focus on propagation of uncertainty over timesteps, since the next state uncertainty is of course conditioned on the uncertainty of the previous step. There are two main propagation approaches:

  - *Parametric*: This propagation method fits a parametric distribution to the uncertainty at every timestep. This approach is for example used by PILCO (Deisenroth and Rasmussen, 2011), which derives closed form analytic expressions to track the uncertainty. However, analytic parametric propagation is not possible for more complicated models, like for example neural networks.

  - *Sample-based*: This propagation approach, also known as *particle methods*, tracks the distributions of uncertainty by propagating a set of particles forward. The particles together represent the predicted distribution at a certain number of steps. Particle methods are for example used in Deep PILCO (Gal, McAllister, and Rasmussen, 2016) and PETS (Chua et al., 2018). Note that fitting to a distribution, or matching moments of distributions, may have a regularizing effect. Therefore, Deep PILCO (Gal, McAllister, and Rasmussen, 2016) does propagate particles through the dynamics function, but then refits these particles to a (Gaussian) distribution at every step. See Chua et al. (2018) for a broader discussion of uncertainty propagation approaches.

We may also use uncertainty to determine the *depth* of our value estimates. *Stochastic ensemble value expansion* (STEVE) (Buckman et al., 2018) reweights value targets of different depths according to their associated uncertainty, which is derived from both the value function and transition dynamics uncertainty. Thereby, we base our value estimates on those predictions which have highest confidence.

This concludes our discussion of the actual planning approach in planning-learning integration. As mentioned before, there are many more considerations in a planning algorithm, like managing exploration (balancing breadth and depth in the search tree). However, these are not challenges of planning-learning integration, and therefore not further covered in this section.

4.4.4 *How to integrate planning in the learning and acting loop?*

We have now specified how to plan (the start point, budget and planning method). However, we still need to integrate planning in the larger learning and acting loop. Figure 4.8, which we already introduced in the Introduction chapter (Fig. 1.1), presents a conceptual overview of the overall training loop. We have so far focused on the planning box (arrow a), but we will now focus on the connection of planning to other aspects of the learning loop. These include: i) directing new planning iterations based on learned knowledge in value or policy functions (Fig. 4.8, arrow b), ii) using planning output to update learned value or policy functions (Fig. 4.8, arrow c), and iii) using planning output to select actions in the real world (Fig. 4.8, arrow d).

PLANNING INPUT FROM LEARNED FUNCTIONS The learned value or policy functions may store much information about the current environment, which may help to focus the next planning iteration. We distinguish the use of value and policy information:

- *Value priors*: The most common way to incorporate value information is through *bootstrapping* (Sutton and Barto, 2018), where we plug in the current prediction of a state or state-action value to prevent having to search deeper (reducing the depth of the search). Various model-based RL algorithm use bootstrapping in their planning approach, for example Baxter, Tridgell, and Weaver (1999), Jiang, Ekwedike, and Liu (2018), Moerland et al. (2018a),

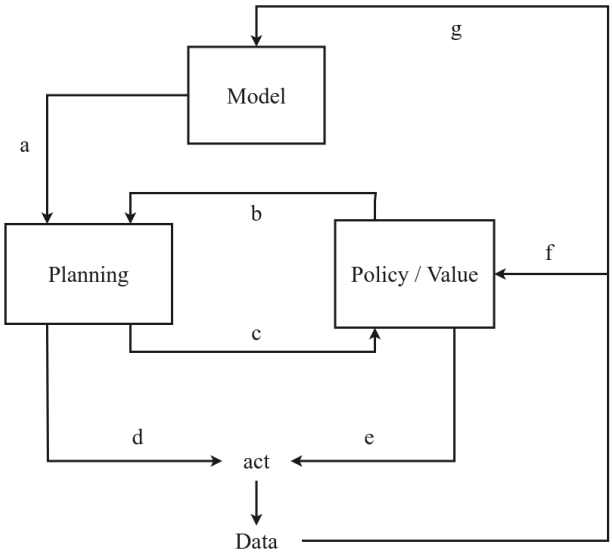


Figure 4.8: Procedural details of planning/learning integrations. Arrows (numbered a-g) indicate possible connections. a) plan over a learned model, b) use information from a policy/value network to improve the planning procedure, c) use the result from planning as training targets for a policy/value, d) act in the real world based on the planning outcome, e) act in the real world based on a policy/value function, f) generate training targets for the policy/value based on real world data, g) generate training targets for the model based on real world data. Most algorithms only implement a subset of these connections. See Figure 4.9 for an illustration of the subsets used by different algorithms.

and Silver et al. (2017c). Note that bootstrapping is also a very common principle in model-free RL. We may also use the learned value function to initialize the values of the action nodes at the root of the search (Hamrick et al., 2020; Silver, Sutton, and Müller, 2008), which we could interpret as a form of bootstrapping at depth 0.

- *Policy priors*: We can also leverage a learned policy in a new planning iteration. Several ideas have been proposed. AlphaGo Zero (Silver et al., 2017c) uses the probability of an action as a prior multiplication term on the upper confidence bound term in MCTS planning. This gives extra exploration pressure to actions with

high probability under the current policy network. Guided Policy Search (GPS) (Levine and Koltun, 2013) penalizes a newly planned trajectory for departing too much from the trajectory generated by the current policy network. As a final example, Guo et al. (2014) let the current policy network decide at which locations to perform the next search, i.e., the policy network influences the distribution of states used as a starting point for planning (Sec. 4.4.1, a form of prioritization). There are various ways in which we may incorporate prior knowledge from a policy, and there seems to be open research ground to identify the best of these approaches.

PLANNING UPDATE FOR POLICY OR VALUE UPDATE Model-based RL methods eventually seek a global approximation of the optimal value or policy function. The planning result may be used to update this global approximation. We generally need to i) construct a training target from the search, and ii) define a loss for training. We again discuss value and policy updates separately:

- *Value update*: A typical choice for a value target is the state(-action) value estimate at the root of the search tree. The estimate of course depends on the back-up policy, which can either be on- or off-policy. For methods that do not go wide over the actions, like Dyna (Sutton, 1990), we may use a classic Q-learning target (one-step, off-policy). For planning cycles that do go wide (and deep), we can combine on- and off-policy back-ups throughout the tree in various ways. Willemsen, Baier, and Kaisers (2020) present a recent study of the different types of back-up policies in a tree search. After constructing the value target, the value approximation is usually trained on a *mean-squared error* (MSE) loss (Moerland et al., 2018a; Veness et al., 2009). However, other options are possible as well, like a cross-entropy loss between the softmax of the Q-values from the search and the Q-values of a learned neural network (Hamrick et al., 2020).
- *Policy update*: For the policy update we again observe a variety of training targets and losses, depending on the type of planning procedure that is used. For example, AlphaGo Zero (Silver et al., 2017c) uses MCTS planning, and constructs a policy training target by normalizing the visitation counts at the root node. The policy network is then trained on a cross-entropy loss with this distribution. Guo et al. (2014) apply the same idea with a one-hot encoding

of the best action, while Moerland et al. (2018a) cover an extension to a loss between discrete counts and a continuous policy network. As a completely different approach, Guided policy search (GPS) (Levine and Abbeel, 2014) minimizes the Kullback-Leibler (KL)-divergence between a planned trajectory and the output of the policy network. Some differential planning approaches also directly update a differentiable global representation (Deisenroth and Rasmussen, 2011).

We may also train a policy based on a value estimate. For example, Policy Gradient Search (PGS) (Anthony et al., 2019) uses the policy gradient theorem (Williams, 1992) to update a policy from value estimates in a tree. Note that gradient-based planning (discussed in Sec. 4.4.3) also belongs here, since it directly generates gradients to update the differentiable policy.

Most of the above methods construct training targets for value or policy at the root of the search. However, we may of course also construct targets at deeper levels in the tree (Veness et al., 2009). This extracts more information from the planning cycle. Many papers update their value or policy from both planned and real data, but other papers exclusively train their policy or value from planning (Deisenroth and Rasmussen, 2011; Depeweg et al., 2016; Ha and Schmidhuber, 2018; Kurutach et al., 2018), using real data only to train the dynamics model.

Note that arrows b and c in Figure 4.8 form a closed sub-loop in the overall integration. There has been much recent interest in this sub-loop, which iterates planning based on policy/value priors (arrow b), and policy/value learning based on planning output (arrow c). A successful algorithms in this class is AlphaGo Zero (Silver et al., 2017c), which is an instance of *multi-step approximate real-time dynamic programming* (MSA-RTDP). MSA-RTDP extends the classic DP ideas by using a 'multi-step' lookahead, learning the value or policy ('approximate'), and operating on traces through the environment ('real-time'). Efroni, Ghavamzadeh, and Mannor (2019) theoretically study MSA-RTDP, showing that higher planning depth d decreases sample complexity in the real environment at the expense of increased computational complexity. Although this result is intuitive, it does show that planning may lead to better informed real-world decisions, at the expense of increased (model-based) thinking time. In addition, iterated planning and learning may also lead to more stable learning, which we discuss in Sec. 4.6.3.

PLANNING OUTPUT FOR ACTION SELECTION IN THE REAL ENVIRONMENT We may also use planning to select actions in the real environment. While model-free RL has to use the value or policy approximation to select new action in the environment (Fig. 4.8, arrow e), model-based RL may also select actions directly from the planning output (Fig. 4.8, arrow d). Some methods only use planning for action selection, not for value/policy updating (Silver, Sutton, and Müller, 2008; Tesauro and Galperin, 1997), for example because planning updates can have uncertainty. However, many methods actually combine both uses (Anthony, Tian, and Barber, 2017; Moerland et al., 2018a; Silver et al., 2018, 2017c).

Selection of the real-world actions may happen in a variety of ways. First of all, we may greedily select the best action from the plan. This is the typical approach of methods that 'plan over a learned model' (Table 4.1). The cardinal example in this group are *model predictive control* (MPC) or *receding horizon control* approaches. In MPC, we find the greedy action of a k-step lookahead search, execute the greedy action, observe true next state, and repeat the same procedure from there. The actual planning algorithm in MPC may vary, with examples including iLQR (Watter et al., 2015), direct optimal control (Chua et al., 2018; Nagabandi et al., 2018b), Dijkstra's algorithm (Kurutach et al., 2018), or repeated application of an inverse model (Agrawal et al., 2016). MPC approaches do not use learning for the value or policy function, and are therefore not model-based RL. MPC easily deals with (changing) constraints on the state and action space (Kamthe and Deisenroth, 2017), and is especially popular in robotics and control tasks.

Note that we do not have to execute the greedy action after planning. Some approaches introduce additional exploration noise over the greedy planning action (Silver et al., 2017c). Other methods intentionally include exploration criteria for their real action. For example, Dearden, Friedman, and Russell (1998) explore based on the 1value of perfect information' (VPI), which estimates from the model what exploratory action has the highest potential to change the greedy policy. Indeed, we may actually 'plan for exploration', i.e., decide which exploratory sequence of real world actions is most promising by means of planning (Lowrey et al., 2018; Sekar et al., 2020). Planning may identify temporally correlated action sequences that perform deep exploration towards new reward regions, which local exploration methods would fail to identify due to jittering behaviour (Osband et al., 2016). We call this approach *two-phase exploration* (first exploring within a plan, then

exploring in the real, irreversible environment), and extensively discuss the benefits of this idea in Sec. 4.6.

This concludes our discussion of the main considerations in planning-learning integration. Table 4.2 summarizes the framework, showing the potential decisions on each dimension.

4.4.5 *Conceptual comparison of approaches*

This chapter discussed the various ways in which planning and learning can be integrated. We will present two summaries of the discussed material. First of all, Figure 4.9 summarizes the different types of connectivity that may be present in planning-learning integration. The figure is based on the scheme of Figure 4.8, as used throughout this section, and the classification of model-based RL methods described in Table 4.1.

We see how well-known model-based RL algorithms like Dyna (Sutton, 1991) and AlphaGo Zero (Silver et al., 2017c) use different connectivity. For example, Dyna learns a model, which AlphaGo Zero assumes to be known, and AlphaGo Zero select actions from planning, while Dyna uses the learned value approximation. The bottom row shows Embed2Control (Watter et al., 2015), a method that only plans over a learned model, and completely bypasses any global policy or value approximation. For comparison, the bottom-right of the figure shows a model-free RL approach, like DQN (Mnih et al., 2015) or SARSA (Rummery and Niranjan, 1994) with eligibility traces.

As a second illustration, Table 4.3 compares several well-known model-based RL algorithms on the dimensions of our framework for planning-learning integration (Table 4.2). We see how different integration approaches make widely different choices on each of the dimensions. It is hard to judge whether some integration approaches are better than others. These aspects also partially depend on the problem type at hand. For example, when we have large computational resources available, we may be able to afford the high planning budget per timestep used by AlphaGo Zero (Silver et al., 2017c), which aims for high asymptotic performance. Gradient-based planning can be useful, but is mostly applicable to continuous control tasks. Backward planning (prioritized sweeping) can be useful, but does require us to learn a backward model. For many considerations, there are both pros and cons. Usually, the eventual decision depends on the type of benefit

(of model-based RL) we aim for, which will be discussed in the next section.

4.5 IMPLICIT MODEL-BASED RL

We have so far discussed the two key steps of model-based RL: 1) model learning and 2) planning over the model to recommend an action or improve a learned policy or value function. All the methodology discussed so far was *explicit*, in a sense that we manually designed each step of the process. This is the classical, explicit approach to model-based RL (and to algorithm design in general), in which we manually design the individual elements of the algorithms.

An interesting observation about the above process is that, although we may manually design various aspects of the model-based RL algorithm, we ultimately only care about one thing: identifying the (optimal) value or policy. In other words, the entire model-based RL procedure (model learning, planning, and possibly integration in value/policy approximation) can from the outside be seen as a model-free RL problem. Eventually, we want our entire system to be able to predict an (optimal) action or value. This intuition leads us to the field of *implicit* model-based RL. The common idea underneath all these approaches is to take one or more aspects of the model-based RL process and optimize these for the ultimate objective, i.e., (optimal) value or policy computation.

In particular, we will focus on methods that use gradient-based optimization. In those case, we embed (parts of) the model-based RL process within a computational graph, which eventually outputs a value or action recommendation. Since the graph remains end-to-end differentiable, we may optimize one or more elements of our model-based RL procedure for a value or action recommendation. One would be tempted to call the field *end-to-end* model-based RL, but not that the underlying principles are more general, and could also work with gradient-free optimization.

We may use implicit model-based RL to replace each (or both) of the steps of explicit model-based RL: 1) to learn implicit transition models, better known as *value equivalent models* (Sec. 4.5.1), and 2) to implicitly *learn how to plan* (Sec. 4.5.2). We will first discuss each category individually, and afterwards discuss how they can also be combined (Sec. 4.5.3). An overview of the ideas and papers in the following sections is provided in Table 4.4.

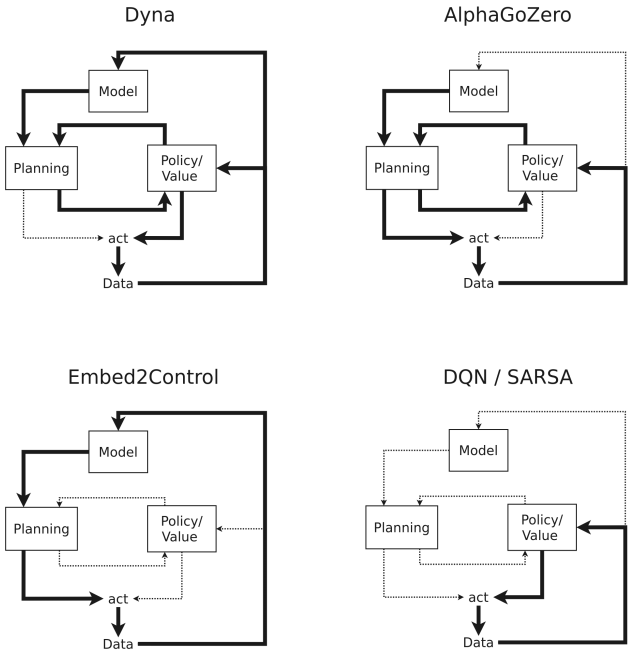


Figure 4.9: Comparison of planning and learning algorithms, based on the general visualization of learning/planning integration from Figure 4.8. Thick lines are used by an algorithm. Dyna (Sutton, 1991) (top-left) is an example of model-based RL with a learned model. AlphaGo Zero (Silver et al., 2017c) (top-right) is an example of model-based RL with a known model. Note that therefore the model does not need updating from data. Embed2Control (Watter et al., 2015) (bottom-left) is an example of planning over a learned model. For comparison, the bottom right shows a model-free RL algorithm, like Deep Q-Network (Mnih et al., 2015) or SARSA (Rummery and Niranjan, 1994) with eligibility traces

.

Table 4.3: Systematic comparison of different model-based RL algorithms on the dimensions of planning-learning integration (Sec. 4.4). Colour coding: green = model-based RL with a learned model, red = model-based RL with a known model, blue = planning over a learned model (see Table 4.1). Uncertainty estimation methods: GP = Gaussian Process, BE = bootstrap ensemble. Uncertainty propagation methods: Par = parametric propagation, Sam = sample-based propagation (particle methods). † = Before learning, the authors collect an initial batch of training data for the model. The number of real steps before the *first* plan is therefore 3.000-30.000, depending on the task. Afterwards, they start to interact with the environment, planning at every step. ⋆ = gradient-based planners improve a reference trajectory based on gradients. Although there is only trajectory, the gradient does implicitly go wide over the actions, since it tells us in which direction the continuous action should be moved.

Paper	**Start state**	**Budget**		**How to plan?**				**Integration within learning loop**		
		Real steps before plan	Budget per planning cycle	Type	Direction	Breadth & depth	Uncertainty	Input from value/policy	Output to value/policy	Output for action selection
Dyna (Sutton, 1990)	Visited	1	10-100 steps	Discrete	Forward	B=1, D=1	Data-close	V	V	-
Prioritized sweeping (Moore and Atkeson, 1993)	Prioritized	1	10 steps	Discrete	Backward	B=full, D=1	-	V	V	-
PILCO (Deisenroth and Rasmussen, 2011)	Start	Episode	↑↑ (convergence)	Gradient	Forward	B>1⋆, D=full	Uncertainty (GP + Par)	P	P	-
Guided policy search (Levine and Abbeel, 2014)	Current	Episode	5-20 rollouts	Gradient	Forward	B=5-40, D=full	Data-close	P	P	-
AlphaGo Zero (Silver et al., 2017c)	Current	1	∼320.000	Discrete	Forward	adaptive	-	V+P	P	✓
SAVE (Hamrick et al., 2020)	Current	1	10-20	Discrete	Forward	adaptive	-	V	V	✓
Embed2Control (Watter et al., 2015)	Current	1†	MPC depth 10	Gradient	Forward	B>1⋆, D=10	-	-	-	✓
PETS (Chua et al., 2018)	Current	1	MPC depth 10-100	Discrete	Forward	B>1, D=10-100	Uncertainty (BE + Sam)	P	-	✓

4.5.1 *Value equivalent models*

Standard model learning approaches, as discussed in Section 4.3, learn a forward model that predicts the next state of the environment. However, such models may predict several aspects of the state that are not relevant for the value. In some domains, the forward dynamics might be complicated to learn, but the aspects of the dynamics that are relevant for value prediction might be much smoother and easier to learn. This is the key insight below *value equivalent models* (Grimm et al., 2020). Value equivalent models are unrolled inside the computation graph to predict a future value, instead of a future state. As such, these models are enforced to emphasize value-relevant characteristics of the environment. Thereby, value equivalent models are really representation learning technique for model learning, as already mentioned at the end of Sec. 4.3.7. They can be combined with any other type of loss function as well, like the ability to predict a future reward.

An example of a successful value-equivalent approach is MuZero (Schrittwieser et al., 2019). During training, Muzero gets a state and action sequence as input, and internally unrolls its model to predict the multi-step, action-conditional value (i.e., on-policy). The training targets for these predictions are obtained from a model-free value estimate. The value-equivalent model can then be used in MCTS procedure, which achieved state-of-the-art performance in the Chess, Go and Shogi, matching or outperforming the performance of AlphaZero (Silver et al., 2018). Value Prediction Networks (VPN) (Oh, Singh, and Lee, 2017) take a very similar approach, but specify a b-best, depth-d search (where b and d are hyperparameters) on the model.

A slightly different approach is taken by the Predictron (Silver et al., 2017a). It only receives a state as input (not a sequence of actions), and internally unrolls its models to predict the value of that state. When we want to select an optimal action (plan), we can unroll the Predictron for each available action in a state. Note that MuZero, VPN and the Predictron all include a state encoding function that gets trained on the same value-equivalent target, which can therefore helps for representation learning (Sec. 4.3.7) as well. Moreover, all three unroll their model in an *implicit policy evaluation* setting, along a single trace. Planning, which includes policy improvement, is still explicit, and does not happen inside the computational graph.

IMPLICIT PLANNING We may also embed an entire planning procedure in a computational graph, which would include some form of a

policy improvement operation (like a maximization of actions). We will call the idea, embedding an entire planning loop inside a computational graph, *implicit planning*. When the planning operations in this graph are differentiable, we may still be able to use end-to-end differentiation.

An implicit planning graph should output either 1) the optimal action or policy or 2) the optimal value. We can therefore train them on two types of losses. In the first case, we may use an *imitation learning loss* with the ground-truth optimal action. The underlying idea is to train on a series of tasks for which we already known the optimal solution, and afterwards apply the obtained solution to new problems. Second, when we have no expert demonstrations available, we may let our planning graph output the optimal value, and train on a standard model-free RL target (*RL loss*). The standard model-free RL target will gradually start to estimate the optimal value, which will generate a training signal for our planning graph.

We may use the above idea to optimize for certain elements of the implicit planning graph. There are two main options: 1) optimize for the transition model that appears in the graph, which is again a form of a value-equivalent model, or 2) optimize for the actual planning operations in the graph. We will discuss each of these in the next sections.

VALUE EQUIVALENT MODELS FROM AN IMPLICIT PLANNING GRAPH

Two papers that optimize a value equivalent model in an implicit planning graph are Value Iteration Networks (VIN) (Tamar et al., 2016) and Universal Planning Networks (UPN) (Srinivas et al., 2018). Both papers embed a known, differentiable planning procedure in the graph (VINs embed value iteration, UPNs embed value-gradients). The entire planning procedure consist of multiple cycles through the planner, which within contains multiple passes through the transition (and reward) models. Both methods then optimize the model against either an imitation los with the ground-truth action or a standard RL loss.

This essentially learns a value-equivalent model, since our planner requires a model that is able to predict correct value information in order to identify the correct optimal action. However, the internal structure of VINs and UPNs does differ from MuZero, VPNs and the Predictron, since they do include policy improvement operations inside. Therefore, these value equivalent models may learn slightly different aspects of the dynamics (i.e., MuZero and VPNs train to make correct multi-step predictions everywhere, while VINs and UPNs would extra emphasize

aspects relevant to estimate the optimal policy). In the next section, we discuss the second application of implicit planning.

4.5.2 *Learning to plan*

We may also use the implicit planning idea to optimize for the planning operations themselves. So far, we encountered two ways in which learning may enter model-based RL: i) to learn a dynamics model (Sec. 4.3), and ii) to learn a value or policy function (from planning output) (Sec. 4.4). We now encounter a third level in which learning may enter model-based RL: to *learn to plan*. The idea is to optimize our planner over a sequence of tasks to eventually obtain a better planning algorithm, which is a form of meta-learning (Vanschoren, 2019).

Learning to plan is likely inspired by the success of end-to-end learning in the deep learning community. While manually designed features were for long the common approach in fields like computer vision, it turned out that end-to-end optimization was better able to find representations. The same idea can be extended to entire algorithms, which may be better constructed through optimization than through manual design. We may call this general approach *algorithmic function approximation* (Guez et al., 2019). Note that algorithmic function approximation differs from standard feedforward approximators, which compute the target in a single pass. Instead, algorithmic function approximators have a recurrent internal structure, and - importantly - *their predictions may improve given additional internal cycles*. They also differ from the standard use of recurrent neural networks (RNNs), which are typically used to deal with additional inputs or outputs (often in the time dimension). Instead, algorithmic function approximators have a fixed length input and output, but still perform internal cycles to compute the prediction.

We will discuss three examples of learning to plan: MCTSNets (Guez et al., 2018), Imagination-augmented agents (I2A) (Racanière et al., 2017), and Imagination-based planner (IBP) (Pascanu et al., 2017). MCTSNets optimize elements of the MCTS algorithm, like selection, back-up and final recommendation, against the ability to output the correct optimal action in the game Sokoban. The dynamics model in MCTSNets is assumed to be known. In contrast, both I2A and IBP first separately learn a standard forward dynamics model, as extensively discussed in Sec. 4.3. In the planning graph, I2A then learns how to aggregate the information in these roll-outs, and how this should influence the learned policy.

Both MCTSNets and I2A optimize only part of the planning procedure, but also leave some manual design, like the order of node expansion. Imagination-based planner (IBP) (Pascanu et al., 2017) takes learning to plan even a step further, by introducing a differentiable manager network that in each iteration decides 1) whether we want to continue planning from this state, and 2) from which node in the current tree this expansion should take place. The IBP graph is still fully differentiable, and trained against a combination of a standard RL loss and the internal cost of simulation. The latter ensures that the manager will not continue to plan forever, which is necessary because this planning algorithm really gets almost full freedom in its algorithmic planning space. The authors show that the agent indeed learns how to plan as well as for how long to plan.

4.5.3 *Combined learning of models and planning*

We may also combine both ideas introduced in the previous sections (value equivalent models and learning to plan). If we specify a parameterized differentiable model and a parameterized differentiable planning procedure, then we can optimize the resulting computational graph jointly for the model and the planning operations. This of course creates a harder optimization problem, since the gradients for the planner depend on the quality of the model, and vice versa. However, it is the most end-to-end approach to model-based RL we can imagine, as all aspects discussed in Sections 4.3 and 4.4 get wrapper into a single optimization.

A partially structured approach in this category is TreeQN (Farquhar et al., 2018). Like previous examples, TreeQN looks on the outside like a standard value network, but is internally structured like a planner. The planning algorithm of TreeQN unrolls itself up to depth d in all directions, and aggregates the output of these predictions through a back-up network. The back-up network outputs the value estimate for the input state, which is optimized against a standard RL loss. This approach internally optimizes both the model (used in the depth d lookahead) and part of the planner (in the form of the back-up aggregation). It is therefore a structured 'learning to plan' approach, although the planner does not have the same freedom as IBP from the previous section.

Full algorithmic freedom is provided by the Deep Repeated ConvLSTM (DRC) (Guez et al., 2019). The authors take the most black-box

approach possible, which they appropriately name 'model-free planning'. DRC is a high-capacity recurrent neural network, but does not have any planning or MDP specific internal structure. Instead, the DRC is repeatedly unrolled, and the output is optimized against the ability to predict a standard model-free RL objective. It is entirely up to the RNN to internally learn both an appropriate (value-equivalent) model and an appropriate planning procedure. The authors show that their final RNN indeed shows signs of planning characteristics, like a test performance that increased with additional computational time.

This concludes our discussion of implicit model-based RL. An overview of the discussed papers and methodology is presented in Table 4.4. The strength of the implicit model-based RL approach is tied to the strength of optimization in general, and other fields of machine learning have already shown that optimization may beat human design intuition (given enough data and computational resources). Moreover, value equivalent models may be beneficial in problems where dynamics are complicated, but the dynamics relevant for value estimation are much smoother.

However, the implicit approach has its challenges as well. For the value-equivalent transition models, all learned predictions focus on the value and reward information, which is derived from a scalar signal (the reward). These methods may therefore not capture all of the relevant characteristics of the environment, and this may become apparent when we face a new task (with a different reward function). A similar problem may occur for optimization of the planner, since we do not want it to exploit task-specific characteristics (like the knowledge that we should always plan towards the left side of the room). The real solution to these problems is to train on a wide variety of tasks (reward functions). This is of course computationally demanding, especially since the implicit model-based RL is already computationally demanding itself (the computational graphs grow very large, and the optimization can be unstable). Model-based RL therefore faces the same fundamental question as many other artificial intelligence and machine learning directions: to what extend should our systems incorporate human priors (*explicit*), or rely on black-box optimization instead (*implicit*).

4.6 BENEFITS OF MODEL-BASED REINFORCEMENT LEARNING

Model-based RL may provide several benefits, which we will discuss in this section. However, in order to identify benefits, we first need to

Table 4.4: Comparison of implicit model-based RL approaches. **Rows**: algorithm colour coding, yellow = explicit model-based RL (for comparison), green = value equivalent models (Sec. 4.5.1), red = learning to plan (Sec. 4.5.2), blue = combination of value equivalent models and learning to plan (Sec. 4.5.3). **Columns**: Implicit planning implies some form of policy improvement in the computation graph. Learning to plan implies that this improvement operation is actually optimized. For planning, we shortly mention the specific planning structure between brackets. MCTS = Monte Carlo Tree Search, iLQR = iterative Linear Quadratic Regulator, RNN = recurrent neural network.

Paper	**Model**			**Planning**		
	Known state-prediction	Learned state-prediction	Learned value-equivalent	Explicit	Implicit	Learning to plan
AlphaZero (Silver et al., 2018)	x			x (MCTS)		
Dyna (Sutton, 1990)		x		x (one-step)		
Embed to Control (E2C) (Watter et al., 2015)		x		x (iLQR)		
MuZero (Schrittwieser et al., 2019)			x	x (MCTS)		
Value prediction networks (VPN) (Oh, Singh, and Lee, 2017)			x	x (*b*-best, depth-*d* plan)		
Predictron (Silver et al., 2017a)			x	x (Roll-out)		
Value Iteration Networks (VIN) (Tamar et al., 2016)			x		x (value iteration)	
Universal Planning Networks (UPN) (Srinivas et al., 2018)			x		x (grad-based planning)	
MCTSNet (Guez et al., 2018)	x				x	x (MCTS aggregation)
Imagination-augmented agents (I2A) (Racanière et al., 2017)		x			x	x (Roll-out aggregation)
Imagination-based planner (IBP) (Pascanu et al., 2017)		x			x	x (Tree constr. + aggregation)
TreeQN (Farquhar et al., 2018)			x		x	x (Tree aggregation)
Deep Repeated ConvLSTM (DRC) (Guez et al., 2019)			x		x	x (RNN)

discuss performance criteria, and establish terminology about the two types of exploration in model-based RL.

PERFORMANCE CRITERIA There are two main evaluation criteria for (model-based) RL algorithms:

- *Cumulative reward/optimality*: the quality of the solution, measured by the expected cumulative reward that the solution achieves.
- *Time complexity*: the amount of time needed to arrive at the solution, which actually has three subcategories:
  - Real-world sample complexity: how many unique trials in the real (irreversible) environment do we use?
  - Model sample complexity: how many unique calls to a (learned) model do we use? This is an infrequently reported measure, but may be a useful intermediate.
  - Computational complexity: how much unique operations (flops) does the algorithm require.

Papers usually report learning curves, which show optimality (cumulative return) on the y-axis and one of the above time complexity measures on the x-axis. As we will see, model-based RL may actually be used to improve both measures.

We will now discuss the potential benefits of model-based RL (Figure 4.10). First, we will discuss enhanced data efficiency (Sec. 4.6.1), which uses planning (increased model sample complexity) to reduce the real-world sample complexity. Second, we discuss exploration methods that use model characteristics (Sec. 4.6.2). As a third benefit, we discuss the potential of model-based RL with a known model to reach higher asymptotic performance (optimality/cumulative reward) (Sec. 4.6.3). A fourth potential benefit is transfer (Sec. 4.6.4), which attempts to reduce the sample complexity on a sequence of tasks by exploiting commonalities. Finally, we also shortly touch upon safety (Sec. 4.6.5), and explainability (Sec. 4.6.6).

4.6.1 *Data Efficiency*

A first approach to model-based RL uses planning to reduce the real-world sample complexity. Real-world samples are expensive, both due to wall-clock time restrictions and hardware vulnerability. Enhanced data efficiency papers mostly differ by how much effort they invest per

planning cycle (Sec. 4.4.2). A first group of approaches tries to *squeeze* out as much information as possible in every planning loop. These typically aim for maximal data efficiency, and apply each planning cycle until some convergence criterion. Note that batch reinforcement learning (Lange, Gabel, and Riedmiller, 2012), where we only get a single batch of data from a running system and need to come up with an improved policy, also falls into this group. The second group of approaches continuously plans in the background, but does not aim to squeeze all information out of the current model.

- *Squeezing*: The squeezing approach, that plans from the current state or start state until (near) convergence, has theoretical motivation in the work on Bayes-adaptive exploration (Duff and Barto, 2002; Guez, Silver, and Dayan, 2012). All data efficiency approaches crucially need to deal with model uncertainty, which may be estimated with a Bayesian approach (Asmuth et al., 2009; Castro and Precup, 2007; Guez, Silver, and Dayan, 2012). These approaches are theoretically optimal in real world sample complexity, but do so at the expense of high computational complexity, and crucially rely on correct Bayesian inference. Due to these last two challenges, Bayes-adaptive exploration is not straightforward to apply in high-dimensional problems.

  Many empirical papers have taken the squeezing approach, at least dating back to Atkeson and Santamaria (1997) and Boone (1997). We will provide a few illustrative examples. A breakthrough approach was PILCO (Deisenroth and Rasmussen, 2011), which used Gaussian Processes to account for model uncertainty, and solved a real-world Cartpole problem in less than 20 seconds of experience. Both PETS (Chua et al., 2018) used a bootstrap ensemble to account for uncertainty, and scales up to a 7 degrees-of-freedom (DOF) action space, while model-based policy optimization (MBPO) (Janner et al., 2019), using a similar bootstrap ensemble for model estimation, even scales up to a 22 DOF humanoid robot (in simulation). Embed2Control (Wahlström, Schön, and Deisenroth, 2015) managed to scale model-based RL to a pixel input problem. Operating on a 51x51 pixel view of Pendulum swing-up, they show a 90% success rate after 15 trials of a 1000 frames each.

- *Mixing*: The second group of approaches simply mixes model-based updates with model-free updates, usually by making model-

based updates (in the background) throughout the (reachable) state space. The original idea dates back to the Dyna architecture of Sutton (1990), who reached improved data efficiency of up to 20-40x in a gridworld problem. In the context of high-dimensional function approximation, Gu et al. (2016) and Nagabandi et al. (2018b) used the same principle to reach a rough 2-5 times improvement in data efficiency.

An added motivation for the mixing approach is that we may still make model-free updates as well. Model-free RL generally has better asymptotic performance than model-based RL with a learned model. By combining model-based an model-free updates, we may speed-up learning with the model-based part, while still reaching the eventual high asymptotic performance of model-free updates. Note that model-based RL with a known model may actually reach higher asymptotic performance (Sec. 4.6.3) than model-free RL, which shows that the instability is really caused by the uncertainty of a learned model.

In short, model-based RL has a strong potential to increase data efficiency, by means of two-phase exploration. Strong improvements in data efficiency have been shown, but are not numerous, possibly due to the lack of stable uncertainty estimation in high-dimensional models, or the extensive amount of hyperparameter tuning required in these approaches. Nevertheless, good data efficiency is crucial for scaling RL to real world problems, like robotics (Kober, Bagnell, and Peters, 2013), and is a major motivation for the model-based approach.

4.6.2 *Exploration*

Exploration is a crucial topic in reinforcement learning. There are two main ways in which models and planning may benefit exploration: i) through two-phase exploration, and/or ii) through state-based exploration. We first introduce these two ideas:

- *One-phase versus two-phase exploration*: Model-free RL methods and pure planning methods use 'one-phase' exploration. One-phase exploration means that they use the same exploration principle in the entire algorithm, i.e., either within a trace (model-free RL) or within a tree (planning). The aim of one-phase exploration is to reduce real-world sample complexity (model-free RL) or reduce model sample complexity (planning).

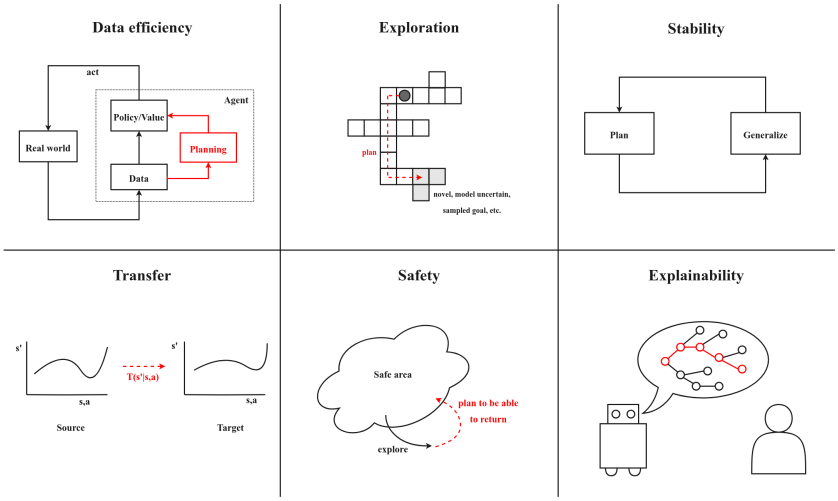


Figure 4.10: Benefits of model-based reinforcement learning, as discussed in Section 4.6.

In contrast, model-based RL agents use 'two-phase exploration', since they may combine 1) an exploration strategy within the planning cycle, and 2) a (usually more conservative) strategy for the irreversible (real environment) step. In the case of model-based RL with a learned model, the aim of this approach is usually to reduce real world sample complexity at the expense of increased model sample complexity. This has a close relation to the previous section (on data efficiency), although we there mostly focused on additional model-based back-ups, not exploration. In the case of model based RL with a known model we also observe two-phase exploration, like confidence bound methods inside the tree search and Dirichlet noise for the real steps in AlphaGo Zero (Silver et al., 2017c). However, with a known model (in which case planning and real steps happen in the same model) the second phase rather seems a pruning technique, to ensure that we terminate the planning cycle at some point and advance.

- *Value-based versus state-based exploration (intrinsic motivation)*: Most RL approaches use a form of 'value-based' exploration. Value-based methods base their exploration strategy on the current value estimates of the available actions. Actions with a higher value estimate will also get a higher probability of selection,

Table 4.5: Categories of exploration methods. Grey cells are considered 'model-based exploration', since they either use state-based characteristics and/or plan over the model to find better exploration decisions (two-phase exploration).

	One-phase exploration	**Two-phase exploration**
Value-based exploration	e.g., ϵ-greedy, value function uncertainty	e.g., planning to find a high value/reward region
State-based exploration	e.g., intrinsic reward for novelty without planning	e.g., planning towards an novel (goal) state

where the perturbation may for example be random (Mnih et al., 2015; Plappert et al., 2017) or based on uncertainty estimates around these values (Auer, 2002; Moerland, Broekens, and Jonker, 2017a; Osband et al., 2016). The model-based alternative is to use 'state-based' exploration. In this case, we do not determine the exploration potential of a state based on reward or value relevancy, but rather based on state-specific, reward independent properties derived from the interaction history with that state. A state may for example be interesting because it is novel or has high uncertainty in its model estimates. These approaches are better known as *intrinsic motivation* (IM) (Chentanez, Barto, and Singh, 2005).

The two above distinctions form four combinations, as visualized in Table 4.5. We define *model-based exploration* as 'any exploration approach that uses either state-based exploration and/or two-phase exploration' (indicated by the grey boxes in Table 4.5). Note that we consider all state-based exploration methods to be model-based RL. State-based exploration methods often use model-based characteristics or a density model over state space, which in the tabular case can directly be derived from a tabular model. We therefore include all state-based exploration as model-based RL, even when it is applied in one-phase (e.g, with model-free value approximation).

Literature on model-based RL is mostly structured along the distinction between knowledge-based and competence-based intrinsic motivation (Oudeyer, Kaplan, and Hafner, 2007; Oudeyer, Kaplan, et al., 2008). We will cover both fields, but we first discuss three underlying distinctions, one for the forward and two for the backwards phase, that are crucial to understand the challenge of exploration in general:

- *Shallow versus deep exploration*: Every exploration method can be classified as either shallow or deep. Shallow exploration methods redecide on their exploratory decision at every timestep. In the model-free RL context, ϵ-greedy exploration is a good example of this approach. The potential problem of these approaches is that they do not stick with an exploratory plan over multiple timestep. This may lead to 'jittering' behaviour, where we make an exploratory decision in a state, but decide to undo it at the next timestep. Intuitively, we want rather want to fix an interesting exploration target in the future, first use exploitation to get close to that new target, and only then start to explore (i.e., commit to a sequence of actions).

  Deep exploration (Osband et al., 2016) methods aim to correlate exploration decision over multiple timesteps (note that 'deep' in this case has nothing to do with the depth of a network). In the model-free RL setting, we may try to achieve deeper exploration through, for example, parameter space noise over episodes (Plappert et al., 2017) or through propagation of value uncertainty estimates (Moerland, Broekens, and Jonker, 2017a; Osband et al., 2016). However, deep exploration is very natural to model-based RL, since the planning cycle can perform a deeper lookahead, to which we can then commit in the real environment (Lowrey et al., 2018; Sekar et al., 2020). Note that for model-based exploration there is one caveat: when we plan for a deep sequence, but then only execute the first action of the sequence and replanning (a receding-horizon), then we still have the risk of jittering behaviour.

- *Task-conflated versus task-separated exploration back-ups*: Once we identify an interesting new state (e.g., because it is novel), we want to back-up this information to possibly return there in a next episode. Therefore, back-ups are a crucial element of the exploration cycle. Many intrinsic motivation approaches use intrinsic rewards (Chentanez, Barto, and Singh, 2005) (e.g., for novelty), and simply add these to the extrinsic reward. The exploration signal is then propagated inside the global value/policy function, combined with the true task information. A downside of task-conflated propagation is that it modifies the global solution, since it conflates task relevance with exploration relevancy. Therefore, after an intrinsic reward has worn out, it may take time to fade out its effect on the value function.

As an alternative, we may also use *task-separated* exploration back-ups. In these cases, the global solution (value or policy function) is explicitly separated from the exploration information, like the way to get back to a particular interesting region. For example, Shyam, Jaśkowski, and Gomez (2019) propose to store separate value functions for the intrinsic and extrinsic rewards. As an alternative, we may also store the exact trace towards particular goals, which has a relation to episodic memory (Pritzel et al., 2017). For example, Ecoffet et al. (2019) assume that we can exactly reset an agent to any state we previously reached. Alternatively, we may also separately learn the policy parameters that reached a particular state (Laversanne-Finot, Pere, and Oudeyer, 2018). Task-separated back-ups introduce more complexity for learning, but also seem a more principled approach to separate exploration from exploitation.

- *Shallow versus deep exploration back-ups*: Similar to shallow and deep exploration (in the forward sense), the depth of the back-up is also important for exploration. As well-known from classic RL theory, a back-up can be *shallow* (e.g., a one-step target) or *deep* (e.g., a Monte Carlo target or n-step method) (Sutton and Barto, 2018). When we search for the optimal solution, one-step back-ups can be off-policy and therefore have the benefit of converging to the optimal solution. Therefore, shallow back-ups clearly have their benefit for convergence. However, from an exploration perspective, they have a potential drawback. Imagine we just encountered an interesting novel state, but we only back-up this information for one-step towards the previous state. In the next episode, we will not be able to see the information near the start location (it has not been propagated far enough). Ecoffet et al. (2019) call this the 'detachment' problem for exploration. The agent then has to stumble around again until it finds a new intrinsic reward, or a state to which the previous intrinsic reward was backed-up.

  The detachment problem can of course be partially mitigated through experience replay (Lin and Mitchell, 1992), or even better, prioritized sweeping (Moore and Atkeson, 1993). However, we can also use a deep back-up, like a Monte Carlo target for value-based propagation. When we combine this with a value-separate back-up (see above), then the Monte Carlo target only affects the exploration information, not the global solution. We may then not store the quickest route back to the novel state, but we do not care

about optimality yet, we just want to get back there. Alternatively, we can also store the entire trace or policy parameters needed to reach a particular interesting state or region, which is by definition deep. In any case, deep back-ups may strongly accelerate our ability to build on novel discoveries.

With these concepts in mind, we will now discuss model-based exploration. We will follow the intrinsic motivation literature (i.e., state-based exploration) (Chentanez, Barto, and Singh, 2005), which is traditionally split up in two sub-fields (Oudeyer, Kaplan, et al., 2008): 1) *knowledge-based* intrinsic motivation, and 2) *competence-based* intrinsic motivation.

KNOWLEDGE-BASED INTRINSIC MOTIVATION Knowledge-based IM prioritizes those state for exploration where we may acquire new information about the MDP. This approach is generally combined with intrinsic rewards and task-conflated propagation. We specify an intrinsic reward function $r^i(s)$ or $r^i(s,a,s')$, which estimates the saliency of a particular state or state transition, and let the agent optimize a combination of extrinsic and intrinsic reward:

$$r_t(s,a,s') = r^e(s,a,s') + \eta \cdot r^i(s,a,s'), \tag{4.2}$$

where r^e denotes the external reward, and $\eta \in \mathbb{R}$ is a hyperparameter that controls the relative strength of the intrinsic motivation.

There are various ways to specify r^i. By far the largest category uses the concept of *novelty* (Bellemare et al., 2016; Hester and Stone, 2012a; Sequeira, Melo, and Paiva, 2014). For example, the Bayesian Exploration Bonus (BEB) (Kolter and Ng, 2009) uses

$$r^i(s,a,s') \propto 1/(1+n(s,a)), \tag{4.3}$$

where $n(s,a)$ denotes the number of visits to state-action pair (s,a). Novelty ideas were recently studied in high-dimensional problems as well, using the concept of pseudo-counts, which closely mimick density estimates (Bellemare et al., 2016; Ostrovski et al., 2017).

There are various other ways to specify the intrinsic reward signal. Long before the term knowledge-based IM became established, Sutton (1990) already included an intrinsic reward for *recency*:

$$r^i(s,a,s') = \sqrt{l(s,a)}, \tag{4.4}$$

where $l(s,a)$ denotes the number of timesteps since the last trial at (s,a). More recent examples of intrinsic rewards include *model prediction error* (Pathak et al., 2017; Stadie, Levine, and Abbeel, 2015), *surprise* (Achiam and Sastry, 2017), *information gain* (Houthooft et al., 2016), and *feature control* (the ability to change elements of our state over time) (Dilokthanakul et al., 2019). Note that intrinsic rewards for recency and model prediction error may help overcome non-stationarity (Sec. 4.3.5) as well (Lopes et al., 2012). Multiple intrinsic rewards can also be combined, like a combination of novelty and model uncertainty (Hester and Stone, 2012a). Note that many of these intrinsic motivation ideas originate in emotion theory, which was surveyed for RL agents by Moerland, Broekens, and Jonker (2018a).

Novelty is a very common concept in exploration research, also outside the intrinsic motivation framework. Another model-based approach to exploration is the *probably approximately correct (PAC)-MDP* framework (Kakade et al., 2003), of which R-Max (Brafman and Tennenholtz, 2002) is an example. R-Max assumes every transition has maximal reward until it has at least been visited a certain number of times. We consider this a model-based approach, since it needs to keep a count-based model over all transitions, although R-Max does not use planning (it is a one-phase exploration method).

Many of the above knowledge-based IM methods are implemented in a one-phase way, i.e., the intrinsic reward is computed when encountered, but there is not explicit planning towards it. We can of course also combine knowledge-based IM with two-phase exploration (Sekar et al., 2020), i.e. 'plan to explore'. As mentioned before, nearly all knowledge-based IM approaches use task-conflated propagation, while Shyam, Jaśkowski, and Gomez (2019) do learn separate value functions for the intrinsic and extrinsic rewards. The back-up depth of knowledge-based IM varies from shallow to deep, depending on the type of value back-up.

COMPETENCE-BASED INTRINSIC MOTIVATION Competence-based intrinsic motivation builds on the same curiosity principles as knowledge-based IM. However, competence-based IM selects new exploration targets based on *learning progress*, which focuses on the competence of the agent, rather than the knowledge about the MDP. The goal is to generate a *curriculum* of tasks for learning progress (Bengio et al., 2009). A formalization of these ideas are Intrinsically Motivated Goal Exploration Processes (IMGEP) (Baranes and Oudeyer, 2009). They consist of

three steps: 1) learn a goal space, 2) sample a goal, and 3) plan towards the goal.

Goal space learning was already discussed in Sec. 4.3.7 and 4.3.8. The general aim is to capture the salient directions of variation in a task in a representation. For competence-based IM, it may be useful to learn a *disentangled* representation, where each controllable object is captured by a separate dimension in the representation. Then, we can create a better curriculum by sampling new subgoals that alter only one controllable object at a time (Laversanne-Finot, Pere, and Oudeyer, 2018).

The second step, goal space sampling, is a crucial part of competence-based IM. We aim to select a goal that has high potential for *learning progress* (Baranes and Oudeyer, 2013; Oudeyer, Kaplan, and Hafner, 2007). One approach is to track a set of goals, and reselect those goals for which the achieved return has shown positive change recently (Laversanne-Finot, Pere, and Oudeyer, 2018; Matiisen et al., 2017). As an alternative, we may also fit a generative model to sample new goals from, which may for example be trained on all previous goals (Péré et al., 2018) or on a subset of goals of intermediate difficulty (Florensa et al., 2018). Note that the concept of learning progress has also appeared in knowledge-based IM literature (Schmidhuber, 1991b).

In the third step, we actually attempt to reach the sampled goal. The key idea is that we should already know how to get close to the new goal, since we sampled it close to a previously reached state. Goal-conditioned value functions (discussed in Sec. 4.3.8) can be one way to achieve this, but we may also attempt to learn a mapping from current state and goal to policy parameters (Laversanne-Finot, Pere, and Oudeyer, 2018). These latter approach attempts to generalize in policy parameter space, and can be considered near episodic propagation approaches.

Competence-based IM often uses deep exploration and task-separated, deep back-ups. However, the true difference between knowledge and competence-based approaches is the type of information that makes a state/goal salient (knowledge, e.g. novelty, or competence, i.e., learning progress). All other distinctions mentioned in the beginning of this section are applicable to both. The vanilla approach in each category is illustrated in Figure 4.11.

Finally, this section on model-based exploration has not discussed any hierarchical RL methods, since these were already covered in Sec. 4.3.8. Hierarchy can be used in a model-free or model-based way. In both

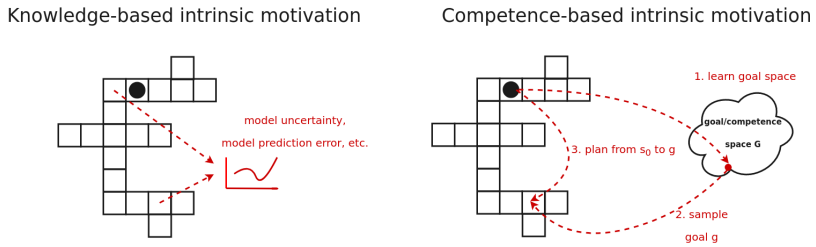


Figure 4.11: Knowledge-based versus competence-based intrinsic motivation. Solid circle identifies the current agent position. **Left**: In knowledge-based intrinsic motivation, every state (the arrows show two examples) in the domain gets associated with an intrinsic reward based on local characteristics, like visitation frequency, uncertainty of the model, prediction error of the model, etc. **Right**: In competence-based intrinsic motivation, we learn some form of a goal-space that captures (and compresses) the directions of variation in the domain. We then sample a new goal, for example at the edge of our current knowledge base, and explicitly try to reach it, re-using the way we previously got close to that state.

cases, good higher level actions can strongly reduce the exploration complexity, like the depth of a tree during planning. As discussed in Sec. 4.3.8, good end-points for hierarchical actions can for example be obtained from global coverage of state space, which reminds of the goal spaces used in competence-based intrinsic motivation. In any cases, hierarchy will like be a crucial component of (model-based) exploration as well.

4.6.3 *Stability*

Another benefit of model-based RL, in the context of a known model, seems better asymptotic performance. For model-based RL with a learned model, the common knowledge is that we may improve data efficiency, but loose asymptotic performance in the long run. However, recent attempts of model-based RL with a known model, like AlphaGo Zero (Silver et al., 2017c) and Guided Policy Search (Levine and Koltun, 2013), manage to outperform model-free attempts on long-run empirical performance. This suggests that with a perfect (or good) model, model-free RL may actually lead to better (empirical) asymptotic performance. Moreover, MuZero (Schrittwieser et al., 2019) uses a (value-equivalent)

learned model and actually outperforms the asymptotic performance of AlphaGo Zero.

A possible explanation for the mutual benefit of planning and learning originates from the type of representation they use. The atomic (tabular) representation of planning does not scale to large problems, since the table would grow too large. The global approximation of learning provides the necessary generalization, but will inevitably make local approximation errors. However, when we add local planning to learning, the local representation may help to locally smooth out the errors in the function approximation, by looking ahead to states with more clearly discriminable value predictions. These local representations are often tabular/exact, and can thereby give better local separation. For example, in Chess the learned value prediction for the current state of the board might be off, but through explicit lookahead we may find states that are a clear win or loss in a few steps. As such, local planning may help learning algorithms to locally smooth out the errors in its approximation, leading to better asymptotic performance.

There is some initial work that supports these ideas. Silver, Sutton, and Müller (2008) already described the use of *transient* and *permanent* memory, where the transient memory is the local plan that fine-tunes the value estimates. Both Moerland et al. (2020) and Wang et al. (2019) recently studied the trade-off between planning and learning (already mentioned in Sec. 4.4.2), finding that optimal performance requires an intermediate planning budget per real step, and not a very high budget (exhaustive search), or no planning budget per timestep at all (model-free RL). Since model-free RL is notoriously unstable in the context of function approximation (Henderson et al., 2018), we may hypothesize that the combination of global function approximation (learning) and local atomic/tabular representation (planning) helps stabilize learning and achieve better asymptotic performance (see Hamrick et al. (2020) as well).

To conclude, we note that this combination of local planning and global approximation also exists in humans. In cognitive science, this idea is known as dual process theory (Evans, 1984), which was more recently popularized as 'thinking fast and slow' (Kahneman, 2011). Anthony, Tian, and Barber (2017) connect planning-learning integration to these ideas, suggesting that global policy or value functions are like 'thinking fast', while local planning relates to explicit reasoning and 'thinking slow'.

4.6.4 *Transfer*

In *transfer learning* (Lazaric, 2012; Taylor and Stone, 2009) we re-use information from a source task to speed-up learning on a new task. The source and target tasks should neither be the same, as then transfer is trivial, nor completely unrelated, as then there is no information to transfer. Konidaris (2006) covers a framework for transfer, specifying three types: i) transfer of a dynamics model, ii) transfer of skills or sub-routines, and iii) transfer of 'knowledge', like shaping rewards and representations. For this model-based RL survey we only discuss the first category, transfer of a dynamics model. There are largely two scenarios: i) similar dynamics function but different reward function, for example a new level in a video game, and ii) slightly changed transition dynamics, for example transfer from simulation to real-world tasks. We discuss examples in both categories.

SAME DYNAMICS WITH DIFFERENT REWARD The first description of model transfer with a changed reward function is by Atkeson and Santamaria (1997). The authors change the reward function in a Pendulum swing-up task after 100 trials, and show that the model-based approach is able to adapt much faster, requiring less data from the real environment. Later on, the problem (different reward function with stationary dynamics) became better known as *multi-objective* reinforcement learning (MORL) (Roijers et al., 2013; Roijers and Whiteson, 2017). A multi-objective MDP has a single dynamics function but multiple reward functions. These rewards can be combined in different ways, each of which lead to a new task specification. There are many model-free approaches for the MORL setting (Roijers et al., 2013), with model-based examples given by Wiering, Withagen, and Drugan (2014), Yamaguchi et al. (2019). Other examples of model-based transfer to different reward functions (goals) are provided by Sharma et al. (2019) and Sekar et al. (2020).

Another approach designed for changing reward functions is the successor representation (Barreto et al., 2017; Dayan, 1993). Successor representations summarize the model in the form of future state occupancy statistics. It thereby falls somewhere in between model-free and model-based methods (Momennejad et al., 2017), since these methods can partially adapt to a different reward function, but it does not fully compute new occupancy statistics like a full model-based method would.

DIFFERENT DYNAMICS In the second category we find transfer to a task with slightly different dynamics. Conceptually, Konidaris and Barto (2007) propose to disentangle the state into an agent space (which can directly transfer) and a problem space (which defines the new task). However, disentanglement of agent and problem space is still hard without prior knowledge.

One way to achieve good transfer is by learning representations that generalize well. The object-oriented and physics-based approaches, already introduced in Sec. 4.3.7, have shown success in achieving this. For example, Schema Networks (Kansky et al., 2017) learn object interactions in Atari games, and manage to generalize well to several variations of Atari Breakout, like adding a new wall or slightly changing the dynamics (while still complying with the overall physics rules).

Simulation-to-real transfer is popular in robotics, but most researchers transfer a policy or value function (Tobin et al., 2017). Example approaches that do transfer a dynamics model to the real world are Christiano et al. (2016) and Nagabandi et al. (2018a). Several researchers also take a zoomed out view, where they attempt to learn a distribution over the task space, better known as *multi-task learning* (Caruana, 1997). Then, when a new task comes in, we may quickly identify in which cluster of known tasks (dynamics models) it belongs (Wilson et al., 2007). Another approach is to learn a global neural network initialization that can quickly adapt to new tasks sampled from the task space (Clavera et al., 2018), which implicitly transfers knowledge about the dynamics of related tasks.

In short, transfer is one of the main benefits of model-based RL. Van Seijen et al. (2020) even propose a metric, the Local Change Adaptation (LoCA) regret, to compare model-based RL algorithms based on their ability to learn on new, slightly altered tasks. An overview of transfer methods for deep reinforcement learning in general is provided by Zhu, Lin, and Zhou (2020).

4.6.5 *Safety*

Safety is an important issue, especially when learning on real-world systems (Amodei et al., 2016). For example, with random exploration it is easy to break a robot before any learning has taken place. Berkenkamp et al. (2017) studies a model-based safe exploration approach based on the notion of asymptotic stability. Given a 'safe region' of the current policy, we want to explore while ensuring that we can always get back

to the safe region. As an alternative, Aswani et al. (2013) keep two models: the first one is used to decide on an exploration policy, while the second model has uncertainty bounds and is used for verification of the safety of the proposed policy. Ostafew, Schoellig, and Barfoot (2016) ensure constraints by propagating uncertainty information in a Gaussian Process model. Safety is a vital aspect of real-world learning, and it may well become an important motivation for model-based RL in forthcoming years.

4.6.6 *Explainability*

Explainable artificial intelligence (XAI) has received much attention in the AI community in recent years. Explainable reinforcement learning (XRL) was studied by Waa et al. (2018), who generated explanations from planned traces. The authors also study contrastive explanations, where the user can ask the agent why it did not follow another policy. There is also work on RL agent transparency based on emotion elicitation during learning (Moerland, Broekens, and Jonker, 2018a), which largely builds on model-based methods. Finally, Shu, Xiong, and Socher (2017) study language grounding in reinforcement learning, which is an important step to explainability as well. Explainability is now widely regarded as a crucial prerequisite for AI to enter society. Model-based RL may be an important element of explainability, since it allows the agent to communicate not only its goals, but also the way it intends to achieve them.

4.6.7 *Disbenefits*

Model-based RL has disbenefits as well. First, model-based RL typically requires additional computation, both for training the model, and for the planning operations themselves. Second, model-based RL methods with a learned model can be very unstable due to uncertainty and approximation errors in the model. Therefore, although these approaches can be more data efficient, they also tend to have lower asymptotic performance. We already extensively discussed how to deal with model uncertainty. Third, model-based RL methods require additional memory, for example to store the model. However, with function approximation this is typically not a large limitation. Finally, model-based RL algorithms typically have more tunable hyperparameters than model-free algorithms, including hyperparameters to estimate uncertainty, and

hyperparameters to balance planning and real data collection. Most of these disbenefits are inevitable, and we are essentially trading extra computation, memory and potential instability (for a learned model) against better data efficiency, targeted exploration, transfer, safety and explainability.

4.7 RELATED WORK

While model-based RL has been very successful and received much attention (Deisenroth and Rasmussen, 2011; Levine and Koltun, 2013; Silver et al., 2017c), a survey of the field currently lacks in literature. Hester and Stone (2012b) gives a book-chapter presentation of model-based RL methods, but their work does not provide a full overview, nor does it incorporate the vast recent literature on neural network approximation in model-based reinforcement learning.

Chapter 3 presented a framework for reinforcement learning and planning that disentangles their common underlying dimensions, but does not focus on their integration. In some sense, Chapter 3 looks 'inside' each planning or reinforcement learning cycle, strapping their shared algorithmic space down into its underlying dimensions. Instead, our work looks 'over' the planning cycle, focusing on how we may integrate planning, learning and acting to provide mutual benefit.

Hamrick (2019) presents a recent coverage of mental simulation (planning) in deep learning. While technically a model-based RL survey, the focus of Hamrick (2019) lies with the relation of these approaches to cognitive science. Our survey is more extensive on the model learning and integration side, presenting a broader categorization and more literature. Nevertheless, the survey by Hamrick (2019) is an interesting companion to the present work, for deeper insight from the cognitive science perspective. Plaat, Kosters, and Preuss (2020) also provide a recent description of model-based RL in high-dimensional state spaces, and puts additional emphasis on implicit and end-to-end model-based RL (see Sec. 4.5 as well).

Finally, several authors (Nguyen-Tuong and Peters, 2011; Polydoros and Nalpantidis, 2017; Sigaud, Salaün, and Padois, 2011) have specifically surveyed structured model estimation in robotics and control tasks. In these cases, the models are structured according to the known laws of physics, and we want to estimate a number of free parameters in these models from data. This is conceptually similar to Sec. 4.3, but our work discusses the broader supervised learning literature, when

applicable to dynamics model learning. Thereby, the methods we discuss do not need any prior physics knowledge, and can deal with much larger problems. Moreover, we also include discussion of a variety of other model learning challenges, like state and temporal abstraction.

4.8 DISCUSSION

This chapter surveyed the full spectrum of model-based RL, including model learning, planning-learning integration, and the benefits of model-based RL. To further advance the field, we need to discuss two main topics: benchmarking, and future research directions.

BENCHMARKING Benchmarking is crucial to the advancement of a field. For example, major breakthroughs in the computer vision community followed the yearly ImageNet competition (Krizhevsky, Sutskever, and Hinton, 2012). We should aim for a similar benchmarking approach in RL, and in model-based RL in particular.

A first aspect of benchmarking is proper assessment of problem difficulty. Classic measures involve the breadth and depth of the full search tree, or the dimensionality of the state and action spaces. While state dimensionality was for long the major challenge, breakthroughs in deep RL are now partially overcoming this problem. Therefore, it is important that we start to realize that state and action space dimensionality are not the only relevant measures of problem difficulty. For example, sparse reward tasks can be very challenging for exploration, even in low dimensions. Osband et al. (2019) recently proposed a benchmarking suite that disentangles the ability of an algorithm to deal with different types of challenges.

A second part of benchmarking is actually running and comparing algorithms. Although many benchmarking environments for RL have been published in recent years (Bellemare et al., 2013; Brockman et al., 2016), and benchmarking of model-free RL has become quite popular, there is relatively little work on benchmarking model-based RL algorithms. Wang et al. (2019) recently made an important first step in this direction by benchmarking several model-based RL algorithms, and the field would profit from more efforts like these.

For reporting results, an important remaining challenge for the entire RL community is standardization of learning curves and results. The horizontal axis of a learning curve would ideally show the number of unique flops (computational complexity) or the number of real world

or model samples. However, many papers report 'training time in hours/days' on the horizontal axis, which is of course heavily hardware dependent. Other papers report 'episodes' on the horizontal axis, while a model-based RL algorithm uses much more samples than a model-free algorithm per episode. When comparing algorithms, we should always aim to keep either the total computational budget or the total sample budget equal.

FUTURE WORK There is a plethora of future work directions in the intersection of planning and learning. We will mention a few research areas, which already received much attention, but have the potential to generate breakthroughs in the field.

- Asymptotic performance: Model-based RL with a learned model tends to have better sample complexity, but inferior asymptotic performance, compared to model-free RL. This is an important limitation. AlphaGo Zero recently illustrated that model-based RL with a known model should be able to surpass model-free RL performance. However, in the context of a learned model, a major challenge is to achieve the same optimal asymptotic performance as model free RL, which probably requires better ways of estimating and dealing with model uncertainty.
- Hierarchy: A central challenge, which has already received much attention, is temporal abstraction (hierarchical RL). We still lack consistent methods to identify useful sub-routines, which compress, respect reward relevancy, identify bottleneck states and/or focus on interaction with objects and salient domain aspects. The availability of good temporal abstraction can strongly reduce the depth of a tree search, and is likely a key aspect of model-based learning.
- Exploration & Competence-based intrinsic motivation: A promising direction within exploration research could be competence-based intrinsic motivation (Oudeyer, Kaplan, and Hafner, 2007), which has received less attention than its brother knowledge-based intrinsic motivation (see Sec. 4.6.2). By sampling goals close to the border of our currently known set, we generate an automated curriculum, which may make exploration more structured and targeted.
- Transfer: We believe model-based RL could also put more emphasis on the transfer setting, especially when it comes to evaluating

data efficiency. It can be very hard to squeeze out all information on a single, completely new task. Humans mostly use forward planning on reasonably certain models that generalize well from previous tasks. Shifting RL and machine learning from single task optimization to more general artificial intelligence, operating on a variety of tasks, is an important challenge, in which model-based RL may definitely play an important role.

- Balancing: Another important future question in model-based RL is balancing planning, learning and real data collection. These trade-offs are typically tuned as hyperparameters, which seem to be crucial for algorithm performance (Moerland et al., 2020; Wang et al., 2019). Humans naturally decide when to start planning, and for how long (Kahneman, 2011). Likely, the trade-off between planning and learning should be a function of the collected data, instead of a fixed hyperparameter.

- Prioritized sweeping: Prioritized sweeping has been very successful in tabular settings, when the model is trivial to revert. As mentioned throughout the survey, it has also been applied to high-dimensional approximate settings, but this creates a much larger challenge. Nevertheless, exploration in the forward direction may actually be just as important as propagation in the backwards direction, and prioritized sweeping in high-dimensional problems is definitely a topic that deserves attention.

- Optimization: Finally, note that RL is effectively an optimization problem. While this survey has focused on the structural aspects of this challenge (what models to specify, how to algorithmically combine them, etc.), we also observe much progress in combining optimization methods, like gradient descent, evolutionary algorithms, automatic hyperparameter optimization, etc. Such research may have an equally big impact on progress in MDP optimization and sequential decision making.

4.9 SUMMARY

This concludes our survey of model-based reinforcement learning. We will briefly summarize the key points:

- Nomenclature in model-based RL is somewhat vague. We define model-based RL as 'any MDP approach that uses i) a model

(known or learned) and ii) learning to approximate a global value or policy function'. We distinguish three categories of planning-learning integration: 'model-based RL with a learned model', 'model-based RL with a known model', and 'planning over a learned model' (Table 4.1).

- Model-based reinforcement learning may first require approximation of the dynamics model. Key challenges of model learning include dealing with: environment stochasticity, uncertainty due to limited data, partial observability, non-stationarity, multi-step prediction, and representation learning methods for state and temporal abstraction (Sec. 4.3).
- Integration of planning and learning involves a few key aspects: i) where to start planning, ii) how much budget to allocate to planning and acting, iii) how to plan, and iv) how to integrate the plan in the overall learning and acting loop. Planning-learning methods widely vary in their approach to these questions (Sec. 4.4).
- Explicit model-based RL manually designs model learning, planning algorithms and the integration of these. In contrast, implicit model-based RL optimizes elements of this process, or the entire model-based RL computation, against the ability to predict an outer objective, like a value or optimal action (Sec. 4.5).
- Model-based RL can have various benefits, including aspects like data efficiency, targeted exploration, transfer, safety and explainability (Sec. 4.6). Recent evidence indicates that the combination of planning and learning may also provide more stable learning, possibly due to the mutual benefit of global function approximation and local tabular representation.

In short, both planning and learning are large research fields in MDP optimization that depart from a crucially different assumption: the type of access to the environment. Cross-breeding of both fields has been studied for many decades, but a systematic categorization of the approaches and challenges to model learning and planning-learning integration lacked so far. Recent examples of model-based RL with a known model (Levine and Koltun, 2013; Silver et al., 2017c) have shown impressive results, and suggest much potential for future planning-learning integrations. This survey conceptualized the advancements in

model-based RL, thereby: 1) providing a common language to discuss model-based RL algorithms, 2) structuring literature for readers that want to catch up on a certain subtopic, for example for readers from either a pure planning or pure RL background, and 3) pointing to future research directions in planning-learning integration.

Part III

EXPERIMENTAL INTEGRATION OF PLANNING AND LEARNING

5

STOCHASTIC DYNAMICS APPROXIMATION WITH CONDITIONAL VARIATIONAL INFERENCE [1]

ABSTRACT

In this chapter we study how to learn stochastic, multimodal transition dynamics in reinforcement learning (RL) tasks. We focus on evaluating transition function estimation, while we defer planning over this model to future work. Stochasticity is a fundamental property of many task environments. However, discriminative function approximators have difficulty estimating multimodal stochasticity. In contrast, deep generative models do capture complex high-dimensional outcome distributions. First we discuss why, amongst such models, conditional variational inference (VI) is theoretically most appealing for model-based RL. Subsequently, we compare different VI models on their ability to learn complex stochasticity on simulated functions, as well as on a typical RL gridworld with multimodal dynamics. Results show VI successfully predicts multimodal outcomes, but also robustly ignores these for deterministic parts of the transition dynamics. In summary, we show a robust method to learn multimodal transitions using function approximation, which is a key preliminary for model-based RL in stochastic domains.

5.1 INTRODUCTION

Dynamics model learning is a crucial step in model-based reinforcement learning, as extensively discussed in Chapter 4. A particular challenge of model learning is the ability to deal with stochasticity, which we introduced in (Sec. 4.3.2. In this chapter, we will experimentally investigate a new approach to this challenge.

Stochasticity is an inherent property of many environments, and increases in real-world settings due to sensor noise. Transition dynamics usually combine both deterministic aspects (such as the falling trajectory of an object due to gravity) and stochastic elements (such as the behaviour of another car on the road). Our goal is to learn to jointly predict these. Note that stochasticity has many forms, both homoscedas-

1 Chapter based on: Moerland TM, Broekens J, Jonker CM. Learning Multimodal Transition Dynamics for Model-Based Reinforcement Learning. Scaling Up Reinforcement Learning (SURL) Workshop @ European Conference on Machine Learning (ECML), 2017.

tic versus heteroscedastic, and unimodal versus multimodal. In this work we specifically focus on multimodal stochasticity, as this should theoretically pose the largest challenge.

To learn such transition models, we require high-capacity function approximators that can predict next-state distributions of complex shape. This problem is not yet accurately solved by currently used methods in model-based RL, like tabular learning (Brafman and Tennenholtz, 2002) (which does not scale to high-dimensions), linear function approximation (Atkeson, Moore, and Schaal, 1997) with Gaussian noise (Li et al., 2011), random forests (Hester and Stone, 2012b), or deep feed-forward networks trained on mean-squared error (MSE) (Oh et al., 2015) (See also Sec. 5.2).

A potential solution to this problem are Deep Generative Models (DGM) (Goodfellow, 2016), as they are non-linear function approximators that can learn complex outcome distributions and scale to high-dimensions. In Section 5.2 we compare different DGM's on their theoretic appeal for stochastic model learning in the RL setting, and identify conditional Variational Inference (VI) as the most promising solution.

The remainder of the chapter then continues as follows. In Section 3, we formally describe conditional variational inference with different types of discrete and continuous latent variables. In Section 4 we empirically compare the different approaches on a simulated function and on a typical RL tasks. Our results show that VI is accurately able to discriminate deterministic from stochastic aspects of the transition dynamics. We also show how the RL agent manages to learn an accurate transition model while solving a task. Finally, Sections 5 and 6 connect our work to related literature and identify opportunities for future work, respectively. Code to replicate experiments is available from `www.github.com/tmoer/multimodal_varinf`.

5.2 CHALLENGE OF MULTIMODAL TRANSITIONS

We will write $x \in \mathcal{X}$ for the current state and action, and $y \in \mathcal{Y}$ for the next state we want to predict. We are interested in models that can approximate distributions $p(y|x)$ with multiple local maxima ('modes'). A cardinal example of such a distribution is shown in Figure 5.1 (blue line).

Discriminative function approximators, for example trained at mean-squared error (MSE) loss, fail at this task. The point prediction of

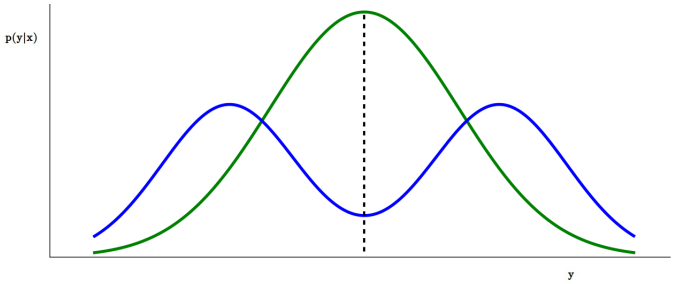


Figure 5.1: Multimodal outcome distribution $p(y|x)$ (blue line) for a simple 1D observation space. Training on a mean-squared error will deterministically predict the conditional mean (dashed line), which implicitly assumes a uni-modal Gaussian outcome (green line).

the function approximator will be the conditional expectation of the outcome distribution (Fig. 5.1, dashed line). Obviously, point estimate predictions will never be a good method to approximate a distribution, but they have actually been frequently applied in model-based RL work (Oh et al., 2015).[2] Clearly, a unimodal outcome distribution will neither solve the multimodality problem (Fig. 5.1).

A mixture of Gaussians per outcome dimension would neither solve the problem. Full covariance matrix Gaussians clearly do not scale to high-dimensional domains (such as (Mnih et al., 2015)), while diagonal Gaussians would loose all covariance structure in the predictions. Moreover, mixture models are tedious to train. What we require are models that 1) flexibly approximate joint distributions of complex (multimodal) shape and 2) scale to high-dimensions.

We hypothesize the group of deep generative models (DGN) (Goodfellow, 2016) are a promising candidate, as they fulfill both requirements. For model-based RL, where we will use the models to sample (a lot of) traces, we additionally require that the model is 1) easy to sample from, and 2) ideally allows for planning at an abstract level. Following the DGN taxonomy by Goodfellow (Goodfellow, 2016) (Figure 5.2), we

2 For example, Oh et al. (2015) shows MSE training does work well in high-dimensional, deterministic domains. However, for example inspecting their Ms. Pacman (a stochastic game) predictions at https://youtu.be/cy96rtUdBuE, we see that the predictions for the stochastic elements (ghosts) fail. The ghosts disappear when they reach a corridor junction, where they stochastically choose in which direction to continue. The feed-forward network predicts the conditional mean of these choices, which completely blurs the ghosts in a few frames.

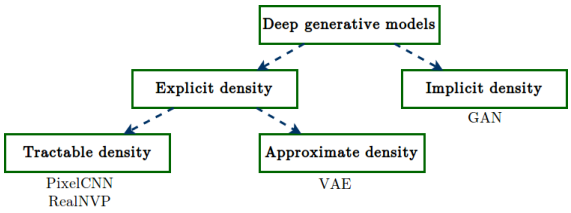


Figure 5.2: Deep generative model taxonomy following (Goodfellow, 2016).

now compare DGN models on their theoretical appeal for transition function estimation.

Implicit density models, like Generative Adverserial Networks (GAN) lack a clear probabilistic objective function, which was the focus of this work. Among the explicit density models, there are two categories. *Change of variable formula* models, like Real NVP (Dinh, Sohl-Dickstein, and Bengio, 2016), have the drawback that the latent space dimension must equal the observation space. Fully visible belief nets like pixelCNN (Oord et al., 2016), which factorize the likelihood in an auto-regressive fashion, hold state-of-the-art likelihood results. However, they have the drawback that sampling is a sequential operation (e.g. pixel-by-pixel, which is computationally expensive), and they do not allow for latent level planning either. Therefore, most suitable for model-based RL seem approximate density models, most noteworthy the variational auto-encoder (VAE) (Kingma and Welling, 2014) framework. These models can estimate stochasticity at a latent level, allow for latent planning (Watter et al., 2015), are easy to sample from, and have a clear probabilistic interpretation. In the next section, we will formally introduce this methodology in the conditional setting, where the generative process of y is conditioned on other variables x.

5.3 CONDITIONAL VARIATIONAL INFERENCE

We will first introduce the conditional variational auto-encoder (CVAE) (Sohn, Lee, and Yan, 2015). Our goal is to learn a generative model of a (possibly multimodal) distribution $p(y|x)$. We assume the generative process is actually conditioned on latent variables z:

$$p(y|x) = \int p(y|z,x)p(z|x)dz \tag{5.1}$$

Here $p(z|x)$ is the *prior* and $p(y|z,x)$ is the *generative model* or 'decoder'. The stochastic latent variables z provide the flexibility to predict

more complex outcome distributions y. The posterior over z, $p(z|y,x)$ is intractable in most models of interest, for example deep non-linear neural networks. However, the parameters of this distribution can be efficiently approximated with Stochastic Gradient Variation Bayes (SGVB) (Kingma and Welling, 2014), which uses a parametric recognition or *inference model* $q(z|y,x)$ to approximate the true posterior $p(z|y,x)$. The inference model learns a mapping from observations to latent space, providing generalization and thereby amortizing the cost of inference (compared to Markov chain Monte Carlo (MCMC) inference methods that needed computationally expensive iterative procedures to estimate the latent variables per datapoint).

We can derive a variational lower bound $\mathcal{L}(y|x)$ on our data likelihood $p(y|x)$:

$$\begin{aligned}\log p(y|x) &\geq \mathbb{E}_{z\sim q(z|x,y)}\left[\log \frac{p(y,z|x)}{q(z|y,x)}\right] \\ &= \mathbb{E}_{z\sim q(z|x,y)}[\log p(y|z,x)] - D_{\mathrm{KL}}[q(z|x,y)\|p(z|x)] \\ &= \mathcal{L}(y|x;\theta,\phi) \end{aligned} \tag{5.2}$$

where θ denotes the parameters in the generative network, ϕ denotes the parameters in the inference network and prior, and D_{KL} denotes the Kullback-Leibler (KL) divergence. We can interpret the left-hand term of the last equation ($\log p(y|z,x)$) as the negative *reconstruction error*, which measures how well we reconstruct y after sampling z. The right-hand term (KL divergence) ensures q does not diverge too much from the prior p. This acts as a regularizer, and ensures that we can at test time (when we do not observe y) sample from $p(z|x)$ instead of $q(z|x,y)$.

In practice, we slightly modify the objective in Eq. 5.2, where we use importance sampling (Burda, Grosse, and Salakhutdinov, 2015) to obtain a tighter bound, and minimize a different distance function instead of the KL-divergence (namely α-divergence with α=0.5 (Depeweg et al., 2016)). Details are provided in Appendix 5.8.1.

5.3.1 *Reparametrization*

For this work we focus on variational methods that *reparametrize* the distribution of $q_\phi(z|y,x)$ to allow gradient-based training on a single computational graph. The trick works when we can write z as a function $z = f_\phi(\epsilon, y, x)$, with $f_\phi(\cdot)$ a deterministic, differentiable function, and $\epsilon \sim p(\epsilon)$ a noise distribution with independent marginal.

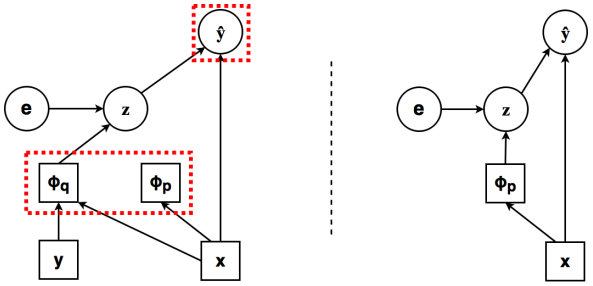


Figure 5.3: Conditional Variational Auto-Encoder as a computational process. Squares are deterministic, circles are probabilistic nodes. **Left**: Training procedure. During training, we sample z according to $q(\cdot|x,y)$, where q is parametrized by ϕ_q. The training loss consists of two terms (indicated by the red dotted boxes): 1) the reconstruction loss $p(y|z,x)$, and 2) the KL-divergence between $q(z|x,y)$ and $p(z|x)$. The latter ensures that the posterior q puts probability mass at the same points as the prior p, effectively acting as a regularizer in latent space. We compute z with the reparametrization trick, where **e** can be any appropriate noise distribution. **Right**: Test procedure. At test time, we cut away the inference network q, and instead sample z according to the prior $p(z|x)$. This allows us to make stochastic predictions for y.

For a continuous variable z, the cardinal example is a location-scale transformation of a standard Gaussian distribution. If $q_\phi(z|y,x) = \mathcal{N}(z|\mu_\phi(y,x), \Sigma_\phi(y,x))$, then we can write

$$z = f_\phi(\epsilon, y, x) = \mu_\phi(y,x) + \Sigma_\phi(y,x) \cdot \epsilon, \quad \text{with} \quad \epsilon \sim \mathcal{N}(0,1) \tag{5.3}$$

The gain is that we can now backpropagate through expectations of the random variable z (Kingma and Welling, 2014):

$$\nabla_\phi \mathbb{E}_{z \sim q_\phi(z|x,y)}[\xi(z)] = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,1)}\left[\nabla_\phi \xi(f_\phi(\epsilon, y, x))\right] \tag{5.4}$$

for some function $\xi(\cdot)$ of z. The right-hand term can then be approximated with a Monte-Carlo estimate. This allows us to backpropagate through z (see Fig. 5.3, left), giving the vanilla conditional variational auto-encoder (CVAE) with Gaussian latent variables. The overall train and test procedure is summarized in Figure 5.3. We now consider two methods to improve the capacity of the latent z distribution, that both could improve learning multimodal outcomes.

5.3.2 *Discrete Latent Variables*

As we want to model multimodal outcomes, it seems natural to consider discrete latent variables. However, for the reparametrization trick to be applicable we require the function $f_\phi(\cdot)$ to be differentiable, which is not possible for a discrete variable. It turns out we can get good estimates by making a smooth approximation to the discrete loss (Jang, Gu, and Poole, 2016; Maddison, Mnih, and Teh, 2016).

Let ω_i be an ordered set of class probabilities of a discrete variable z_i[3] with n_i categories. We can draw samples from this distribution through the Gumbel-Max trick:

$$z_i = \text{one-hot}\left(\underset{j\in[1..n_i]}{\arg\max}[g_j + \log \omega_{i,j}]\right) \tag{5.5}$$

with g_j i.i.d. draws from a Gumbel$(0, 1)$ distribution[4]. Since arg max is not differentiable, we can make a softmax approximation to the above equation:

$$z_{i,j} = \frac{\exp((\log \omega_{i,j} + g_j)/\tau)}{\sum_{o=1}^{n_i} \exp((\log \omega_{i,o} + g_o)/\tau)} \qquad \text{for} \quad j = 1, .., n_i \tag{5.6}$$

which is known as the Gumbell-Softmax (Jang, Gu, and Poole, 2016) or Concrete (Maddison, Mnih, and Teh, 2016) distribution. The softmax temperature $\tau \in (0, \infty)$ regulates the discreteness of the approximation: for $\tau \to 0$, the samples effectively become one-hot, while for $\tau \to \infty$, the samples become uniform over the class categories. The above specification allows us to use the reparametrization trick for discrete latent variables, as the noise distribution g is now decoupled from the gradient path $\frac{\delta z}{\delta \omega}$. Note that Eq. 5.6 is a type of reparametrization $f_\phi(\cdot)$ (as introduced in Sec. 5.3.1, with g the noise distribution and $\omega_\phi(x, y)$ the distribution parameters. In practice, we anneal τ from > 1 to 0 over the course of training.

3 We use subscripts z_i to index the elements of the vector random variable z, and double subscripts $z_{i,j}$ to index the categories within one discrete random variable.

4 We can sample from a Gumbel(0,1) distribution by sampling $u \sim$ Uniform$(0, 1)$ and computing $g = -\log(-\log(u))$.

5.3.3 *Transformations of Continuous Variables (Flow)*

We already specified the reparametrization trick for spherical Gaussian latent variables (Eq. 5.3). As spherical Gaussians may be too restricting for multimodality, we can increase the capacity of the latent layer by using transformations of distributions for which we can track the density, an idea originally propose by Rezende and Mohamed (2015).

To obtain more expressive distributions for a continuous random variable $z \in \mathbb{R}^D$ with known density $q(z)$, we consider bijective smooth mappings $h : \mathbb{R}^D \to \mathbb{R}^D$ with inverse h^{-1}. We are interested in the distribution of the transformed variable $z' = h(z)$. As long as we are able to invert h, we can easily compute the density of the transformed variable z':

$$q(z') = q(z) \left| \det\left(\frac{\delta h^{-1}(z')}{\delta z'}\right) \right| = q(z) \left| \det\left(\frac{\delta h(z)}{\delta z}\right) \right|^{-1} \tag{5.7}$$

which is known as the *change-of-variable formula*. If we can specify our neural network to learn transformations which are easily invertible, we can construct more complicated distributions by repeatedly applying the above transformation (while being able to track the density). If we repeatedly apply a sequence of transformations $z^L = h^L \circ ... \circ h^1(z^0)$ for some random variable $z^0 \sim q^0(\cdot)$, then the density of the last variable z^L can be computed as:

$$\log q^L(z^L) = \log q^0(z^0) - \sum_{l=1}^{L} \log \left| \det \frac{\delta z^l}{\delta z^{l-1}} \right| \tag{5.8}$$

The problem with the above transformation is that, especially for high-dimensional domains, computing the determinant is computationally very expensive. An elegant solution appears from the observation that the determinant of a triangular matrix is simply the product of its diagonal terms (Dinh, Sohl-Dickstein, and Bengio, 2016; Kingma et al., 2016). Therefore, given a random variable z of length D, we can specify the transformation $z' = h(z)$ as:

$$\begin{aligned} z'_{1:d} &= z_{1:d} \\ z'_{d+1:D} &= t(z_{1:d}) + z'_{d+1:D} \odot \exp(s(z_{1:d})) \end{aligned} \tag{5.9}$$

The Jacobian of the this transformation is:

$$\frac{\delta z'}{\delta z} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\delta z'_{d+1:D}}{\delta z_{1:d}} & \text{diag}(\exp(s(z_{1:d}))) \end{bmatrix} \tag{5.10}$$

The determinant of this matrix is easily computed as $\exp\left[\sum_i s(z_{1:d})_i\right]$. Note that the $t(\cdot)$ (translation) and $s(\cdot)$ (scale) function can be arbitrarily complex functions, for example deep, non-linear neural networks. In these transformations, we do not need to compute the determinant of $s(\cdot)$ or $t(\cdot)$ to track the density of the random variable z'. Moreover, it is trivial to invert the above transformation:

$$\begin{aligned} z_{1:d} &= z'_{1:d} \\ z_{d+1:D} &= (z_{d+1:D} - t(z'_{1:d})) \odot \exp(-s(z'_{1:d})) \end{aligned} \tag{5.11}$$

This allows us to use the change-of-variable formula of the previous section. We effectively perform an auto-regressive transformation on the z variables. In practice, we repeatedly modify the order of the z variables to have a different part of z transformed in each layer. In Fig. 5.3, we would apply these transformations to a sample from $q(z|x,y)$ before calculating the KL-divergence with $p(z|x)$.

5.3.4 *Enforcing Latent Variable Use*

One of the challenges of training latent variable models is their tendency to overfit the prior early in training. Initially, the likelihood term $p(y|z,x)$ is relatively weak. Therefore, the learning signal is dominated by the KL-divergence, and stochastic optimization gets stuck in the undesirable equilibrium $q(z|y,x) \approx p(z|x)$.

To give a simple illustration, imagine y is strictly bimodal given a fixed x, taking value y_1 or y_2 with $p(y_1|x) = 0.3$ and $p(y_2|x) = 0.7$. We fit a latent model with a single binary variable z taking values z_1 or z_2. Clearly, we want our prior $p(z|x)$ to learn the distribution $\{p(z_1|x) = 0.3, p(z_2|x) = 0.7\}$ (assuming z_1 maps to y_1 and z_2 to y_2, which can of course be interchanged). However, the inference network $q(z|x,y)$ has access to additional information, as it knows which y we need to reconstruct. Therefore, if we present a datapair (x,y_1), then we want our latent distribution more like $\{q(z_1|x,y_1) = 0.999, q(z_2|x,y_1) = 0.001\}$, as this ensures we make a good draw and

good reconstruction. However, for this datapoint this would incur a KL-penalty $D_{KL}(q(z|x,y_1)\|p(z|x)) \approx 1.20$. This illustrates how a good fitting VI model will necessarily incur some KL-cost.

A solution is to enforce each (set of) latent variables to encode a minimum amount of information (Kingma et al., 2016), i.e. force $q(z|y,x)$ to at least have a KL-divergence of λ from the prior $p(z|x)$. The modified objective becomes:

$$\tilde{\mathcal{L}}(y|x) = \mathbb{E}_{(x,y)\sim\mathcal{M}}\Big[\mathbb{E}_{z\sim q(z|y,x)}\big[\log P(y|z,x)\big]\Big] - \sum_{j=1}^{D_z} \max\Big(\lambda, \mathbb{E}_{(x,y)\sim\mathcal{M}}\big[D_{KL}[q(z_j|x,y)\|p(z_j|x)]\big]\Big) \quad (5.12)$$

where D_z is the dimensionality of the latent space z, and $\mathcal{M}$ denotes a mini-batch. Different solutions have been proposed, like KL annealing (Sønderby et al., 2016), but we empirically found them to be less effective.

5.4 RESULTS

We now test the different types of conditional variational inference, introduced in the previous section, on two tasks. Evaluating generative model performance is not straightforward, as standard metrics like mean-squared error (MSE) are non-valid for multimodal outcome distributions. In this work, we evaluate **i)** the log likelihood of a test set under the learned generative model (see Appendix 5.8.2), and (if possible) **ii)** we draw new data from the learned model and compute KL divergences or Hellinger distances with respect to the true data generating distribution. Training details and hyperparameters are described in Appendix 5.8.3.

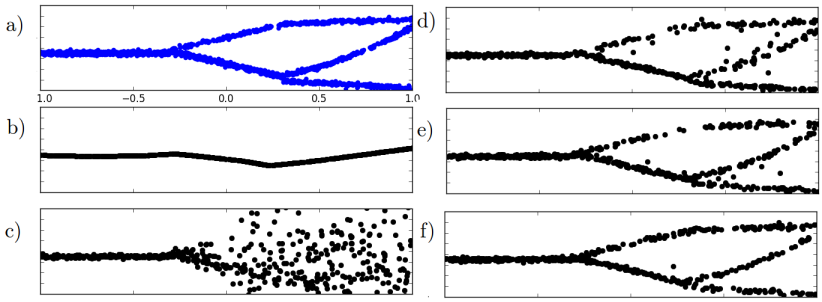


Figure 5.4: Comparison of samples from the models produced by multi-layer perceptron (MLP) and variational inference (VI) networks after training for 30,000 mini-batches. a) Ground truth data. b) MLP (deterministic predictions). c) MLP with stochastic inputs. d) VI with spherical Gaussian. e) VI with spherical Gaussian and 5 layers of flow. f) VI with discrete latent variables. Numerical results are reported in Table 5.1.

5.4.1 *Toy Problem*

We generate a one-dimensional multimodal transition function by sampling $x \sim \text{Uniform}(-1, 1)$ and sampling y from a conditional Gaussian distribution $\mathcal{N}(\cdot|\mu = f(x), \sigma = 0.1)$ according to:

$$p(y|x) = \begin{cases} \mathcal{N}(2.5), & \text{if } x < -0.3 \\ \rho_1\mathcal{N}(4x) + \rho_2\mathcal{N}(-4x), & \text{if } -0.3 \leq x < 0.3 \\ \rho_3\mathcal{N}(5 + \log(x+1)) + \rho_4\mathcal{N}(-x+0.2) + \rho_5\mathcal{N}(5x^2), & \text{if } x \geq 0.3 \end{cases}$$

where $\rho_1 = 0.2$, $\rho_2 = 0.8$, $\rho_3 = 0.3$, $\rho_4 = 0.5$ and $\rho_5 = 0.2$. This generates the multimodal function shown in Figure 5.4a. We study this toy problem to visualize how different architectures will fit this simple data structure with conditional unimodal (left), bimodal (middle) and trimodal (right) structure (see Figure 5.4a left, middle and right parts). Figure 5.4b-f show the samples generated by different models after training on 30,000 mini-batch steps (See Appendix 5.8.3 for details). Table 5.1 displays the variational lower bound (VLB) and negative log-likelihood (NLL) on a test set.

A feed-forward network trained on mean squared error deterministically predicts the conditional expectation (Figure 5.4b). For fair comparison, we also train a feed-forward network that does receive

Table 5.1: Performance on toy domain. All results are averaged over 10 runs. VLB = Variational Lower Bound, NLL = Negative Log Likelihood on test dataset, MLP = Multi-Layer Perceptron, VAE = Variational Auto-Encoder.

Method	**VLB**	**NLL**
MLP (deterministic)	NA	NA
MLP (with stochastic input)	NA	4.49
VAE continuous (n=3, no flow)	0.33	-0.29
VAE continuous (n=3, n_{flow}=5)	0.32	-0.33
VAE discrete (n=3, k=3)	**0.47**	**-0.48**

noise variables ϵ as input but without an inference network (Figure 5.4c). Theoretically this network could learn the same decoder distribution, but without the inference network the model does not converge.

Figure 5.4d-f show the samples generated by different variational methods (spherical Gaussian Sec. 5.3.1), Gaussian with flow (Sec. 5.3.3), and discrete latent variables (Sec. 5.3.2), respectively. We see how these models are much better at fitting the true data distribution. Importantly, notice that the variational approach consistently predicts the deterministic part correctly (left part of the function). This is important, as the network is able to ignore the input noise when needed. Table 5.1 indicates the discrete latent variable model fits this problem best.

5.4.2 *Stochastic Gridworld*

We now study a typical RL gridworld task with multimodal stochastic dynamics. The world is a 7x7 grid (see Figure 5.5) with some walls. The agent (green) starts in the bottom-left, can deterministically move in each cardinal direction, and needs to reach the top-right ($r = +10$). There are two ghosts, starting in locations as shown in Fig. 5.6, top-left. Ghost 1 (red) uniformly chooses one of the available directions. Ghost 2 (blue) has a bias to move to the left or right (40% each), and moves vertically with small probability (10%). Our interest here is to learn to predict this stochasticity from observed data. As state-space we use a vector of length 6 containing the 2D coordinates for the agent and both

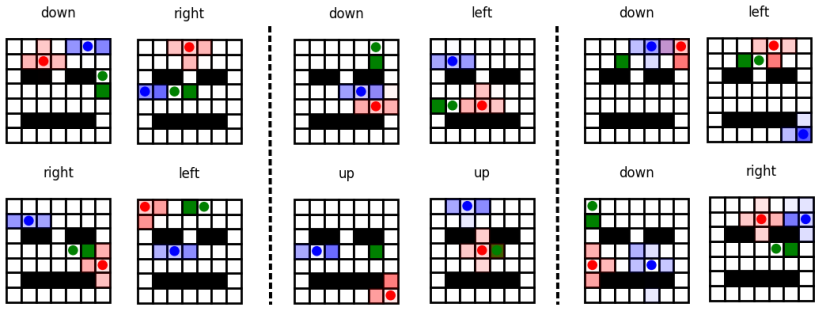


Figure 5.5: Visual predictions on gridworld. Each sub-picture shows the agent (green), ghost 1 (red) and ghost 2 (blue) with current location as a circle, and predicted next location as a shaded box (color intensity corresponds to predicted probability). Black locations are walls, the text above each subplot indicates the action chosen by the agent. **Left:** Continuous latent variables (n=8, no flow), **Middle:** continuous latent variable (n=8, n_{flow} =6), **Right:** discrete latent variables (n=8,k=4). We observe stochastic predictions for the ghosts and deterministic predictions for the agent. Numerical comparison is provided in Table 5.2.

ghosts. Each element in the vector is treated as a categorical variable with 7 classes (for the 7x7 grid).[5]

UNCORRELATED DATA On of the core challenges of RL is the exploration problem, which can make the data we observe strongly correlated. For example, if the agent never explores the top-left region of the domain, we can not expect it to learn an accurate model there. To overcome this problem, we first study an idealized setting in which our dataset consists of the transitions of state-action combinations randomly sampled across state-space.

ON-POLICY AGENT The results on this task are shown in Table 5.2. Compared to Table 5.1 we do not show MLP results anymore for this task. We see the discrete latent variables again perform best on

5 Note that, although this is a discrete MDP, it is not a trivial task to model multimodality here. Indeed, the outcome distribution *per state dimension* is categorical, but the joint distribution (generally) does not factorize over the dimensions. Therefore, we would already need a categorical with $7^6 = 117649$ outcome categories to learn this problem without conditional variational inference, and this would exponentially aggravate in larger state-spaces (e.g. images).

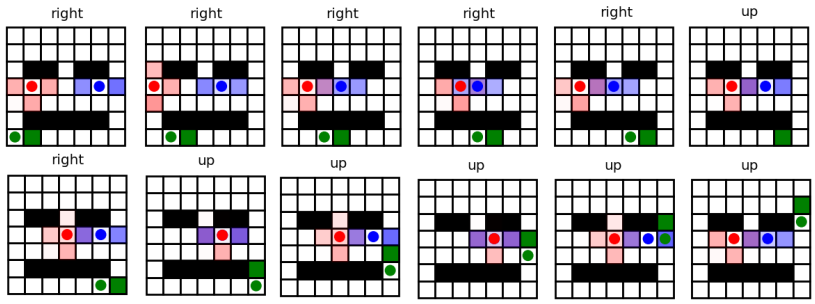


Figure 5.6: *On-policy* predictions for RL agent (see Fig. 5.5 for color explanation). The sub-plots progress row-wise along a roll-out in the learned transition model. Note that this is a true 12-step roll-out, i.e. each next plot is based on sampling a single prediction from the model (we do not observe any true next state along the way).

negative log-likelihood (NLL) evaluation. However, when we compare the learned distribution to the true distribution (which is available for this problem), we see the continuous latent variables without flow actually perform best. We see a conflict between both performance measures (the NLL indicates the discrete model performs best, while the distances with the true distribution point at the continuous latent model). In this case, visual comparison (Figure 5.5) does not show important differences across methods. We therefore conclude that the differences between methods are small for this problem, while the best performance measures for generative models remains an open questions in general (see (Goodfellow, 2016)).

As a next step, we investigate to what extent an RL agent is capable of learning an accurate transition model *on-policy*, i.e. while observing correlated data. Note that the agent is still learning its policy in a model-free sense here (as a deep Q-network (Mnih et al., 2015)), and we simply investigate to what extend the learned transition model is accurate after observing correlated data. Therefore, we evaluate the learned transition model while the agent is executing the policy.

Figure 5.6 shows the results of a roll-out in the learned model under a model-free policy. We see the agent first walking along the bottom corridor, and then moving up in the vertical corridor. Note that the agent consistently predicts its next state deterministically and correctly. In frame 6 it makes a wrong action decision, probably because we execute the behavioural policy with small ϵ-greedy noise. The ghosts

Table 5.2: Performance on gridworld predictions for different types of variational inference. For this table, $\hat{p}$ denotes the predicted distributions by the VAE model, while p denotes the ground truth (which is known for this scenario). VLB = Variational Lower Bound, NLL = Negative Log Likelihood.

Method	**VLB**	**NLL**	$D_{KL}(p\|\hat{p})$	$D_{Hel}(p\|\hat{p})$	$D_{KL}(\hat{p}\|p)$
VAE Continuous (n=8, no flow)	-2.53	2.52	**0.91**	**0.48**	**3.12**
VAE Continuous (n=8, n_{flow}=6)	-2.66	2.70	2.74	0.60	4.29
VAE Discrete (n=8, k=4)	**-2.17**	**2.20**	1.26	0.61	4.75

have multimodal, stochastic behaviour. The first ghost (red) moves uniformly in one of the available directions, which is captured by the red shades around the current ghost location. Note that the model consistently predicts the ghost to move at each step. The second ghost (blue) has a bias to primarily step to the left or right. We also note the difference in the predicted next state between red and blue ghost, matching their true dynamics. Altogether, the agent has learned to predict both the deterministic effects of its own actions as well as its stochastic environment, from on-policy, correlated data.

5.5 RELATED WORK

Variational inference in the conditional setting was previously studied by Sohn, Lee, and Yan (2015) and Walker et al. (2016). Compared to our work, these papers only use spherical Gaussian priors, and do not focus on reinforcement learning tasks. Our work focussed on VI with reparametrization gradients. There is a second line of research on latent variable models that uses score function gradients (Mnih and Gregor, 2014), which are also known as REINFORCE in the RL context. A benefit of reparametrization gradients is that they don't suffer from the high variance usually encountered with score function gradients (a problem also known in RL).

The idea to apply flow to the latent layer originates from Rezende and Mohamed (2015). The transformation in Eq. 5.9 are related to the *affine coupling layers* of Dinh, Sohl-Dickstein, and Bengio (2016), but then applied to the latent layer of a CVAE, while Dinh, Sohl-Dickstein, and Bengio (2016) use them directly from observation level without variational inference. Applying flow transformations at latent VI level was introduced by Kingma et al. (2016), where the authors used fully autoregressive transformations (which are harder to implement com-

pared to our transformations, but potentially have more representational capacity).

To increase the expressivity of the latent approximation, we focussed on different types of latent variables, as well as (normalizing) flow. A third way to increase latent capacity is to factorize the distribution into several layers (Sønderby et al., 2016). However, activating deeper stochastic layers is not straightforward (Sønderby et al., 2016), requiring either batch normalization or weight normalization (Kingma et al., 2016). We defer factorized inference networks to future work, especially in higher-dimensional tasks.

The different deep generative models discussed in Section 5.2 are not mutually exclusive. For example, the variational lossy auto-encoder (LVAE) (Chen et al., 2016) combines variational inference with PixelCNN-based decoders (Oord et al., 2016). Such architectures force high-level conceptual information into the latent level, while the decoder should capture fine-grained details. This could be beneficial to sparsify the latent layer and as such benefit RL planning as well.

There is relatively little work on Bayesian Neural Networks for RL. Closest to ours is the work by Depeweg et al. (2016), who study VI to estimate both transition function stochasticity (as studied in this work) combined with uncertainty (due to limited data). Compared to their work, we use a parametric inference network which allows us to generalize in the inference part, while they perform VI per individual datapoint. Second, they only considered Gaussian latent variables, while we investigate discrete latent variables and normalizing flow as well. The results of Depeweg et al. (2016) also show the ability to learn multi-modal stochasticity, and additionally show the benefit of planning over the model. Watter et al. (2015) also used variational auto-encoders in a control task, but only as a regularizer for learning representations, not to make stochastic predictions. Gal, McAllister, and Rasmussen (2016) use Bayesian neural networks, in the form of Bayesian dropout, to track uncertainty (due to limited data) in transition dynamics estimation.

Finally, there is also a line of RL research that uses the transition function target to speed-up model-free RL. This idea has been identified as RL with 'auxiliary tasks' (Jaderberg et al., 2016). The gradients of the transition function predictions are denser compared to the sparse RL training signal, and used to speed-up training of deeper network layers shared between policy and transition network. However, this approach does not learn stochastic transitions (but could benefit from it, as it improves the learning signal), nor is it used for sample-based planning as in model-based RL.

5.6 FUTURE WORK

One clear line of future work is to use these transition models to improve agent performance, by planning over the model with either a given or learned reward function. Depeweg et al. (2016) already provided a study in this direction. Compared to their work, it would especially be interesting to apply more adaptive roll-outs in the model, like Monte Carlo Tree Search (MCTS). Moreover, it would be important to evaluate these methods in high-dimensional RL tasks, e.g. with convolutional neural networks on raw pixel data (Oh et al., 2015). Another extension is to use these models to improve exploration in stochastic domains (e.g. (Houthooft et al., 2016; Oh et al., 2015)).

An important second challenge, briefly mentioned in the Introduction and Section 5.4.2, is planning under uncertainty. RL initially provides correlated data from a limited part of state-space. When planning over this model, we should not extrapolate too much, nor trust our model too early with limited data. Planning under uncertainty was for example studied by Gal, McAllister, and Rasmussen (2016) and Houthooft et al. (2016). Note that 'uncertainty' (due to limited data) is fundamentally different from the 'stochasticity' (true probabilistic nature of the domain) discussed in this chapter.

A third challenge for transition dynamics estimation is memory (partial observability), when the current state does not provide all available information to make a prediction. Proposed solutions are recurrent neural networks (RNN) or Neural Turing Machines (NTM), which have both been studied in the variational inference context (in (Chung et al., 2015) and (Gemici et al., 2017), respectively).

Combining stochasticity, uncertainty and memory in one function approximator would be an important integrating step in model-based RL.

5.7 CONCLUSION

This chapter studied multimodal transition function estimation for RL agents, with a focus on variational inference with different types of latent variables. Our experiments show variational inference is a robust method to discriminate deterministic and stochastic elements of the transition function using function approximation, clearly improving over discriminative training. We verified results on a typical RL domain where tabular learning would be infeasible, showing the ability of these

models to learn the multimodal transition dynamics online. We did not observe important distinction in performance between the different types of latent variables studied. Therefore, for the domain size studied in this work, it seems safe to use the standard spherical Gaussian conditional VAE. Our results are generally applicable in model-based RL, and help solve a fundamental problem of many domains: the complex stochastic behaviour of its transition dynamics.

5.8 APPENDIX

5.8.1 *Variational Auto-Encoder (VAE) Training Objective*

We can obtain a tighter bound on Equation 5.2 by using importance sampling (Burda, Grosse, and Salakhutdinov, 2015). We sample M values of z per datapoint, and average over them inside the log. Otherwise, the model strongly penalizes for single samples that explain the objective poorly. Second, instead of the KL divergence we optimize Renyi α-divergences (Li and Turner, 2016)). We use α=0.5 according to the results by Depeweg et al. (2016), which makes the divergence term become a function of the Hellinger distance (Li and Turner, 2016). The combined objective, known as the variational Renyi (VR) bound (Li and Turner, 2016) is:

$$\mathcal{L}_{VR}(y|x) = \frac{1}{1-\alpha} \log \frac{1}{M} \sum_{m=1}^{M} \left[\left(\frac{p(y, z^m|x)}{q(z^m|y, x)} \right)^{1-\alpha} \right] \tag{5.13}$$

with $z^m \sim q(\cdot|x, y)$.

5.8.2 *Test Set Negative Log-likelihood (NLL) for VAE*

We are interested in the likelihood $p(y|x)$ of a set of test data $\{x_i, y_i\}_{i=1}^N$. We therefore need to marginalize over z:

$$p(y|x) = \mathbb{E}_{z \sim p(\cdot|x)} \Big[p(y|z, x) \Big] \tag{5.14}$$

One problem with this estimator is that we may need many empirical samples from z to get an accurate estimate. As an alternative, we

estimate the quantity through importance sampling, by sampling from $q(\cdot|x_i, y_i)$ instead of $p(\cdot|x_i)$:

$$p(y|x) = \mathbb{E}_{z\sim q(\cdot|x,y)}\left[p(y|z,x)\frac{p(z|x)}{q(z|x,y)}\right] \tag{5.15}$$

The empirical estimate of the negative log likelihood (NLL), as reported in the results section, then becomes

$$-\log p(y|x) = -\frac{1}{N}\sum_{i=1}^{N}\log\left[\frac{1}{M}\sum_{m=1}^{M}p(y_i|z_i^m, x_i)\frac{p(z_i^m|x_i)}{q(z_i^m|x_i, y_i)}\right] \tag{5.16}$$

with $z_i^m \sim q(\cdot|x_i, y_i)$.

5.8.3 *Training Details*

For all experiments we follow standard train, validation and test set set-up. For all domains, we train the VAE target on $k = 3$ importance samples with Renyi-α divergence for $\alpha = 0.5$ (see appendix 5.8.1). This gave us slightly better results compared to the 'default' settings of $k = 1$ and $\alpha = 1.0$. All models are trained in Tensorflow using Adam optimizer.

Toy Domain: We draw a training set of size 2000, and independent validation and test sets of size 500 and 2000, respectively. The decoder distribution is Gaussian, where we also learn its standard deviation. We train for 30000 batches with batch size 64, with a learning rate linearly annealed from 0.005 to 0.0005 over 90% of training steps. The minimal KL penalty per dimension λ (Eq. 5.12) is fixed at 0.07.

The generative network has three layers with 50 units per layer and Relu non-linearities. The inference network has two layers with 30 units per layer and Relu non-linearities. For the discrete latent variables, we anneal the temperature from 2.0 to 0.001 over 70% of training steps.

Gridworld: For the first task, we repeatedly draw training data by sampling a new state-action combination uniformly across state-space, and sampling a single transition. Optimal model performance is based on a VAE performance on a validation and test set of size 750 and 1500 respectively. The decoder distribution is discrete taking values in 7 categories. We train on mini-batches of size 32 for 75000 iterations,

with a learning rate linearly annealed from 0.0005 to 0.0001 over 70% of training steps. The generative network has three layers with 250 units per layer and Relu non-linearities. The inference network has two layers with 100 units per layer and Relu non-linearities. The minimal KL penalty per dimension λ (Eq. 5.12) is fixed at 0.07.

For the on-policy evaluation, the RL policy is trained as a deep Q-network (Mnih et al., 2015) with target network and no experience replay. The state-action value network has three layers of 50 units and Relu activations. Given a mini-batch $\mathcal{M}$ of roll-out data under the current policy, the network is trained on the 1 step Q-learning objective:

$$L_{RL}(\eta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \eta^-) - Q(s, a; \eta)\right)^2\right] \quad (5.17)$$

where s, a, r denote state, action and reward, $Q(s, a)$ is the expected discounted return (discount parameter $\gamma = 0.99$) from state s and action a under the current policy, η are the parameters in the value function network, and η^- the parameters in the target network (which are fixed in the above loss, and only updated every 500 steps). During learning, we follow an ϵ-greedy policy with ϵ linearly decayed from 1.0 to 0.10 over 60% of training steps.

6

ALPHA ZERO IN CONTINUOUS ACTION SPACE [1]

ABSTRACT

A core novelty of AlphaZero is the interleaving of tree search and deep learning, which has proven very successful in board games like Chess, Shogi and Go. These games have a discrete action space. However, many real-world reinforcement learning domains have continuous action spaces, for example in robotic control, navigation and self-driving cars. This chapter presents the necessary theoretical extensions of Alpha Zero to deal with continuous action space. We also provide some preliminary experiments on the Pendulum swing-up task, empirically showing the feasibility of our approach. Thereby, this work provides a first step towards the application of iterated search and learning in domains with a continuous action space.

6.1 INTRODUCTION

We already introduced the AlphaZero algorithm in Chapters 3 and 4. In particular, it is a successful approach to combine planning and learning, for example achieving state-of the art performance in Chess, Shogi (Silver et al., 2017b) and the game of Go (Silver et al., 2016, 2017c). In this chapter, we will experimentally investigate this combination of planning and learning, showing how it can be extended to continuous action spaces.

Compared to traditional RL techniques, the key innovation of AlphaZero is the use of a small, nested tree search as a policy improvement operator. Whereas traditional reinforcement learning treats each environment step or trace as an individual training target, AlphaZero aggregates the information of multiple traces in a tree, and eventually aggregates these tree statistics into targets to train a neural network. The neural network is then used as a prior to improve new tree searches. This approach closes the loop between search and function approximation (Figure 6.1). In section 6.6 we further discuss why this works so well.

1 Chapter based on: Moerland TM, Broekens J, Plaat A, Jonker CM. AoC: Alpha Zero in Continuous Action Space. 2018. Planning and Learning (PAL) Workshop @ International Conference on Machine Learning, 2018.

Many real-world problems, such as robotics control, navigation and self-driving cars, have a continuous action space. This chapter therefore investigates the applicability of the AlphaZero paradigm to problems with a continuous action space. Compared to the AlphaZero paradigm for discrete action spaces, we require:

1. A Monte Carlo Tree Search (MCTS) method that works in continuous action space. We built here on earlier results on *progressive widening* (Section 6.3.1).

2. Incorporation of a continuous prior to steer a new MCTS iteration. While AlphaZero uses the discrete density as a prior in a (P)UCT formula (Kocsis and Szepesvári, 2006; Rosin, 2011), we need to leverage a continuous density (which is unbounded) to direct the next MCTS iteration (Section 6.3.2)

3. A training method. AlphaZero transforms the MCTS visitation counts to a discrete probability distribution. We need to estimate a continuous density from a set of support points, and specify an appropriate training loss in continuous policy space (Section 6.4).

The remainder of this chapter is organized as follows. Section 6.2 presents essential preliminaries on MCTS and the concepts of AlphaZero. Section 6.3 discusses the required MCTS modifications for a continuous action space with a continuous prior (Fig. 6.1, upper part of the loop). In Section 6.4 we cover the generation of training targets from the tree search and specify an appropriate neural network loss (Fig. 6.1, lower part of the loop). Sections 6.5, 6.6 and 6.7 present experiments, discussion and conclusions. Code to replicate experiments is available from `https://github.com/tmoer/a0c`.

6.2 PRELIMINARIES

Please refer to Chapter 2 for an introduction of the Markov Decision Process problem, and the relevant notation. We here present a brief introduction of the well-known MCTS algorithm (Browne et al., 2012; Coulom, 2006). In particular, we will introduce a variant of the PUCT algorithm (Rosin, 2011), as also used in AlphaZero (Silver et al., 2017b,c). Every action node in the tree stores statistics $\{n(\boldsymbol{s},\boldsymbol{a}), W(\boldsymbol{s},\boldsymbol{a}), Q(\boldsymbol{s},\boldsymbol{a})\}$, where $n(\boldsymbol{s},\boldsymbol{a})$ is the visitation count, $W(\boldsymbol{s},\boldsymbol{a})$ the cumulative return over all roll-outs through $(\boldsymbol{s},\boldsymbol{a})$, and $Q(\boldsymbol{s},\boldsymbol{a}) = W(\boldsymbol{s},\boldsymbol{a})/n(\boldsymbol{s},\boldsymbol{a})$ is the mean action value estimate. PUCT alternates four phases:

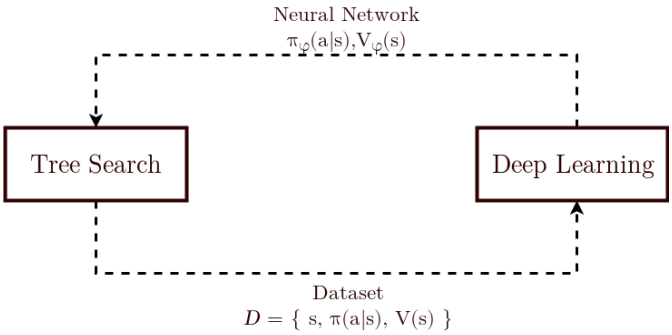


Figure 6.1: Iterated tree search and function approximation.

1. **Select** In the first stage, we descent the tree from the root node according to:

$$\pi_{tree}(\boldsymbol{a}|\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} \left[Q(\boldsymbol{s},\boldsymbol{a}) + c_{\text{puct}} \cdot \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) \cdot \frac{\sqrt{n(\boldsymbol{s})}}{n(\boldsymbol{s},\boldsymbol{a})+1} \right] \quad (6.1)$$

   where $n(\boldsymbol{s}) = \sum_{\boldsymbol{a}} n(\boldsymbol{s},\boldsymbol{a})$ is the total number of visits to state s in the tree, $c_{\text{puct}} \in \mathbb{R}^+$ is a constant that scales the amount the exploration/optimism, and $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ is the probability assigned to action $\boldsymbol{a}$ by the network.[2] The tree policy is followed until we either reach a terminal state or select an action we have not tried before.

2. **Expand** We next expand the tree with a new leaf state $\boldsymbol{s}_L$[3] obtained from simulating the environment with the new action from the last state in the current tree.

2 This equation differs from the standard UCT-like formulas in two ways. The $\pi_\phi(a|s)$ term scales the confidence interval based on prior knowledge, as stored in the the policy network. The $+1$ term in the denominator ensures that the policy prior already affects the decision when there are unvisited actions. Otherwise, every untried action would be tried at least once, since without the $+1$ term Eq. 6.3 becomes ∞ for untried actions. This is undesirable for large action spaces and small trees, where we directly want to prune the actions that we already know are inferior from prior experience.

3 We use superscript $\boldsymbol{s}^t$ to index real environment states and actions, subscripts $\boldsymbol{s}_d$ to index states and actions at depth d in the search tree, and double subscripts $\boldsymbol{a}_{d,j}$ to index a specific child action j at depth d. For example, $\boldsymbol{a}_{0,0}$ is the first child action at the root $\boldsymbol{s}_0$. At every timestep t, the tree root $\boldsymbol{s}_0 := \boldsymbol{s}^t$, i.e. the current environment state becomes the tree root.

3. **Roll-out** We then require an estimate of the value $V(\boldsymbol{s}_L)$ of the new leaf node, for which MCTS uses the sum of reward of a (random) roll-out $\mathrm{R}(\boldsymbol{s}_L)$. In AlphaZero, this gets replaced by the prediction of a value network $\mathrm{R}(\boldsymbol{s}_L) := V_\phi(\boldsymbol{s}_L)$.

4. **Back-up** Finally, we recursively back-up the results in the tree nodes. Denote the current forward trace in the tree as $\{\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, ..\boldsymbol{s}_{L-1}, \boldsymbol{a}_{L-1}, \boldsymbol{s}_L\}$. Then, for each state-action edge $(\boldsymbol{s}_i, \boldsymbol{a}_i)$, $L > i \geq 0$, we recursively estimate the state-action value as

$$\mathrm{R}(\boldsymbol{s}_i, \boldsymbol{a}_i) = r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \gamma \mathrm{R}(\boldsymbol{s}_{i+1}, \boldsymbol{a}_{i+1}). \tag{6.2}$$

   where $\mathrm{R}(\boldsymbol{s}_L, \boldsymbol{a}_L) := \mathrm{R}(\boldsymbol{s}_L)$. We then increment $W(\boldsymbol{s}_i, \boldsymbol{a}_i)$ with the new estimate $\mathrm{R}(\boldsymbol{s}_i, \boldsymbol{a}_i)$, increment the visitation count $n(\boldsymbol{s}_i, \boldsymbol{a}_i)$ with 1, and set the mean estimate to $Q(\boldsymbol{s}_i, \boldsymbol{a}_i) = W(\boldsymbol{s}_i, \boldsymbol{a}_i)/n(\boldsymbol{s}_i, \boldsymbol{a}_i)$. We repeatedly apply this back-up one step higher in the tree until we reach the root node s_0.

This procedure is repeated until the overall MCTS trace budget N_{trace} is reached. MCTS returns a set of root actions $A_0 = \{\boldsymbol{a}_{0,0}, \boldsymbol{a}_{0,1}, .., \boldsymbol{a}_{0,m}\}$ with associated counts $N_0 = \{n(\boldsymbol{s}_0, \boldsymbol{a}_{0,0}), n(\boldsymbol{s}_0, \boldsymbol{a}_{0,1}), .., n(\boldsymbol{s}_0, \boldsymbol{a}_{0,m})\}$. Here m denotes the number of child actions, which for AlphaZero is always fixed to the cardinality of the discrete action space $m = |\mathcal{A}|$. We select the real action $\boldsymbol{a}^t$ to play in the environment by sampling from the probability distribution obtained from normalizing the action counts at the root $\boldsymbol{s}_0 (= \boldsymbol{s}^t)$:

$$\boldsymbol{a}^t \sim \hat{\pi}(\boldsymbol{a}|\boldsymbol{s}_0), \quad \text{where} \quad \hat{\pi}(\boldsymbol{a}|\boldsymbol{s}_0) = \frac{n(\boldsymbol{s}_0, \boldsymbol{a})^{\frac{1}{\tau}}}{\sum_{\boldsymbol{b} \in A_0} n(\boldsymbol{s}_0, \boldsymbol{b}^{\frac{1}{\tau}})}, \tag{6.3}$$

where τ denotes a temperature parameter that scales the greediness of the target. Note that the denominator is larger than N_{trace}, since we store the subtree that belongs to the picked action $\boldsymbol{a}^t$ for the MCTS at the next timestep.

Finally, in AlphaZero we introduce two neural networks: one to estimate a parametrized policy $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$, and one to estimate the state value $V_\phi(\boldsymbol{s})$. Both networks share the initial layers. The joint set of parameters of both networks is denoted by ϕ. These neural networks interact with the planning cycle, since they are 1) trained on targets extracted from the tree search, and 2) potentially influence new tree

search iterations. This scheme is visualized in Fig. 6.1. We will now discuss how this integration of planning and learning, introduced by AlphaZero, can be modified for the continuous action space setting.

6.3 TREE SEARCH IN CONTINUOUS ACTION SPACE

As noted in the introduction, we require two modifications to the MCTS procedure: 1) a method to deal with continuous action spaces, and 2) a way to include a continuous policy network into the MCTS search.

6.3.1 *Progressive Widening*

During MCTS with a discrete action space we evaluate the PUCT formula for *all* actions. However, in continuous action space we can not enumerate all actions, i.e., there are actually infinitely many actions in a continuous set. A practical solution to this problem is *progressive widening* (Chaslot et al., 2008a; Coulom, 2007), where we make the number of child actions of state $\boldsymbol{s}$ in the tree $m(\boldsymbol{s})$ a function of the total number of visits to that state $n(\boldsymbol{s})$. This implies that actions with good returns, which will get more visits, will also gradually get more child actions for consideration. In particular, Couëtoux et al. (2011) use

$$m(\boldsymbol{s}) = c_{pw} \cdot n(\boldsymbol{s})^{\kappa} \tag{6.4}$$

for constants $c_{pw} \in \mathbb{R}^+$ and $\kappa \in (0, 1)$, making $m(\boldsymbol{s})$ a polynomial (root) function of $n(\boldsymbol{s})$. The idea of progressive widening was introduced by Coulom (2007), who made $m(\boldsymbol{s})$ a logarithmic function of $n(\boldsymbol{s})$. Although originally conceived for discrete domains, this technique turns out to be an effective solution for continuous action space as well (Couëtoux et al., 2011).

6.3.2 *Continuous policy network prior*

For now assume we manage to train a policy network $\pi_\phi(\boldsymbol{s})$ from the results of the MCTS procedure. AlphaZero can enumerate the probability for all available discrete actions, and uses this probability as a prior scaling on the upper confidence bound term in the UCT formula (Eq. 6.1). For the continuous policy space, we could use a similar equation, where we use $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ of the considered $\boldsymbol{a}$ as predicted

by the network. However, the continuous $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ is unbounded.[4] This gives us the risk of rescaling/stretching the confidence intervals too much. Another option - which we consider in this work - is to use the policy network to sample new child actions in the tree search (when adding a new action based on progressive widening). Thereby, the policy net steers the actions that we will consider in the tree search. This has a similar effect as Eq. 6.1 for AlphaZero does, as it effectively prunes away child actions in subtrees of which we already know that they perform poorly.

6.4 NEURAL NETWORK TRAINING IN CONTINUOUS ACTION SPACE

We next want to use the MCTS output to improve our neural networks. Compared to AlphaZero, the continuous action space forces us to come up with a different policy network specification, policy target calculation and training loss. These aspects are covered in Section 6.4.1. Afterwards, we briefly detail the value network training procedure, including a slight variant of the value target estimation (Section 6.4.2).

6.4.1 *Policy Network*

POLICY NETWORK DISTRIBUTION We require a neural network that outputs a continuous density. However, continuous action spaces usually have some input bounds. For example, when we learn the torques or voltages on a robot manipulator, then a too extreme torque/voltage may break the motor altogether. Therefore, continuous action spaces are generally symmetrically bounded to some $[-c_b, c_b]$ interval, for scalar $c_b \in \mathbb{R}^+$. To ensure that our density predicts in this range, we use a transformation of a factorized Beta distribution $\pi_\phi(\boldsymbol{a}|\boldsymbol{s}) = g(\boldsymbol{u})$, with elements $u_i \sim \text{Beta}(\alpha_i(\phi), \beta_i(\phi))$ and deterministic transformation $g(\cdot)$. Details are provided in Appendix 6.8.1. Note that the remainder of this section holds for any $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ network output distribution from which we know how to sample and evaluate (log) densities.

TRAINING TARGET We want to transform the result of the MCTS with progressive widening to a continuous target density $\hat{\pi}$ (to training our neural network with). Recall that MCTS returns the sets A_0 and

4 For a discrete probability distribution, $\pi(\boldsymbol{a}) \leq 1 \; \forall \boldsymbol{a}$. However, although the probability density function (pdf) of continuous random variables integrates to 1, i.e. $\int \pi(\boldsymbol{a}|\boldsymbol{s}) \mathrm{d}\boldsymbol{a} = 1$, this does not bound the value of the pdf $\pi(\boldsymbol{a})$ at a particular point $\boldsymbol{a}$, i.e. $\pi(\boldsymbol{a}) \in [0, \infty)$.

N_0 of root actions and root counts, respectively. We can not normalize these counts like AlphaZero does (Eq. 6.3) for the discrete case. The only assumption, similar to AlphaZero, that we make here is that the density at a root action $\boldsymbol{a}_{0,i}$ is proportional to the visitation counts, i.e.[5]

$$\hat{\pi}(\boldsymbol{a}_i|\boldsymbol{s}) = \frac{n(\boldsymbol{s},\boldsymbol{a}_i)^\tau}{Z(\boldsymbol{s},\tau)} \tag{6.5}$$

where $\tau \in \mathbb{R}^+$ specifies some temperature parameter, and $Z(\boldsymbol{s},\tau)$ is a normalization term (that is assumed to not depend on $\boldsymbol{a}_i$, as the density at the support points is only proportional to the counts). Note that this does not define a proper density, as we never specified a density in between the support points. However, we can ignore this issue, as we will only consider the loss at the support points.

LOSS In short, our main idea is to leave the normalization and generalization of the policy over the action space to the network loss. If we specify a network output distribution that enforces $\int_a \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) = 1$, i.e., making it a proper continuous density, then we may specify a loss with respect to a target density $\hat{\pi}(\boldsymbol{a}|\boldsymbol{s})$, *even when the target density is only known on a relative scale*. More extreme counts (relative densities) will produce stronger gradients, and the restrictions of the network output density will ensure that we can not pull the density up or down over the entire support (as it needs to integrate to 1). This way, we make our network output density mimic the counts on a relative scale.

We will first give a general derivation, acting as if $\hat{\pi}(\boldsymbol{a}|\boldsymbol{s})$ is a proper density, and swap in the empirical density at the end. We minimize a policy loss $\mathcal{L}^{\text{policy}}(\phi)$ based on the Kullback-Leibler divergence between the network output $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ and the empirical density $\hat{\pi}(\boldsymbol{a}|\boldsymbol{s})$ (Eq. 6.5):

$$\begin{aligned}\mathcal{L}^{\text{policy}}(\phi) &= \mathrm{D}_{KL}\Big(\pi_\phi(\boldsymbol{a}|\boldsymbol{s}) \| \hat{\pi}(\boldsymbol{a}|\boldsymbol{s})\Big) \\ &= \mathrm{E}_{a\sim\pi_\phi(\boldsymbol{a}|\boldsymbol{s})}\Big[\log \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - \log \hat{\pi}(\boldsymbol{a}|\boldsymbol{s})\Big]\end{aligned} \tag{6.6}$$

We may use the REINFORCE[6] trick to get an unbiased gradient estimate of the above loss:

5 The remainder of this section always concerns the root state $\boldsymbol{s}_0$ and root actions $\boldsymbol{a}_{0,i}$. Therefore, we omit the depth subscript (of 0) for readability.

6 The REINFORCE trick (Williams, 1992), also known as the likelihood ratio estimator, is an identity regarding the derivative of an expectation, when the expectation depends on the parameter towards which we differentiate: $\nabla_\phi \mathrm{E}_{\boldsymbol{a}\sim p_\phi(\boldsymbol{a})}[f(\boldsymbol{a})] = \mathrm{E}_{\boldsymbol{a}\sim p_\phi(\boldsymbol{a})}[f(\boldsymbol{a})\nabla_\phi \log p_\phi(\boldsymbol{a})]$, for some function $f(\cdot)$ of $\boldsymbol{a}$.

$$\begin{aligned}\nabla_\phi \mathcal{L}^{\text{policy}}(\phi) &= \nabla_\phi \mathrm{E}_{a\sim\pi_\phi(\boldsymbol{a}|\boldsymbol{s})}\Big[\log \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - \tau \log n(\boldsymbol{a},\boldsymbol{s}) \\ &\qquad + \log Z(\boldsymbol{s},\tau)\Big] \\ &= \mathrm{E}_{a\sim\pi_\phi(\boldsymbol{a}|\boldsymbol{s})}\Big[\Big(\log \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) - \tau \log n(\boldsymbol{a},\boldsymbol{s}) \\ &\qquad + \log Z(\boldsymbol{s},\tau)\Big)\nabla_\phi \log \pi_\phi(\boldsymbol{a}|\boldsymbol{s})\Big]\end{aligned}$$

We now drop $Z(\boldsymbol{s},\tau)$ since it does not depend on ϕ (or chose an appropriate state-dependent baseline, as is common with REINFORCE estimators). Moreover, we replace the expectation over $\boldsymbol{a} \sim \pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ with the empirical support points $\boldsymbol{a}_i \sim \mathcal{D}_\boldsymbol{s}$, where $\mathcal{D}_\boldsymbol{s}$ denotes the subset of the database containing state $\boldsymbol{s}$. Our final gradient estimator becomes

$$\begin{aligned}\nabla_\phi \mathcal{L}^{\text{policy}}(\phi) = \mathrm{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}_i\sim\mathcal{D}_s}\Big[&\Big(\log \pi_\phi(\boldsymbol{a}_i|\boldsymbol{s}) - \tau \log n(\boldsymbol{s},\boldsymbol{a}_i)\Big) \\ &\cdot \nabla_\phi \log \pi_\phi(\boldsymbol{a}_i|\boldsymbol{s})\Big]\end{aligned} \tag{6.7}$$

ENTROPY REGULARIZATION Continuous policies have a risk to collapse (Haarnoja et al., 2018). If all sampled actions are close to each other, then the distribution may narrow too much, loosing any exploration. In the worst case, the distribution may completely collapse, which will produce NaNs and break the training process. As we empirically observed this problem, we augment the training objective with an entropy maximization term.[7] This prevents the policy from collapsing, and additionally ensures a minimum level of exploration. We define the entropy loss as

$$\mathcal{L}^H(\phi) = H(\pi_\phi(\boldsymbol{a}|\boldsymbol{s})) = -\int \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) \log \pi_\phi(\boldsymbol{a}|\boldsymbol{s}) \mathrm{d}\boldsymbol{a}. \tag{6.8}$$

Details on the computation of the entropy for the case where $\pi_\phi(\boldsymbol{a}|\boldsymbol{s})$ is a transformed Beta distribution are provided in Appendix 6.8.2. The full policy loss thereby becomes

$$\mathcal{L}^\pi(\phi) = \mathcal{L}^{\text{policy}}(\phi) - \lambda \mathcal{L}^H(\phi), \tag{6.9}$$

7 As an alternative solution, we could also consider adding extra noise to the MCTS search. However, it is harder to tune the proper amount of exploration in the MCTS search. Entropy regularization is a more straightforward approach to prevent collapse.

where λ is a hyperparameter that scales the contribution of the entropy term to the overall loss.

6.4.2 *Value Network*

Value network training is almost identical to the AlphaZero specification. The only thing we modify is the estimation of $\hat{V}(\boldsymbol{s})$, the training target for the value. AlphaZero uses the eventual return of the full episode as the training target for every state in the trace. This is an unbiased, but high-variance signal (in reinforcement learning terminology (Sutton and Barto, 2018), it uses a full Monte Carlo target). Instead, we use the MCTS procedure as a value estimator, leveraging the action value estimates $Q(s_0, a)$ at the root s_0. We could weigh these according to the visitation counts at the root. However, we usually built relatively small trees,[8] for which a non-negligible fraction of the traces are exploratory. Therefore, we propose an *off-policy* estimate of the value at the root:

$$\hat{V}(\boldsymbol{s}_0) = \max_{\boldsymbol{a}} Q(\boldsymbol{s}_0, \boldsymbol{a}) \tag{6.10}$$

The value loss $\mathcal{L}^V(\phi)$ is a standard mean-squared error loss:

$$\mathcal{L}^V(\phi) = \mathrm{E}_{\boldsymbol{s}\sim\mathcal{D}}\left[\left(V_\phi(\boldsymbol{s}) - \hat{V}(\boldsymbol{s})\right)^2\right]. \tag{6.11}$$

6.5 EXPERIMENTS

Figure 6.2 shows the results of our algorithm on the Pendulum-v0 task from the OpenAI Gym (Brockman et al., 2016). The curves show learning performance for different computational budgets per MCTS at each timestep. Note that the x-axis displays true environment steps, which includes the MCTS simulations. For example, if we use 10 traces per MCTS, then every real environment step counts as 10 on this scale.

First, we observe that our continuous AlphaZero version does indeed learn on the Pendulum task. Interestingly, we observe different learning performance for different tree sizes, where the 'sweet spot' appears to

8 AlphaGo Zero uses 1600 traces per timestep. We evaluate on smaller domains, and have less computational resources.

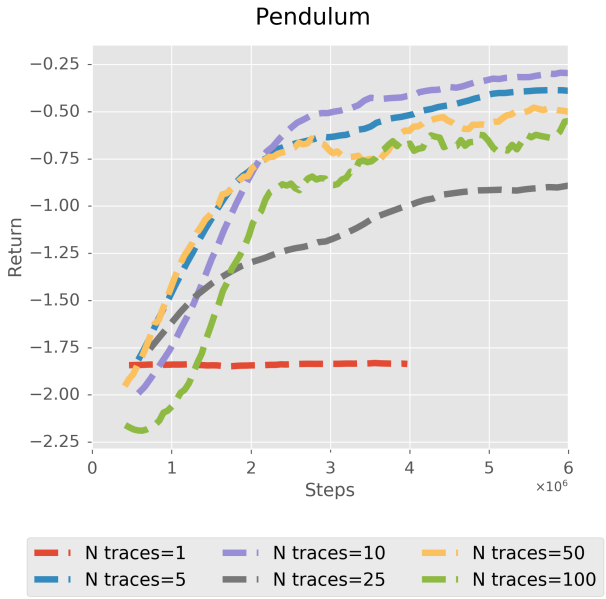


Figure 6.2: Learning curves for Pendulum domain. Compared to the OpenAI Gym implementation we rescale every reward by a factor 1/1000 (which leaves the task and optimal solution unchanged). Results averaged over 10 repetitions.

be at an intermediate tree size (of 10). For larger trees, we complete less episodes (a single episode takes longer) and therefore we have less training targets to train our neural network on. Therefore, although each individual trace gets more budget, it takes longer before the tree search starts to profit from improved network estimates (generalization).

We train our neural network after every completed episode. However, the runs with smaller tree sizes complete much more episodes compared to the runs with a larger tree size. Moreover, the data generated from larger tree searches could be deemed 'more trustworthy', as we spend more computational effort in generating them. We try to compensate for this effect by making the number of training epochs over the database

after each episode proportional to the size of the nested tree search. Specifically, after each episode we train for

$$n_{\text{epochs}} = \left\lceil \frac{N_{\text{traces}}}{c_e} \right\rceil \tag{6.12}$$

for constant $c_e \in \mathbb{R}^+$ and $\lceil \cdot \rceil$ denoting the ceiling function. In our experiments we set $c_e = 20$. This may explain why the run with $N_{\text{traces}} = 25$ performs suboptimal compared to the others, as the non-linearity in Eq. 6.12 (due to the ceiling function) may accidentally turn out bad for this number of tree traces. Moreover, note that the learning curve of training with a tree size of 1 is shorter than the other curves. This happens because we gave each run an equal amount of wall-clock time. The run with tree size 1 finishes much more episodes, and because $c_e > 1$ it still trains more frequently than the other runs, which makes it eventually perform less total steps in the domain.

IMPLEMENTATION DETAILS We use a three layer neural network with 128 units in each hidden layer and ELu activation functions. For the MCTS we set $c_{\text{puct}} = 0.05$, $c_{\text{pw}} = 1$ and $\kappa = 0.5$, and for the policy loss $\lambda = 0.1$ and $\tau = 0.1$. We train the networks in Tensorflow (Abadi et al., 2016), using RMSProp optimizer on mini-batches of size 32 with a learning rate of 0.0001. Episodes last at maximum 300 steps.

6.6 DISCUSSION

The results in Fig. 6.2 reveal an interesting trade-off in the iterated tree search and function approximation paradigm. We hypothesize that the strength of tree search is the in the locality of information. Each edge stores its own statistics, and this makes it easy to locally separate the effect of actions. Moreover, the forward search gives a more stable value estimate, smoothing out local errors in the value network. In contrast, the strength of the neural network is generalization. Frequently, we re-encounter the (almost) same state in a different subtree during a next episode. Supervised learning is a natural way to generalize the already learned knowledge from a previous episode.

One of the key observations of the present chapter is that we actually need both. If we *only* perform tree search, then we eventually fail at solving the domain because all information is kept locally. In contrast, if we only build trees of size 1, then we are continuously generalizing

without ever locally separating decisions and improving our training targets. Our results suggest that there is actually a sweet spot halfway, where we build trees of moderate size, after which we perform a few epochs of training. We will further explore this topic in the next chapter.

Future work will test the AoC algorithm in more complicated, continuous action space tasks (Brockman et al., 2016; Todorov, Erez, and Tassa, 2012). Moreover, our algorithm could profit from recent improvements in the MCTS algorithm (Moerland et al., 2018b) and other network architectures (Szegedy et al., 2015), as also leveraged in AlphaZero.

6.7 CONCLUSION

This chapter introduced AlphaZero for Continuous action space (AoC). Our method learns a continuous policy network - based on transformed Beta distributions - by minimizing a KL-divergence between the network distribution and an unnormalized density at the support points from the MCTS search. Moreover, the policy network also directs new MCTS searches by proposing new candidate child actions in the search tree. Preliminary results on the Pendulum task show that our approach does indeed learn. Future work will further explore the empirical performance of AoC. In short, AoC may be a first step in the application of iterated search and learning to problems with a continuous action space, like often encountered in robotics, navigation and self-driving cars.

6.8 APPENDIX

6.8.1 *Enforcing Action Space Bounds with Transformed Beta Distributions*

Continuous action spaces are generally bounded, i.e., we want to sample $\boldsymbol{a} \in [-c_b, c_b]^{n_a}$ for some constant $c_b \in \mathbb{R}^+$ and action space dimensionality n_a. There are various probability distributions with support on a continuous bounded interval. A well-known and flexible option is the Beta distribution, which has support in $[0, 1]$. We will therefore make our network predict the parameters of a factorized Beta distribution $\boldsymbol{u} \sim q(\boldsymbol{u})$, where each element $u_i \sim \text{Beta}(\alpha_i(\phi), \beta_i(\phi))$. Our goal is to transform the random variable $\boldsymbol{u}$ to a random variable $\boldsymbol{a}$ with support $\boldsymbol{a} \in [-c_b, c_b]^{n_a}$. A simple transformation g that achieves this goal is

$$\boldsymbol{a} = g(\boldsymbol{u}) = c_b \cdot (2\boldsymbol{u} - 1) \tag{6.13}$$

For the loss specification in the chapter, we require the (log)-density $\pi(\boldsymbol{a})$ of the transformed variable. We know from the change of variables rule that:

$$\pi(\boldsymbol{a}) = q(\boldsymbol{u}) \left| \det\left(\frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{u}}\right) \right|^{-1} \tag{6.14}$$

For the transformation $\boldsymbol{a} = g(\boldsymbol{u})$, the Jacobian $\frac{\mathrm{d}\boldsymbol{a}}{\mathrm{d}\boldsymbol{u}} = \mathrm{diag}(2c_b)$ is a diagonal matrix. Therefore, we can derive a simple expression for the (log-)likelihood of $\boldsymbol{a}$:

$$\pi(\boldsymbol{a}) = q(\boldsymbol{u}) \cdot (2c_b)^{-n_a}, \quad \text{and} \quad \log \pi(\boldsymbol{a}) = \log q(\boldsymbol{u}). - n_a \cdot \log(2c_b). \tag{6.15}$$

6.8.2 *Entropy of Transformed Beta Distribution*

We know the entropy of a linear transformation of some variable from differential entropy (Michalowicz, Nichols, and Bucholtz, 2013). For a linear transformation $\boldsymbol{M}\boldsymbol{u} + \boldsymbol{l}$, with matrix $\boldsymbol{M}$ and vector $\boldsymbol{l}$, we have

$$H(\boldsymbol{M}\boldsymbol{u} + \boldsymbol{l}) = H(\boldsymbol{u}) + \log |\det(\boldsymbol{M})| \tag{6.16}$$

For our transformation $g(\boldsymbol{u})$ (Eq. 6.13), the second term of this equation equals $n_a \log(2c_b)$. Since this term does not depend on ϕ, and therefore does not contribute any gradients, we will simply ignore it. The entropy of the Beta distribution $q(\boldsymbol{u})$ can be computed analytically (Michalowicz, Nichols, and Bucholtz (2013), p.63).

7

THINK TOO FAST NOR TOO SLOW: THE COMPUTATIONAL TRADE-OFF BETWEEN PLANNING AND REINFORCEMENT LEARNING[1]

ABSTRACT

Multi-step approximate real-time dynamic programming, a recently successful algorithm class of which AlphaZero (Silver et al., 2018) is an example, nests planning within a learning loop. However, the combination of planning and learning introduces a new question: how should we balance time spend on planning, learning and acting? The importance of this trade-off has not been explicitly studied before. We show that it is actually of key importance, with computational results indicating that we should neither plan too long nor too short. Conceptually, we identify a new spectrum of planning-learning algorithms which ranges from exhaustive search (long planning) to model-free RL (no planning), with optimal performance achieved midway.

7.1 INTRODUCTION

Multi-step approximate real-time dynamic programming (MSA-RTDP), a class of algorithms in which AlphaZero (Silver et al., 2018) also resides, was already discussed in Chapter 4. However, the iterated planning and learning paradigm als introduces a new question: how long should we plan at a given state? We already touched upon this topic in Sec. 4.4.2 and Sec. 4.6.3. In this chapter, we will further investigate this trade-off.

We hypothesize that trade-off between planning and real data collection is crucial: when we plan too extensively, we make too little progress in the domain and have less training targets for learning, while when we plan too briefly, our local decisions and training targets are likely to be less optimal. This trade-off was never present in online planning, where the budget per real step is typically as high as the application permits (in the order of milliseconds for a video game, or in the order of

1 Chapter based on: Moerland TM, Deichler A, Baldi S, Broekens J, Jonker CM. Think Too Fast Nor Too Slow: The Computational Trade-off Between Planning And Reinforcement Learning. 2020. Bridging the Gap Between AI Planning and Reinforcement Learning (PRL) workshop at the International Conference on Automated Planning and Scheduling (ICAPS).

seconds to minutes for a game of Chess (Campbell, Hoane Jr, and Hsu, 2002)). It was neither present in model-free reinforcement learning (RL), since those approaches do not have access to a dynamics model and therefore can not plan. Model-based RL, where we use observed data to approximate the dynamics model, has mostly focused on dealing with enhancing data efficiency and dealing with uncertainty in the learned models (Chua et al., 2018; Sutton, 1991). Instead, we focus on the situation with a known, perfect model without uncertainty, to fully investigate the trade-off between planning and learning once a good model is available.

We therefore study the AlphaZero algorithm (Silver et al., 2018), a successful variant of MSA-RTDP, on several known tasks. On each task, we fix the overall computational budget, but vary the planning budget per real step. Our results show that, for a fixed overall time budget, approaches with an intermediate planning budget per time-step achieve the highest final performance. This is an important empirical insight for model-based reinforcement learning and MSA-RTDP algorithms. Anthony, Tian, and Barber (2017) have previously connected planning-learning integrations to dual process theory (Evans, 1984), now popularized as 'thinking fast and slow' (Kahneman, 2011). In this analogy, planning is 'thinking slow', while learned approximations are 'thinking fast'. Adopting this nomenclature, a short summary of our results could be: 'think too fast nor too slow'.

The remainder of this chapter is organized as follows. Section 7.2 introduces the algorithm class of interest, multi-step approximate real-time dynamic programming. Section 7.3 and 7.4 detail methodology and results, respectively. The final sections cover Related work (Sec. 7.5), Discussion (Sec. 7.6) and Conclusion (Sec. 7.7). Code to replicate experiments is available from `https://github.com/ratponto/tree-rl-adaptive`.

7.2 MULTI-STEP APPROXIMATE REAL-TIME DYNAMIC PROGRAMMING

We refer the reader back to Chapter 2 for an introduction of the Markov Decision process. One of the cardinal algorithms to solve a Markov Decision process is Dynamic Programming (DP) (Bellman, 1966). For example, in Q-value iteration we sweep through a state-action value table, where at each location we update $Q(s, a)$ according to:

$$Q(s,a) \leftarrow \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)} \Big[\mathcal{R}(s,a,s') + \gamma \max_{a \in \mathcal{A}} Q(s,a) \Big] \tag{7.1}$$

Dynamic programming is guaranteed to converge to the optimal policy. However, due to the curse of dimensionality, it can not be applied in high-dimensional problems.

We therefore introduce an extension of DP, multi-step approximate real-time dynamic programming (Efroni, Ghavamzadeh, and Mannor, 2019), which has recently shown impressive empirical results, for example beating humans and achieving state-of-the-art performance in the game of Go (Silver et al., 2017c), Chess and Shogi (Silver et al., 2018). MSA-RTDP is based on Dynamic Programming concepts, but adds three additional concepts:

- 'Real time' (Barto, Bradtke, and Singh, 1995) implies that we act on traces through the environment that start from some initial state $s_0 \sim p(s_0)$. This property is assumed by most RL and planning algorithms. Compared to the DP sweeps, it avoids work on states that we will never reach.
- 'Approximate' implies that we will use function approximation to store a global parametrized solution, in the form of a value $V_\theta(s)/Q_\theta(s,a)$ and/or policy function $\pi_\theta(a|s)$, where $\theta \in \Theta$ denote the parameters of the approximation. Compared to a tabular representation, approximate representations can deal with high-dimensional state spaces and benefit from generalization between similar states, although they do make approximation errors. Approximate solutions are especially popular in RL literature.
- 'Multi step' implies that for every Dynamic Programming backup, we are allowed to make a multi-step lookahead, i.e., we can *plan*.

The resulting multi-step approximate RTDP algorithm class has three key components, which are visualized in Figure 7.1:

1. **Plan**: At every state s_t in the trace, we get to expand some computational budget $\mathtt{B}$ of forward planning, which could for example be a depth-d full-breadth search (Russell and Norvig, 2016), or a more complicated planning procedure like Monte Carlo Tree Search (Browne et al., 2012). The planning procedure can use learned value/policy functions to aid planning, for example through *bootstrapping* (Sutton and Barto, 2018).
2. **Learn**: After planning, we use the output of planning (our improved knowledge about the optimal value and policy at s_t) to train our global value/policy approximation.

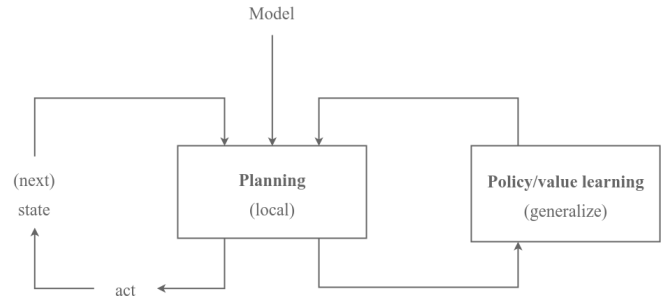


Figure 7.1: Multi-step Real-time Dynamic Programming. The three key procedures are 1) Planning, 2) Learning, and 3) Real steps (acting).

3. **Real step**: We finally use the planning output to decide which action a_t we will commit to, and make a 'real step', transitioning to a sampled next state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$. The next iteration of planning continues from s_{t+1}.

MSA-RTDP has two special cases that depend on the computation planning budget $\mathtt{B}$ per real step. On the one extreme, $\mathtt{B} \to \infty$, we completely enumerate all possible future traces, better known as *exhaustive search* (Russell and Norvig, 2016). On the other extreme, $\mathtt{B} = 0$, we do not plan at all, but directly make a real step based on the global approximations, better known as *model-free reinforcement learning* (Sutton and Barto, 2018).

Anthony, Tian, and Barber (2017) already related this approach to cognitive psychology research, in particular dual process theory (Evans, 1984; Kahneman, 2011). The global value/policy approximation, which makes fast predictions about the value of actions, can be considered a System 1 ('Thinking fast'), while explicit forward planning to improve over these fast approximations seems related to System 2 ('Thinking slow').

7.3 METHODS

For this chapter, we will follow the AlphaGo Zero (Silver et al., 2017c) variant of MSA-RTDP. AlphaGo Zero uses a variant of MCTS (Browne et al., 2012) for planning, and deep neural networks for leaning of a policy $\pi_\theta(a|s)$ and value $V_\theta(s)$ approximation. A key aspect of iterated planning-learning is their mutual influence, where planning improves the learned function, and the learned function directs new planning

iterations. We will detail both these integrations, starting with training target construction based on planning output.

To train the policy network, we normalize the action visitation counts $n(s,a)$ at the tree root state s to a probability distribution, and train on a cross-entropy loss:

$$\mathrm{L}_{\pi}(\theta) = \sum_{a} \frac{n(s,a)^{\frac{1}{\tau}}}{\sum_{b} n(s,b)^{\frac{1}{\tau}}} \log \pi_{\theta}(a|s), \tag{7.2}$$

where τ denotes a temperature parameter. For value network training, we use a target based on the reweighted value estimates at the root of the MCTS,

$$\hat{V}(s) = \sum_{a} \frac{n(s,a)}{n(s)} \bar{Q}(s,a), \tag{7.3}$$

where $\bar{Q}(s,a)$ denotes the mean pay-off of all traces through (s,a), and train on a squared error loss,

$$\mathrm{L}_{V}(\theta) = \left(V_{\theta}(s) - \hat{V}(s)\right)^2. \tag{7.4}$$

This is a slight variation of the original AlphaZero implementation, based on recent results of Efroni et al. (2018). The above equations define the planning to learning connection in Fig. 7.1.

For the reverse connection, influencing planning based on the learned functions, we i) replace the MCTS rollout by a bootstrap estimate from the value network, and ii) modify the MCTS selects step to

$$\arg\max_{a} \left[\bar{Q}(s,a) + c \cdot \pi_{\theta}(a|s) \cdot \sqrt{\frac{n(s,a)}{1+n(s)}} \right], \tag{7.5}$$

where $c \in \mathbb{R}$ is a constant that scales exploration pressure. We did not include the Dirichlet noise used in AlphaZero.

We vary the planning budget per timestep through adjustment of the number of traces per MCTS iteration, denoted by n_{MCTS}, while keeping the overall computational budget (in the form of wall clock time) fixed. We experiment with two well-known control tasks, CartPole (branching factor 2) and MountainCar (branching factor 3), available from the OpenAI Gym (Brockman et al., 2016), and with the RaceCar

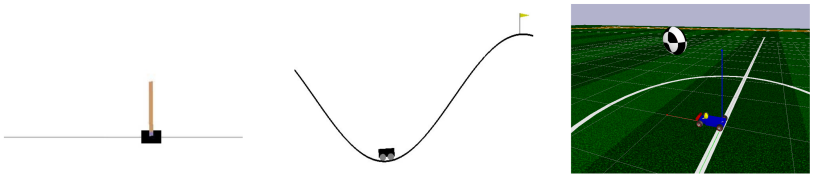


Figure 7.2: Image stills from the studied tasks. Left: CartPole, where we attempt to balance the pole. Middle: MountainCar, where we attempt to reach the top-left flag by swinging back and forth. Right: RaceCar, where we need to control a car to reach a goal, indicated by a ball.

task (branching factor 9), available in the PyBullet package (Coumans and Bai, 2016). For MountainCar, we use a reward function variant with $r = -0.005$ on every step, and $r = +1$ when the Car reaches the top of the hill. Visualizations of the tasks are shown in Figure 7.2.

The total computational budget (planning, training and acting) was fixed in advance on every environment: 500 seconds for CartPole, 150 minutes for MountainCar, and 270 minutes for RaceCar. These budgets were predetermined to allow for convergence on each domain. Therefore, long planning per timestep (higher n_{MCTS}) also implies less real steps and less new training targets over the entire training period.

HYPERPARAMETERS The effect of search budget may also interact with the setting of other hyperparameters. We chose the following approach. We quickly search for a general hyperparameter configuration that shows increasing learning curves on all domains. Crucially, the search budget was varied in this quick search, but we were unaware of its actual values, to not bias the other hyperparameter settings towards good performance on a particular search budget. We will touch upon alternative approaches in the Discussion.

We here report the fixed values for the other hyperparameters. For neural network training, we used batches of size 16 with a replay buffer of size 5e3 and learning rate of 1e-3 on all domains, optimized with ADAM optimizer (Kingma and Ba, 2014). Policy and value network shared their hidden layers, with 256 hidden nodes per layer. Since the reward scales between the task varied greatly, the c parameter (Eq. 7.5) did require adjustment per domain: for CartPole we decayed it from 0.8 to 0.05 in 500 steps, for MountainCar from 5 to 0.5 in 5000 steps, and for RaceCar from 1.0 to 0.05 in 1500 steps. All results are averaged over 3 repetitions.

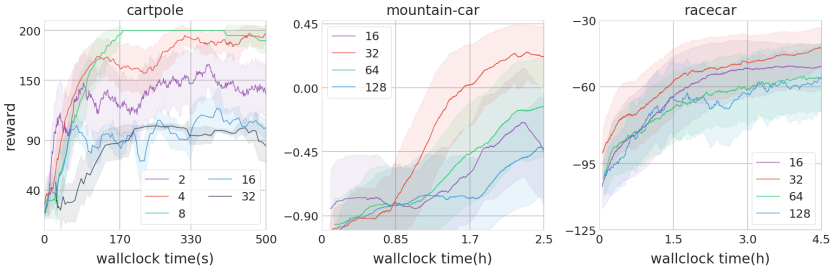


Figure 7.3: Learning curves on CartPole, MountainCar and RaceCar environments. The colour legend per plot displays the MCTS trace budget before every real step (n_{MCTS}). We see that AlphaGo Zero learns on all tasks, with best performance on CartPole, MountainCar and RaceCar achieved for budgets of, respectively, 8, 32 and 32 traces per timestep. The error bars plots plus and minus 1 standard deviation over repetitions.

7.4 RESULTS

Figure 7.3 shows learning curves for the three environments. We see that the AlphaZero algorithm manages to learn all three tasks. The largest variation in performance is seen on the CartPole task. Clearly, the most stable performance for CartPole uses $n_{\text{MCTS}} = 8$. A possible confounder is that at low MCTS budgets, the training targets derived from visit counts are highly unstable (Hamrick et al., 2020). This could partially explain why the low MCTS budgets show poor performance in CartPole.

Compared to CartPole, MountainCar has a sparser reward. We therefore require longer total budget and more traces per timestep to achieve best performance, which is attained with $n_{\text{MCTS}} = 32$. Finally, RaceCar has a larger action space than both other domains, wich requires longer training, and generally more traces per timestep. The best performance is achieved for $n_{\text{MCTS}} = 32$ traces.

The learning curves indicate that optimal performance is achieved for an intermediate search budget. To better illustrate this observation, we aggregate the average pay-offs from the last 15% of total time for every planning budget in each environment. These results are visualized in Figure 7.4. The horizontal axis now displays search budget, while the vertical axis displays mean pay-off at the end of training. For all three environments, we observe clear optimal performance for an intermediate search budget per real step.

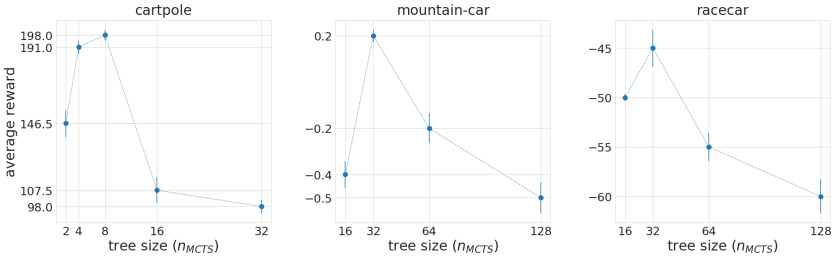
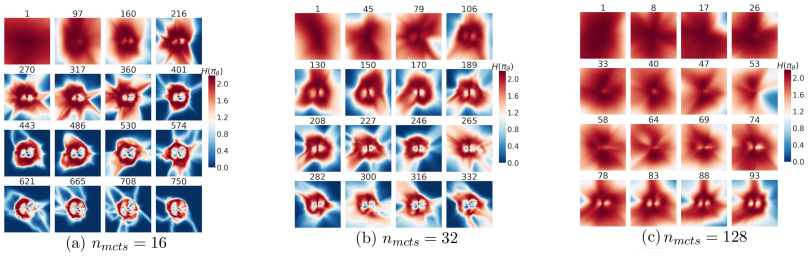


Figure 7.4: Trade-off between planning and learning. The horizontal axis shows the computational budget per MCTS search in the form of the total number of traces. The vertical axis shows the cumulative reward achieved by the specific set-up. Data based on last 15% of the learning curves in Fig. 7.3. Note that the total computation time for every repetition was fixed, i.e., higher planning budget per timestep will yield less real steps and less targets for training the neural networks. We observe a clear trade-off on all domains, with optimal results achieved for intermediate search budgets. Error bars show the standard deviation of the mean estimate (which scales as $1/sqrt(n)$ for n seeds, and are therefore smaller than those in Figure 7.3).

To further investigate what happens during training, we visualize the output of the policy network on RaceCar for different search budgets in Figure 7.5. The right, middle and left progression refer to n_{MCTS} settings of 16, 32 and 128, respectively. Each subplot shows the two-dimensional RaceCar state space, which describes the (x,y)-location of the ball in first person view. Each state in this state space is coloured according to the entropy of the policy network. Red colour implies high entropy and therefore an uncertain policy, while blue colour implies low entropy and a near converged policy. The number above each subplots indicates the episode number.

First of all, we may note that the entropy of the policy is high in the entire state space at the beginning of all three search budgets, which is to be expected. Second, we can clearly observe a difference in the number of completed episodes. Looking at the bottom-right subplot of the left ($n_{\text{MCTS}} = 16$), middle ($n_{\text{MCTS}} = 32$) and right ($n_{\text{MCTS}} = 128$) plot, we observe that we completed 750, 332 and 93 full episodes for the search budgets of 16, 32 and 128 traces per real step, respectively. Of course, a higher search budget implies that we complete less episodes.

We also attempt to qualitatively compare the convergence of the policy networks in all three scenarios. When we compare the high search budget (right) with the intermediate one (middle), we see that



(a) $n_{mcts} = 16$ (b) $n_{mcts} = 32$ (c) $n_{mcts} = 128$

Figure 7.5: Training progression of policy network on RaceCar, for a) $n = 16$ trace budget per MCTS iteration, b) $n = 32$ trace budget, and c) $n = 128$ trace budget. Each plot (a-c) visualizes a progression over training, where the number above the subplot indicates the episode number. A subplot within each plot visualizes the two-dimensional state space (x-y location of the ball in first person view), where each state is colour coded according to the entropy of the policy network at that state. High entropy (red colour) implies an uncertain policy, while low entropy (blue) implies a converged policy network. We see that the right progression ($n_{\text{MCTS}} = 128$) qualitatively seems to slow, as there are too little training targets. The left progression ($n_{\text{MCTS}} = 16$) seems to converge fast, but Fig. 7.4 shows that convergence is premature, as the achieved return is worse than the middle progression ($n_{\text{MCTS}} = 128$).

the high search budget shows a similar progression, but it progresses slower. For example, the policy network at episode 93 for $n_{\text{MCTS}} = 128$ shows similarity with the situation after episode 170 for $n_{\text{MCTS}} = 32$, with near convergence (blue) at the border of the state space, and demarcation of early convergence areas (white) in the center of state space. Although we did require less episodes to reach that situation for $n_{\text{MCTS}} = 128$, it did take more computation due to the relatively high planning effort per real step. Therefore, the high planning budget cannot benefit enough from generalization of information. The reverse situation is visible when we compare the left plot ($n_{\text{MCTS}} = 16$) with the middle plot ($n_{\text{MCTS}} = 32$). In the left plot, the policy network seems to converge faster, with a very certain policy (blue) in most of the state space at the end of the total time budget. However, if we look at the performance in Fig. 7.4, the convergence was probably premature, as we probably trained on planning targets that were too unstable. Of course, such a qualitative analysis has a subjective aspects as well. We will further interpret these observations in the discussion.

7.5 RELATED WORK

AlphaGo Zero (Silver et al., 2017c) and Alpha Zero (Silver et al., 2018), as used in this work as well, are examples of multi-step approximate real-time dynamic programming. AlphaGo Zero treats the trade-off between planning and learning as a fixed hyperparameter, where they use 1600 MCTS traces per real step in the game of Go, and 800 MCTS traces per real step for both Chess and Shogi. A very similar algorithm is Expert Iteration (ExIt) (Anthony, Tian, and Barber, 2017), which shows state-of-the-art performance in the game Hex. The authors do not report the MCTS budget per search used during training.

The earliest idea of iterated search and learning seems to date back to Samuel's checkers programme (Samuel, 1967). In later work, Carmel and Markovitch (1999) explicitly studies *lookahead-based exploration*. The authors do mention that 'it is rational for the agent to invest in computation in order to save interaction', but do not further investigate this trade-off. Chang et al. (2015) made a step towards multi-step approximate real-time dynamic programming with Locally Optimal Learning to Search (LOLS). LOLS iterates i) Monte Carlo search, which leverages the policy, and ii) policy training, which is based on the estimated values during planning. Other algorithms that update a global value approximation based on nested search are Sheppard (2002) and Veness et al. (2009).

For model-based RL with a learned model, Dyna (Sutton, 1990) already showed that planning may reduce the required number of environment steps. Moreover, Jiang et al. (2015) show that with a learned model, we should neither plan too long nor too short, since model errors will compound over long horizons. Our work extends this idea even further, by showing that the same principle already applies for *perfect* environment models, i.e., the trade-off is fundamental and does not (only) depend on the presence of a learned, uncertain model.

A theoretically study of multi-step greedy real-time dynamic programming with a known model was recently provided by Efroni, Ghavamzadeh, and Mannor (2019). One of their results shows that the *sample* complexity of multi-step greedy RTDP scales as $\Omega(1/d)$, where d denotes the depth of the lookahead, while the *computational* complexity scales as $\Omega(d)$. We directly see the trade-off appearing here, as deeper planning decreases the required number of real steps at the expense of increased computation. Our work provides an empirical investigation of this effect, indicating that the optimal, intermediate

planning budget also correlates with the dimensionality of the problem, where more complex problems require a higher budget.

Our empirical results are also partly visible in the concurrent work of Wang et al. (2019). These authors benchmark several model-based RL algorithms. They do not focus on iterated search and RL algorithms, like multi-step approximate real-time dynamic programming, but do include results of standard RL methods that train on learned dynamics models. Their results show a similar trade-off. However, their results could also be caused by the uncertainty in a learned model, which makes planning far ahead less reliable. In contrast, our work shows a more fundamental trade-off exist, even in the case of a converged/perfect model.

7.6 DISCUSSION

The computational experiments in this work clearly show a trade-off between planning, learning and acting. We identify planning budget per timestep as the major factor of importance: with a higher budget per timestep, we generate fewer training targets (and therefore spend less time on training) and make fewer real steps (complete less full episodes).

Figure 7.6 conceptually illustrates the observations from this chapter. On the left side of this plot, we find model-free RL, where the planning budget per timestep $\texttt{B} = 0$, and we only make real steps. Although model-free RL has shown impressive results (Mnih et al., 2015), it is known to be notoriously unstable, especially in combination with function approximation (Sutton and Barto, 2018). On the right side of this plot we find exhaustive search, where the computational budget per timestep $\texttt{B} \rightarrow \infty$, and we try to completely enumerate all futures from the root before choosing an action. Exhaustive search has high computational complexity that scales exponentially in the depth of the problem, and is therefore generally not a feasible approach. The problem is that it never generalizes information between states it encounters (no learning), and therefore repeats much work.

Given the above observations, the shape of Figure 7.6 may come as no surprise, as it appears to keep the best of both worlds. On the one hand, we use local planning to i) create better training targets for our global value/policy approximation, and ii) correct for local errors in these approximations by looking ahead to more clearly discriminable states. On the other hand, learning adds to pure planning the ability to

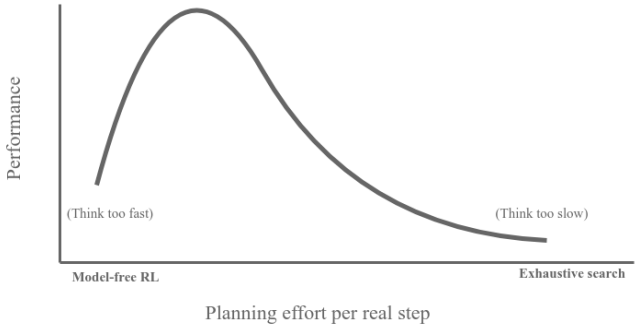


Figure 7.6: Conceptual illustration of the trade-off between planning and learning. The horizontal axis shows the computational budget of planning before every real step. On the left extreme we find model-free RL, which samples only a single transition before every step. On the far right, we find exhaustive search, which completely enumerates the search tree before executing a step. The curve illustrates the experimental results, which show a trade-off.

generalize and store global solutions in memory, which avoids repeating much work, as for example present in exhaustive search.

As mentioned in Sec. 7.3, the effect of planning budget per timestep may interact with the value of other hyperparameters. For this work we chose to quickly search for a general hyperparameter setting on all domains, while being agnostic to the search budget in that phase. There could be two alternative approaches. First, we could separately optimize all other hyperparameters for every search budget on every domain. This would squeeze out the optimal performance, but is very computationally demanding. Second, we could specify an interval for every hyperparameter with reasonable values, and test on a set of random samples from these ranges, which would test robustness to hyperparameter variation. These could be interesting extensions with slightly different messages. Nevertheless, our approach is also unbiased, shows consistent results over tasks, and complies with empirical search budget decisions in other papers, for example in AlphaGo Zero (Silver et al., 2017c) (which used 1600 MCTS traces per real step, not 1 or 10 million).

A clear direction of future work would be to adaptively adjust the planning budget per timestep in a data-driven way. Cognitive science has for long investigated how humans decide on planning duration, for example halting when a solution is 'satisficing' (a portmanteau

of satisfy and suffice) (Schwartz et al., 2002). Computational models of such data-dependent trade-offs, possibly based on the remaining uncertainty in the plan, may further improve performance of planning-learning integrations.

7.7 CONCLUSION

This chapter investigated the computational trade-off between planning and learning. Our results indicate that, in a model-based RL setting, high performance requires that the planning budget per real time-step should neither be too high nor too low. This is an important insight for the empirical application of model-based RL algorithms, as we already identified in Chapter 4. Moreover, it opens up towards future research on this trade-off, for example identifying whether the budget per time-step should be a context-dependent function of the observed data.

8

IMPROVED MONTE CARLO TREE SEARCH THROUGH SUBTREE DEPTH ESTIMATION [1]

ABSTRACT

Monte Carlo Tree Search (MCTS) efficiently balances exploration and exploitation in tree search based on count-derived uncertainty. However, these local visit counts ignore a second type of uncertainty induced by the size of the subtree below an action. We first show how, due to the lack of this second uncertainty type, MCTS may completely fail in well-known sparse exploration problems, known from the reinforcement learning community. We then introduce a new algorithm, which estimates the size of the subtree below an action, and leverages this information in the UCB formula to better direct exploration. Subsequently, we generalize these ideas by showing that loops, i.e., the repeated occurrence of (approximately) the same state in the same trace, are actually a special case of subtree depth variation. Testing on a variety of tasks shows that our algorithms increase sample efficiency, especially when the planning budget per timestep is small.

8.1 INTRODUCTION

Monte Carlo Tree Search (MCTS) (Coulom, 2006), already used in Chapters 6 and 7, is a state-of-the-art planning algorithm (Browne et al., 2012; Chaslot et al., 2008b). As already discussed in Chapter 3, MCTS shares its algorithmic space with all reinforcement learning algorithms. In this chapter, we will experimentally investigate this connection. In particular, we show how MCTS may fail at a well-known RL toy task, and identify the underlying problem. Then, we propose an improvement to MCTS based on known solutions from RL literature, which experimentally illustrates the connection between the planning and learning fields.

The strength of MCTS is the use of statistical uncertainty to balance exploration versus exploitation (Munos et al., 2014). A popular MCTS

1 Chapter based on: Moerland TM, Broekens J, Plaat A, Jonker CM. Monte Carlo Tree Search for Asymmetric Trees. 2018. Planning and Learning (PAL) Workshop @ ICML 2018.

selection rule is Upper Confidence Bounds for Trees (UCT) (Cazenave and Jouandeau, 2007; Kocsis and Szepesvári, 2006), which explores based on the Upper Confidence Bound (UCB) (Auer, Cesa-Bianchi, and Fischer, 2002) of the mean action value estimate. More recently, MCTS was also popularized in an iterated planning and learning scheme, where a low-budget planning iteration is nested in a learning loop. This approach achieved super-human performance in the games Go, Chess and Shogi (Silver et al., 2016, 2017c).

However, the UCB selection rule only uses a local statistical uncertainty estimate derived from the number of visits to an action node. Thereby, it does not take into account how many reachable states there are from a particular action, or, in other words, how large the remaining subtree below that action is *in the ground-truth MDP tree*. Note that, in this chapter, we will use the term 'remaining subtree' to refer to all nodes below an action in the true MDP tree, not those in the current MCTS tree (the MCTS tree is our partial reconstruction of the true MDP tree). When we sample a single trace from a very large remaining MDP subtree, we have much more remaining uncertainty than when we sample a trace from a very shallow MDP subtree. However, standard MCTS cannot discriminate these settings, since it does not attempt to estimate the size of remaining MDP subtree from which it is sampling. It turns out that MCTS can perform arbitrarily bad when the variation in subtree size between arms is large.

The identification of this problem originates from work in reinforcement learning on sparse exploration (Moerland, Broekens, and Jonker, 2017a; Osband et al., 2016). We empirically observed that Monte Carlo Tree Search performs really bad on common toy tasks in this field, like 'the Chain' (also used in this chapter). In this chapter, we show that a solution from the RL literature can actually be transferred to MCTS, illustrating that both fields indeed share the same algorithmic space (as extensively discussed in Chapter 3.

We propose a solution to the introduced problem through an extra back-up of *an estimate of the size of the subtree below an action*. This information is then integrated in an adapted UCB formula to better inform the exploration decision. Next, we show that *loops*, where the same state re-appears in a trace, can be seen as a special case of our framework. Our final algorithm, MCTS-T+, vastly increases performance in environments with variation in subtree depth and/or many loops, while performing at least on par to standard MCTS on environments that have less of these characteristics. Our experiments indicate that the benefits are mostly present 1) for single-player RL tasks with more

early termination and loops, and 2) for lower computational budgets, which is especially relevant in real-time search with time limitations (e.g., robotics) and in iterated search and learning paradigms with small nested searches (e.g., in AlphaGo Zero (Silver et al., 2017c)).

The remainder of this chapter is organized as follows. Section 8.2 illustrates MCTS, the problems caused by variation in subtree depth, and introduces a solution based on subtree depth estimation. Section 8.3 identifies the problem of loops, and extends the algorithm of the previous section to MCTS-T+, which naturally deals with loops. The remaining sections 8.4, 8.5, 8.6 and 8.7 present experiments, related work, discussion and conclusion, respectively. Code to replicate experiments is available from `https://github.com/tmoer/mcts-t.git`.

8.2 VARIATION IN SUBTREE SIZE

Please refer to Chapter 2 for an introduction of the Markov Decision Process problem and Section 6.2 for an introduction of PUCT (Rosin, 2011), a variant of the UCT algorithm (Browne et al., 2012; Kocsis and Szepesvári, 2006) used in AlphaGo Zero (Silver et al., 2017c) as well. In this chapter, we will modify the prior term of PUCT, which allows us to estimate the size of the true MDP tree below an action.

We now focus on a specific aspect that the MCTS formulation does not account for: variation in the size of the subtree below actions (in the select step). Imagine we have two available actions in a certain state. The first action directly leads to a terminal state, and sampling it once therefore provides much information. In contrast, the second action has a large true MDP subtree below it, and sampling it once only explores a single realization of all possible traces, with much more remaining uncertainty about the true optimal value. Now the key issue is: standard MCTS does not discriminate both cases, since it only tracks how often a node is visited, but completely ignores the size of the subtree below that action.

Variation in subtree size is widespread in many single-player RL tasks. Examples include grid worlds (Sutton and Barto, 2018), exploration/adventure games (e.g. Montezuma's Revenge (Bellemare et al., 2013)), shooting games (where in some arms we die quickly) (Kempka et al., 2016), and robotics tasks (where the robot breaks or environment terminates if we exceed certain physical limitations) (Brockman et al., 2016). In the experimental section we test on different versions of such problems.

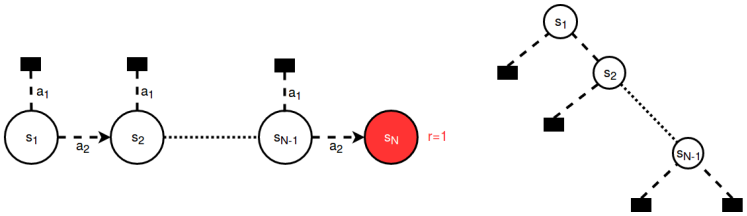


Figure 8.1: **Left:** Chain domain. At each state we have two available actions: one action terminates the episode with reward 0, the other moves one step ahead in the chain with reward 0. Only the final state terminates the episode with reward 1. **Right:** Search tree of the Chain domain.

When the subtree size below actions varies, then we can vastly gain efficiency by incorporating information about their size. For conceptual illustration, we will first focus on the Chain domain (Figure 8.1, left) (Osband, Van Roy, and Wen, 2016), a well-known task from RL exploration research. The Chain is a long, narrow path with sparse reward at the end, which gives a very asymmetric tree structure that extends much deeper in one direction (Figure 8.1, right).

The total number of terminating traces in this domain is $N + 1$ for a Chain of length N. Exhaustive search therefore solves the task with $O(N)$ time complexity. Surprisingly, MCTS actually has *exponential* time complexity, $O(2^N)$, on this task. The problem is that MCTS receives returns of 0 for both actions at the root (since the chance of sampling the full correct trace is very small, $\sim \frac{1}{2^N}$. Therefore, MCTS keeps spreading its traces at the root, and recursively the same spreading happens at deeper nodes, leading to the exponential complexity. What MCTS lacks is information about the depth of the subtree below an arm. We empirically illustrate this behaviour in Sec. 8.2.2.

8.2.1 *MCTS with Tree Uncertainty Back-up (MCTS-T)*

We now extend the MCTS algorithm to make a soft estimate of the size of the subtree below an action, which we represent as the remaining uncertainty $\sigma_\tau(s) \in [0, 1]$. For each state in the tree, we will estimate and recursively back-up σ_τ, where $\sigma_\tau(s) = 1$ indicates a completely unexplored subtree below s, while $\sigma_\tau(s) = 0$ indicates a fully enumerated subtree.

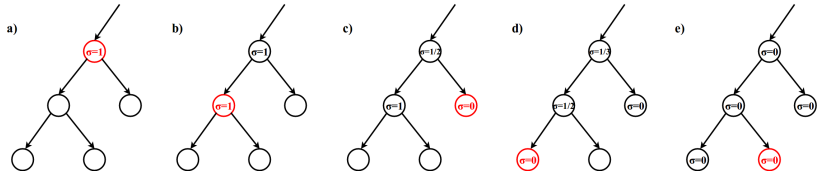


Figure 8.2: Process of σ_τ back-ups. Graphs a-e display subsequent estimates and back-ups of σ_τ. In a) and b) we arrive at a non-terminal leaf node, of which the σ_τ automatically becomes 1. In the next subtree visit (c), we encounter a terminal leaf, and the uncertainty about the subtree at the subtree root decreases to $\frac{1}{2}$. In d) we encounter another terminal leaf. Because the back-ups are on-policy, we now estimate the root uncertainty as $\sigma_\tau = \frac{(2\cdot\frac{1}{2})+(1\cdot 0)}{2+1} = \frac{1}{3}$ (Eq. 8.4). Finally, at e) we enumerated the entire sub-tree, and the tree structure uncertainty at the subtree root is reduced to 0.

We first define the $\sigma_\tau(s_L)$ of a new leaf state s_L as:

$$\sigma_\tau(s_L) = \begin{cases} 0 & \text{, if } s_L \text{ is terminal} \\ 1 & \text{, otherwise.} \end{cases} \tag{8.1}$$

We then recursively back-up σ_τ to previous states in the search tree, i.e., we update $\sigma_\tau(s_i)$ from the uncertainties of its successors $\sigma_\tau(s_{i+1})$. We could use a uniform policy for this back-up, but one of the strengths of MCTS is that it gradually starts to prefer (i.e., more strongly weigh) the outcomes of good arms. We therefore weigh the σ_τ back-ups by the empirical MCTS counts. Moreover, if an action has not been tried yet (and we therefore lack an estimate of σ_τ), then we initialize the action as if tried once and with maximum uncertainty (the most conservative estimate). Defining

$$m(s,a) = \begin{cases} n(s,a) & \text{, if } n(s,a) \geq 1 \\ 1 & \text{, otherwise,} \end{cases} \tag{8.2}$$

$$\sigma^\star_\tau(s') = \begin{cases} \sigma_\tau(s') & \text{, if } n(s,a) \geq 1 \\ 1 & \text{, otherwise,} \end{cases} \tag{8.3}$$

then the weighted σ_τ backup is

$$\sigma_\tau(s) = \frac{\sum_a m(s,a) \cdot \sigma_\tau^\star(s')}{\sum_a m(s,a)} \tag{8.4}$$

for $s' = \mathcal{T}(s,a)$ given by the deterministic environment dynamics. This back-up process is illustrated in Figure 8.2.

MODIFIED SELECT STEP Small σ_τ reduces our need to visit that subtree again for exploration, as we already (largely) know what will happen there. We therefore modify our tree policy at node s to:

$$\pi_{tree}(s) = \arg\max_a \left[Q(s,a) + c \cdot \sigma_\tau(s') \cdot \frac{\sqrt{n(s)}}{n(s,a)} \right] \tag{8.5}$$

for $s' = \mathcal{T}(s,a)$ the successor state of action a in s. The introduction of σ_τ acts as a prior on the upper confidence bound, reducing exploration pressure on those arms of which we have (largely) enumerated the subtree. Note that the prior term makes this a variant of the PUCT algorithm (Rosin, 2011).

VALUE BACK-UP The normal MCTS back-up averages the returns of all traces that passed through a node. However, the σ_τ mechanism, introduced above, puts extra exploration pressure on actions with a larger subtree below. Now imagine such a deep subtree has poor return. Then, due to σ_τ, we will still visit the action often, and this will make the state above the action look too poor. When we are overly optimistic on the forward pass, we do not want to commit to always backing up the value estimate of the explored action.

To overcome this issue, we specify a different back-up mechanism, that essentially recovers the standard MCTS back-up. On the forward pass, we track a second set of counts, $\tilde{n}(s,a)$, which are incremented *as if we acted according to the standard MCTS formula* (without σ_τ):

$$\tilde{n}(s,a) \leftarrow \tilde{n}(s,a) + \mathbf{I}\left[a = \arg\max_b Q(s,b) + c \cdot \frac{\sqrt{n(s)}}{n(s,b)}\right], \tag{8.6}$$

where $\mathbf{I}[\cdot]$ denotes the indicator function. We act according to Eq. 8.5, but on the backward pass use the $\tilde{n}(s,a)$ counts for the value back-up:

$$Q(s,a) = \frac{\sum_{a'} \tilde{n}(s',a') \cdot Q(s',a')}{\tilde{n}(s')}, \tag{8.7}$$

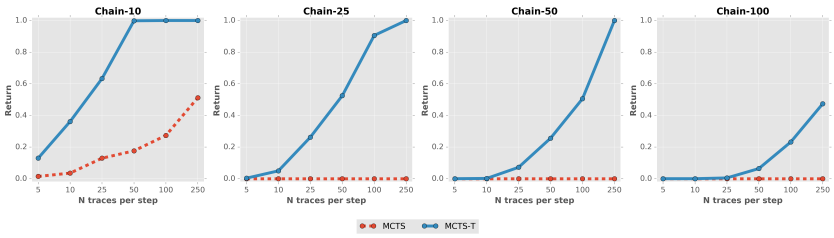


Figure 8.3: Comparison of vanilla MCTS (red) versus MCTS-T (blue) on the Chain domain of various lengths (progressing horizontally over the plots). Each plot displays computational budget per timestep (x-axis) versus average return per episode (y-axis). Results averaged over 25 episodes. We observe that MCTS-T achieves much higher returns in these domains with asymmetric termination and therefore variation in subtree depth.

for $s' = \mathcal{T}(s, a)$. This reweighs the means of all child actions according to the visit count they would have received in standard MCTS, which is the same as the standard MCTS back-up.

Finally, we do no longer want to recommend an action at the root based on the counts, so we instead recommend the action with the highest mean value at the root.

8.2.2 *Results on Chain*

Figure 8.3 shows the performance of MCTS versus MCTS-T on the Chain (Fig. 8.1). Plots progress horizontally for longer lengths of the Chain, i.e., stronger asymmetry and therefore a stronger exploration challenge. In the short Chain of length 10 (Fig. 8.3, left), we see that both algorithms do learn, although MCTS-T is already more efficient. For the deeper chains of length 25, 50 and 100 (next three plots), we see that MCTS does not learn at all any more (flat red dotted lines), even for higher budgets. This illustrates the exponential sample complexity (in the length of the Chain) that MCTS starts to suffer from. In contrast, MCTS-T does consistently learn in the longer chains as well.

8.3 LOOPS

We will next generalize the ideas about tree asymmetry to the presence of *loops* in the domain. A loop occurs when the same state appears twice

in the same trace within a single search. In such cases, it never makes sense to further expand the tree below the second appearance. As an example, imagine we need to navigate three steps to the left. If we first plan one step right, then one step back left (a loop), then it does not make sense to continue planning to the left from that point. We could better plan to the left directly from the root itself.

There is an important conceptual difference between a loop and a transposition (Plaat et al., 1996). Transpositions are ways of sharing information between states that were visited in other traces. In contrast, a loop is a property within a single search, where two nodes appear in the *same* trace (above eachother). A transposition table would share information from the first occurrence to the second, but it does not prevent us from expanding the tree in the direction of the loop. First of all, this wastes resources within the search. Potentially more problematic, the first action in the direction of the loop may at the end of the MCTS seem optimal (if all rewards in the loop are 0). Since we only execute the first action after the MCTS search, we may actually move in the wrong direction. In the earlier example, we may in the end step right, because our deeper transposition table told us that we will eventually move left again. This is of course problematic, and we would prefer to eliminate these loops from the search.

Loops are especially frequent in single-player RL tasks, for example navigation tasks where we may step back and forth between two states. Note that the detection of loops does require full observability (since otherwise we do not know whether we truly observe a repeated state, or something relevant changed in the background). We will illustrate the problem of loops with a variant of the Chain where the 'wrong' action at each timestep returns the agent to state s_1 without episode termination (Figure 8.4, left). When we now unfold the search tree (Figure 8.4, right), we see that the tree is no longer asymmetric, but does have a lot of repeated appearances of state s_1. Standard MCTS cannot detect this problem, and will therefore repeatedly expand the tree in all directions.

8.3.1 *MCTS-T+: blocking loops.*

When we remove all the repeated visits of s_1, then we actually get the same tree as for the normal Chain again. This suggests that our σ_τ mechanism has a close relation to the appearance of loops as well. A natural solution is to detect duplicate states s° in a trace, and then set

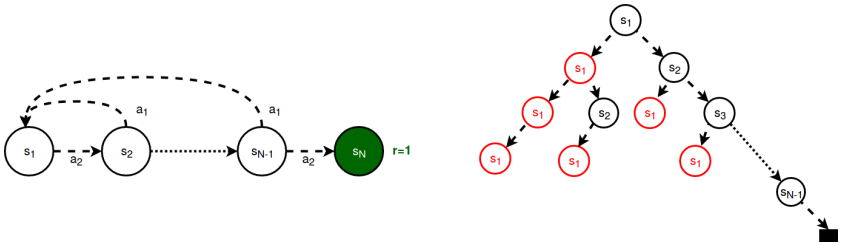


Figure 8.4: **Left:** Chain domain with loops/cycles. **Right:** Search tree of the cyclic Chain domain. Red nodes indicate a loop, i.e., the repetition of a state which already occurred in the trace above it.

$\sigma_\tau(s^\circ) = 0$. Thereby, we completely remove the exploration pressure from this arm, i.e., treat the looped state as if it has an empty subtree.

The value/roll-out estimate of the duplicate state $\mathrm{R}(s^\circ)$ depends on the sum of reward in the loop $\mathrm{S}^\circ = \sum_{s,a \in g} r(s,a)$, where $g = \{s^\circ, .., s^\circ\}$ specifies the subset of the trace containing the loop. For infinite time-horizon problems with $\gamma = 1$ (whose return is not guaranteed to be finite itself), we could theoretically repeat the loop forever, and therefore:

$$R(s^\circ) = \begin{cases} \infty & , \text{if } \mathrm{S}^\circ \geq 0 \\ -\infty & , \text{if } \mathrm{S}^\circ \leq 0 \\ 0 & , \text{if } \mathrm{S}^\circ = 0 \end{cases} \tag{8.8}$$

$R(s^\circ)$ is the return estimate that we will actually back-up from state $^\circ$. For finite horizon problems, or problems with $\gamma < 1$, we may approximate the value of the loop based on the number of remaining steps and the discount parameter. However, note that most frequently loops with a net positive or negative return are a domain artifact, as the solution of a (real-world) sequential decision making task is seldom to repeat the same action loop forever.

In larger state spaces, exact loops are rare. We therefore check for approximate loops, where the looped state is very similar to a state above. We mark a new leaf state s_L as looped when for any state s_i above it, $L > i \geq 0$, the L2-norm with the new expanded state is below a tunable threshold $\eta \in \mathbb{R}$:

$$\|s_L - s_i\|_2 < \eta. \tag{8.9}$$

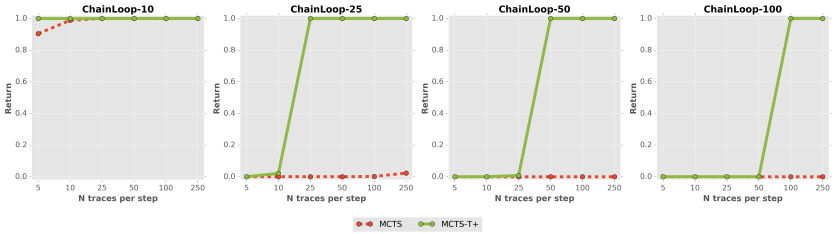


Figure 8.5: Comparison of MCTS (red) versus MCTS-T+ (green). MCTS-T+ uses tree uncertainty and loop blocking. Chain length progresses horizontally over the plots. Results averaged over 25 episodes. We observe that MCTS-T+ strongly outperform MCTS, which hardly incurs any reward on longer chains.

Once a loop is detected, we set $\sigma_\tau = 0$, and apply all methodology from the previous section.

Note that a simpler solution to blocking loops could be to completely remove the parent action of a looped state from the tree. We present the above introduction to i) be robust against situations where the loop is relevant, and ii) to conceptually show what a loop implies: a state with an empty subtree below it ($\sigma_\tau = 0$).

8.3.2 *Results on Chain with loops*

We illustrate the performance of MCTS-T+ on the Chain with loops (Figure 8.4). The results are shown in Figure 8.5. We observe a similar pattern as in the previous section, where MCTS only (partially) solves the shorter chains, but does not solve the longer chains at all. In contrast, MCTS-T+ does efficiently solve the longer chains as well. Note that MCTS-T (without loop detection) does not solve this problem either (curves not shown), as the loops prevent any termination, and therefore all σ_τ estimates stay at 1.

8.4 EXPERIMENTS

The previous experiments, on the Chain and Chain with loops, present extreme cases of variation in subtree depth and the presence of loops. They are example cases to show the worst-case performance of MCTS in such scenarios, but are not very representative of most problems in the RL community. We therefore compare our algorithm to standard

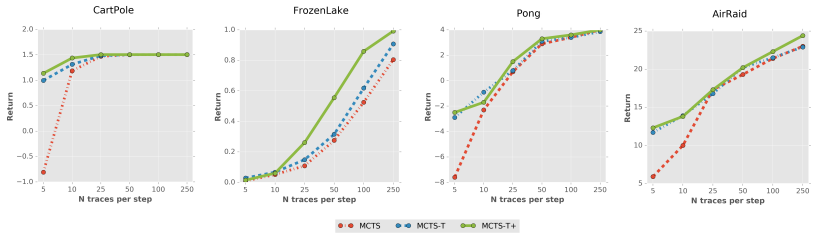


Figure 8.6: Learning curves on CartPole, FrozenLake, Pong and AirRaid. CartPole rewards are 0.005 for every timestep that the pole does not fall over, and -1 when the pole falls (episode terminates). We use the non-stochastic version of FrozenLake. The two Atari games, Pong and AirRaid, clip rewards to $[-1, 1]$. All episodes last 400 steps, with a frameskip of 3 on the Atari games. Results averaged over 25 repetitions.

MCTS on several reinforcement learning tasks from the OpenAI Gym repository (Brockman et al., 2016): CartPole, FrozenLake and the Atari games Pong and AirRaid.

These results are visualized in Figure 8.6. We see that MCTS-T and MCTS-T+ consistently perform equal to or better than MCTS. This difference seems more pronounced for smaller MCTS search budgets. This seems to make sense, since the σ_τ machinery is especially applicable when we want to squeeze as much information out of our traces as possible.

Note that the search budgets are relatively small compared to most tree search implementations. We will return to this point in the discussion. The computational overhead of MCTS-T itself is negligible (compared to the environment simulations). For MCTS-T+, loop detection does incur some cost in larger state spaces. In the worst case, on Atari, MCTS-T+ has $\sim$10% increase in computation time.

8.5 RELATED WORK

The closest related work is probably MCTS-Solver (Winands, Björnsson, and Saito, 2008), designed for two-player, zero-sum games. In MCTS-Solver, once a subtree is enumerated, the action link above it is associated with its game-theoretical value $+\infty$ (forced win) or $-\infty$ (forced loss). It then uses specific back-up mechanisms (e.g., if one child action is a win, then the parent node is a win, and if all child actions are a loss, then the parent node is a loss). Compared to MCTS-Solver,

our approach can be seen as a soft variant, where we gradually squeeze arms based on their estimated subtree size, instead of only squeezing completely once we fully enumerated the arm. Moreover, our approach is more generally applicable: it does not have any constraints on the reward functions (like win/loss), nor does it use back-up rules that are specific to two-player, zero-sum games. As such, MCTS-Solver would not be applicable to the problems studied in this chapter.

Other related work has focused on maintaining confidence bounds on the value of internal nodes, first introduced in B* (Berliner, 1981). For example, score-bounded MCTS (Cazenave and Saffidine, 2010) propagates explicit upper and lower bounds through the tree, and then prunes the tree based on alpha-beta style cuts (Knuth and Moore, 1975). This approach is only applicable to two-player games with minimax structure, while our approach is more general. Tesauro, Rajan, and Segal (2012) present a MCTS variant that propagates Bayesian uncertainty bounds. This approach is robust against variation in subtree size (not against loops), but requires priors on the confidence bounds, and will generally be quite conservative. One of the benefits of MCTS is that it gradually starts to ignore certain subtrees, without ever enumerating them, a property that is preserved in our approach.

While MCTS is a regret minimizing algorithm, a competing formulation, known as best-arm identification (Audibert and Bubeck, 2010; Kaufmann and Koolen, 2017), only cares about the final recommendation. Our approach also departs from the regret minimization objective, by putting additional exploration pressure on arms that have more remaining uncertainty. Finally, our solution also bears connections to RL exploration research papers that use the return distribution to enhance exploration (Moerland, Broekens, and Jonker, 2018b; Tang and Agrawal, 2018), which implicitly may perform a similar mechanism as described in this chapter.

8.6 DISCUSSION

This chapter introduced MCTS-T+, an MCTS extension that is robust against variation in subtree size and loops. We will briefly cover some potential criticism and future extensions of our approach.

From a games perspective, one could argue that our method is only useful in the endgame, when the search is relatively simple anyway (compared to the midgame). While this is true in two-player games, such as Go and Chess, many single-player reinforcement learning tasks,

as studied in this chapter, tend to have terminating arms right from the start (like dying in a shooting game), or many loops (like navigation tasks where we step back and forth). Our results are especially useful for the latter scenarios.

Our methods seems predominantly beneficial with relatively small search budgets per timestep, compared to the budgets typically expended for search on two-player games. We do see three important ways in which our approach is relevant. First, real-time search with a limited time budget, as for example present in robotics applications, will benefit from maximum data efficiency. Second, we have recently seen a surge of success in iterated search and learning paradigms, like AlphaGo Zero (Silver et al., 2017c), which nest a small search within a learning loop. Such approaches definitely require an effective small search. Finally, we also believe that our work is conceptually relevant in itself. It identifies a second type of uncertainty not frequently identified in MCTS, nor individually studied.

A limitation of our algorithm may occur when a sparse reward is hiding within an otherwise poorly returning subtree. In such scenarios, we risk squeezing out much exploration pressure based on initial traces that do not hit the sparse reward. However, MCTS itself suffers from the same problem, as its success also builds on the idea that the payoffs of leafs in a subtree show correlation. Although MCTS does have asymptotic guarantees (Kocsis and Szepesvári, 2006), it will generally also take very long on such sparse reward problems. This is due to the nature of such problems, which have such little structure in them, that they technically require exhaustive search.

Note that in a large domain without early termination (like the game of Go), MCTS-T will behave exactly like MCTS for a long time. As long as there is no expand step that reaches a terminal node, all σ_τ estimates remain at 1, and MCTS-T exactly reduces to MCTS. This gives the algorithm a sense of robustness: it exploits variation in subtree depth when possible, but otherwise automatically reduces to standard MCTS.

There are several directions for future work. First, the approach could be generalized to deal with stochastic and partially observable environments. Another direction would be to generalize information about σ_τ, for example by training a neural network that predicts this quantity. This would allow us to plug-in more informed estimates of σ_τ obtained from previous episodes. Finally, the σ_τ mechanism may also suggest when a search can be stopped (e.g., all $\sigma_\tau \to 0$ at the root). Time management for MCTS has been studied before, for example by Huang, Coulom, and Lin (2010).

8.7 CONCLUSION

This chapter introduced MCTS-T+, an extension to vanilla MCTS that estimates the depth of subtrees below actions, uses these to better target exploration, and uses the same mechanism to deal with loops in the search. Empirical results indicate that MCTS-T+ performs on par or better than standard MCTS on several illustratory tasks and OpenAI Gym experiments, especially for smaller planning budgets. The method is simple to implement, has negligible computational overhead, and, in the absence of termination, stays equal to standard MCTS. It can be useful in single-player RL tasks with frequent termination and loops, real-time planning with limited time budgets, and iterated search and learning paradigms with small nested searches. Together, the chapter also provides a conceptual introduction of a type of uncertainty that standard MCTS does not account for.

Part IV

INTEGRATION

9 DISCUSSION

This chapter will provide a broader perspective on the material presented in this thesis. We will first summarize our findings on the two main research questions, as posed in the Introduction. Afterwards, we will zoom out and take a broader perspective on the research field and PhD project as a whole. We will also shortly comment on two specific topics: the computational demands in AI research, and the relation between AI and psychology research. At last, we also present a discussion of potential future work in the planning-learning field.

9.1 ANSWERS TO RESEARCH QUESTIONS

We will separately summarize the answer to each research question:

RESEARCH QUESTION 1: HOW ARE PLANNING AND LEARNING RELATED? The framework for planning and learning (FRAP), as presented in Chapter 3, shows that planning and learning actually share the same algorithmic space, and are therefore fundamentally related. Previous work always presented planning and learning as separate fields, which may be complementary, but never identified that they actually do exactly the same thing. FRAP disentangles any planning or learning algorithm into six key dimensions:

1. *Computational effort*: which set of states do we intend to learn a solution?
2. *Trial selection*: how do we select the state-action pair for the next trial? (which includes exploration-exploitation balancing)
3. *Cumulative reward estimation*: how do we estimate the cumulative reward for the subtree that remains after the trial?
4. *Back-ups*: how do we back-up the information about the trail and the cumulative reward to the start state of the trial?
5. *Representation*: how do we store the solution of the MDP?

6. *Update*: how do we use the back-up estimate to update the solution?

Each dimension has a variety of subdimensions, as summarized in Chapter 3. The key message of the framework was displayed in Table 3.3, which displayed a variety of well-known planning, model-free RL and model-based RL algorithms along the dimensions of our framework. The main observation of the table is that it reads like a patchwork. On most of the dimensions, the choices vary within and over the planning, model-free and model-based RL categories. This illustrates that both approaches share the same decisions and methods, and are really two sides of the same (algorithmic) coin.

On one of the dimensions of the framework we do observe a more consistent difference between both fields. Learning approaches tend to use permanent, global solution representations, typically stored with function approximation. Planning methods have put greater emphasis on local, transient representations, typically stored in tabular/atomic form. This difference in focus may be explained by the different type of access to the environment in both fields. Since RL methods never know when they will be able to return to a state, they are almost forced to store a global solution. Global solutions also require function approximation in larger problems. In contrast, planning methods typically replan from the same state multiple times, for which tabular methods allow for good separation between the available actions.

It may seem surprising that planning and learning have such similar dimensions, since they do have a different assumption about the type of access to the environment. RL methods do need to move forward, which is a limitation *in the order in which they can make trials*. However, it is important to realize that over multiple episodes RL algorithms will eventually get back to similar states, and then face exactly the same problem as planning methods: which trial should I select next, given information obtained from previous back-ups in (approximately) similar states. In this perspective, running 100 episodes of model-free RL is actually very similar to running a MCTS search from the root with 100 traces. We only make different decisions on how to select, back-up and store the solution.

The framework identifies the shared algorithmic dimensions, but intentionally does not argue which choices on each dimension are best. Although there are definitely general recommendations, the choice of method is mostly problem dependent. For example, some successful algorithms, like A* (Hart, Nilsson, and Raphael, 1968), Q-learning

(Watkins and Dayan, 1992) and DQN (Mnih et al., 2015), make max back-ups over the actions (off-policy), while other successful algorithms, like MCTS (Browne et al., 2012), use averaging back-ups (on-policy). Many approaches use step-wise, value-based exploration, like UCB-variants (Auer, 2002), while other problems, like sparse reward domains, do seem to require different approaches like frontier exploration and intrinsic motivation (Ecoffet et al., 2019). While function approximation representation has been crucial for scaling RL to more complex, high-dimensional problems (Mnih et al., 2015), it was actually a combination of function approximation and local tabular representation that recently set state-of-the-art results in Go, Chess and Shogi (Silver et al., 2018, 2017c). We will further discuss which methods we believe are important for further advancement of intelligent sequential decision-making in the future work section.

The framework of Chapter 3 also has its standalone value in the individual research fields of planning and reinforcement learning. In other words, we could just as well use it as a framework for reinforcement learning (FR) or a framework for planning (FP). Especially in the reinforcement learning literature, there lacks a systematic view on the way to categorize algorithms. On a high-level, researchers usually indicated whether they use a value-based, policy search or actor-critic method, but this really only specifies the type of representation that is used to store the solution. The framework illustrates the broad set of choices that any RL or planning algorithm has to make, and that all contribute to its final performance.

Finally, Chapter 8 illustrated the commonalities between both fields, by designing a new method in one field (planning) by taking inspiration from the other field (reinforcement learning). Indeed, some ideas have received more interest in one community, while being mostly ignored in the other. For example, RL exploration research has frequently focused on tasks with sparse rewards. We showed that a popular planning algorithm, MCTS, can actually be highly inefficient in such problems. Our extension of MCTS, called MCTS-T+, overcomes this problem by tracking an additional form of uncertainty, derived from the depth of a subtree below an action. This is an example of cross-breeding on dimension 2 (trial selection) of our framework, and served as an empirical illustration of the commonalities between planning and learning.

RESEARCH QUESTION 2: HOW CAN PLANNING AND LEARNING BE COMBINED? We conceptually discussed the combination of planning and learning in Chapter 4. It discussed: i) methods for model learning,

ii) methods to combine planning and learning, and iii) benefits of such combinations. We shortly summarize our main findings on each of these aspects below, and also mention which empirical chapter illustrated these ideas.

While model learning is technically a supervised learning problem, MDP transition dynamics do pose their own specific approximation challenges. In particular, we discussed stochasticity, uncertainty, partial observability, non-stationarity, multi-step prediction, state abstraction and temporal abstraction as seven key challenges, which are all more or less crucial to learn a model that is useful for planning. We experimentally studied one of these challenges in Chapter 5, where we presented a new neural network training method based on variational inference to deal with stochastic environments.

The second part of Chapter 4 systematically structured the ways to combine planning and learning. We identify four main dimensions, each with several subquestions, that together specify how a planning cycle can be integrated in a larger learning loop. These four questions are:

1. At which state do we start planning?
2. How much planning budget do we allocate for planning and real data collection?
3. How do we plan?
4. How do we integrate planning in the learning and acting loop?

Again, we compared a variety of model-based RL papers along these dimensions of planning-learning integration, in Table 4.3. This systematically structures the way to think about planning-learning integration, and also helps identify novel combinations of both fields. As an empirical illustration of this idea, Chapter 6 studied a new planning-learning combination, extending the well-known AlphaGo Zero algorithm (Silver et al., 2017c) to deal with continuous action spaces. This is a novel form of question 4 from above.

Finally, Chapter 4 also structurally discussed the potential benefits of model-based RL. These benefits include enhanced data efficiency, targeted exploration, improved training stability, ability to transfer to different tasks, explainability and safety. In the empirical work section, we specifically zoom in on the third benefit (improved training stability), which has only recently surfaced. We hypothesize that planning

and learning actually provide mutual benefit, since local tabular representations (planning) may locally correct errors in global function approximation representation (learning) of the solution. We thereby identify a key trade-off: for how long should we plan in between every real action? Chapter 7 studies this trade-off, showing that optimal performance requires us to neither plan too short, nor too long. There are clear possibilities for future work here, which we will discuss in a later section of this chapter.

In short, this thesis covered various aspects of the possible integrations of planning and learning, as first shown in Fig. 1.1 in the Introduction. We reproduce this scheme in Fig. 9.1, but this time indicating the specific connections that were treated in each chapter. Chapter 3 started of by comparing the planning, model-free RL and model-based RL cycles, showing that inside they face the same algorithmic space. Next, Chapter 4 (which was actually the basis for the scheme) zoomed out and surveyed all possible connections in the scheme. In the second half of the thesis, we experimentally investigated specific elements of planning learning integration. Chapter 5 discussed a new method for model learning (arrow g), Chapter 6 discussed a novel form of iterated planning and learning in continuous action spaces (arrows b and c), Chapter 7 studied the trade-off between planning and acting (arrows a and d), and Chapter 8 investigated a novel planning method (arrow a) inspired by work from the reinforcement learning community.

Altogether, we have seen — on a conceptual level — how planning and learning are fundamentally related through the same underlying algorithmic space, and how they may be combined through a variety of model learning and planning-learning integration decisions. Moreover, the experimental part showed how their close relation may lead to 1) new algorithms in each field based on inspiration from the other, and 2) new combinations of planning and learning.

9.2 BIGGER PICTURE

The previous section discussed the technical answers to our research questions. We will also attempt to put our work in a broader perspective, zooming out over the entire research field, and over the PhD project.

- *What is the impact of our work on the planning-learning field?*:
  The combination of planning and learning promises to be an important research field in AI, both because of the strong empirical successes in AI (Deisenroth and Rasmussen, 2011; Levine

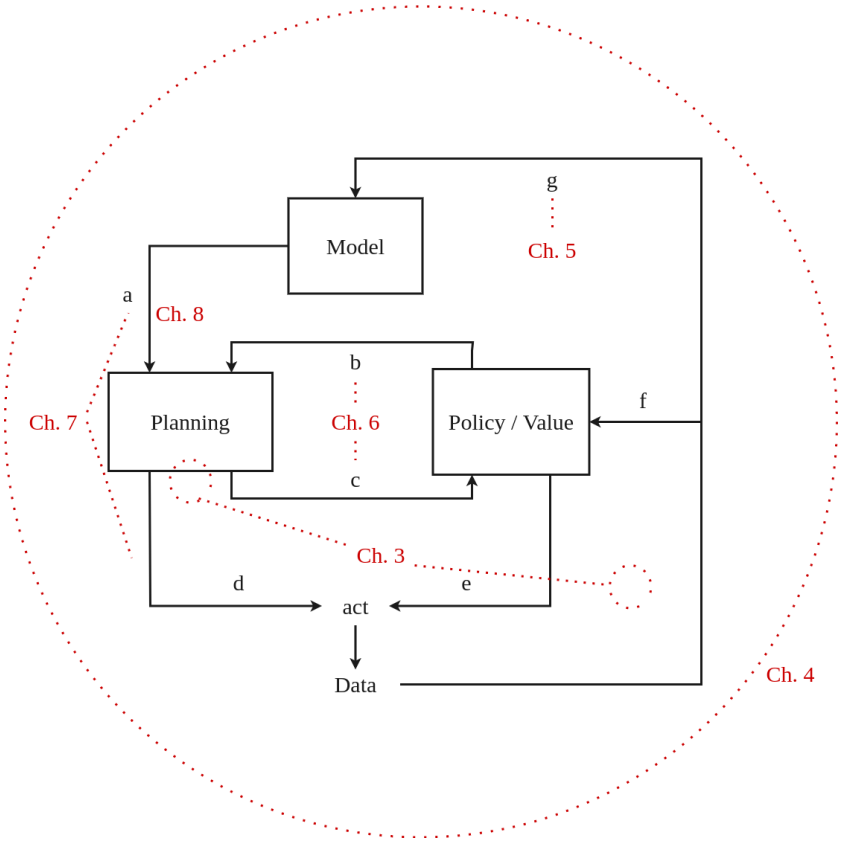


Figure 9.1: Schematic overview of the topic of each thesis chapter. The scheme (in black) shows the possible algorithmic connections between planning and learning, as already presented in Fig. 1.1 in the Introduction of this thesis. The content chapters of this thesis are shown in red, each pointing to the specific connection(s) in this scheme which are the focus of that chapter (red dashed lines).

and Koltun, 2013; Silver et al., 2017c), and because planning and learning are both integral parts of decision-making in humans (Hamrick, 2019). Like the entire machine learning community, the field is gradually attracting more researchers.

We believe the most important contribution of this book for the field is the conceptual structuring of Chapters 3 and 4. Despite the high research interest in both fields, we still lacked an integrated view on both fields. Chapter 3 should make clear that both really share the same algorithmic space. This may help to better understand each other's approaches and focus, increase communication between both fields, and as such allow for more crossbreeding. It also provides a more holistic view on sequential decision making, which is a key setting for artificial intelligence. This may benefit students that enter the field, but researchers as well. We hope that, in the future, researchers will not mention they "use a policy gradient algorithm", but realize that, according to the FRAP dimensions, there were many more relevant decisions involved.

Second, there was no systematic view on the way in which planning and learning can be combined. Therefore, Chapter 4 did systematically structure the possible combinations of planning and learning. This should be of value to the community, to better understand known algorithm, and to help design new combinations of both fields. For the latter, one may simply follow the integration dimensions and their possible choices to come up with new combinations.

The empirical work in this thesis will likely have a smaller impact on the field. The work on the trade-off between planning and learning (Chapter 7) does have several important follow-up questions, like how to determine the planning budget per timestep in a data-dependent way. We believe that the spectrum we identify in Chapter 7, ranging from model-free RL (no planning budget per timestep) to exhaustive search (infinite planning budget per timestep) should receive more attention in forthcoming years. The other empirical chapters have their isolated value in the specific subfield, but are of less importance to the field as a whole.

- *What could have been done differently in the entire PhD project?*

  The topic of this PhD project was not fixed from the start. We first made two detours, one towards the modeling of emotion in

reinforcement learning agents (Moerland, Broekens, and Jonker, 2016, 2018a), and one towards exploration methods in reinforcement learning (Moerland, Broekens, and Jonker, 2017a, 2018b). Although these topics did not end up in the final thesis, we do not regret these detours. In both these other research directions, we found out that model-based learning was a key underlying technique. It shows us that model-based learning is crucial to all sequential decision making, and one of the topics that deserves most attention in this field.

If we had known the eventual PhD topic from the start, then we would have likely first written the framework and survey papers. However, we doubt whether such an initial effort at the beginning of the PhD project would have succeeded. These research fields are large, literature is extensive, and terminology differs between the fields. Thereby, it is hard to obtain an integrated view on these fields while reading up. We believe we first needed the experimental work to write the conceptual sections.

We could have saved much time on the experimental work. During the years in which this PhD research was conducted, the reinforcement learning field dramatically changed. Deep reinforcement learning was invented roughly around 2015 (Mnih et al., 2015), a year before the start of this PhD project. With this development, and the large increase of company-based publications in the field, the requirements for submissions also changed. Suddenly, papers were required to contain experimental work sections with high-dimensional applications like Atari 2600 games. We probably spend too much time running many experiments in a range of high-dimensional problems, which is not really feasible with university resources. We will further discuss this topic in the next section, but would advise any future PhD student to always first test their ideas in a small problem. Only when your idea works in a toy environment, you want to advance to a more complex environment.

9.3 COMPUTATIONAL DEMANDS IN AI RESEARCH

A general critique on the experimental work in this thesis could regard the dimensionality of the considered problems. Mostly, we focused on relatively smaller problems, with state spaces that do not exceed 10-20 state variables, and action spaces up to 20 discrete actions, or a few

continuous dimensions. Up to a few years ago, nearly all reinforcement learning dissertations studied problems of this dimensionality, but the growth of deep reinforcement learning (Arulkumaran et al., 2017) has strongly changed the requirements and expectations in machine learning.

There are two reasons why we studied relatively small problems. First of all, the results of small scale experiments are usually easier to design, visualize and interpret. We believe that small scale experiments have gotten somewhat undervalued in the RL community in recent years, although opinions now start to shift back. As already shortly mentioned in Chapter 4: state and action space dimensionality are only one factor of problem difficulty. A good example is the Chain domain, introduced in Chapter 8, which is small in dimensionality, but very challenging for exploration (due to the sparse reward). Another good illustration is the Atari 2600 benchmark, on which the recent state-of-the-art (Schrittwieser et al., 2019) reports, compared to human performance, a mean and median score of 4999.2% and 2041.1%, respectively. However, inspection of the individual game scores reveals a huge variation, with some games scoring over 25000% of human performance, while on other games, like Montezuma's Revenge and Venture, the agent does not achieve any reward at all, and is far from human performance. While all these games have the same dimensionality, they clearly differ in other factors of problem difficulty. As a final illustration, it is noteworthy that DQN (Mnih et al., 2015), a high impact deep RL approach, usually fails at the low-dimensional MountainCar problem (Tang and Agrawal, 2018), which was also used in Chapter 6.

The key message of the above observations is that real problems have more factors of difficulty than dimensionality only, and those other factors of difficulty may well be studied in lower-dimensional examples. The benefit of lower dimensional test environments is that they allow for fast testing loops, better interpretable results, and keep the field open to a wider range of researchers with different computational resources. We are not the only ones to note this issue. Recently, Osband et al. (2019) released *b-suite*, a benchmarking suite for RL which contains a range of low dimensional tasks that each measure the robustness of an algorithm against a certain factor of difficulty, like sparse exploration, partial observability, dimensionality, stochasticity, etc. We believe that proper identification of the factors of problem difficulty is an important aspect to better compare RL algorithms in the future.

The second reason we use relatively lower-dimensional problems are of course computational limitations, since university resources can-

not match company ones. While model-free RL experiments can already be computationally expensive, model-based RL work is usually even worse. For example, we extensively experimented with iterated planning-learning ideas in the research line of AlphaGo Zero (Silver et al., 2017c). AlphaGo Zero itself trained on 4.9 million games of $\sim$400 real steps, where each real step is preceded by a MCTS search with 1600 traces. In total, this equals roughly $4.9e^6 \cdot 10^6 \cdot 1600 \cdot 400 = 3.1 \cdot 10^{12}$, or 3.1 *trillion*, environment calls. One could of course consider Go an outlier in problem difficulty, but — on the other hand — many real-world problems that we have not solved yet may be even more difficult than Go. In any case, model-based RL experiments are in general very computationally demanding.

The required number of samples is however not the only aspect that determines the run time of our algorithm. Before every sample, we need to make a network pass to predict a policy distribution and a value for bootstrapping. Although these passes can be aggregated into batches to increase performance, they still vastly increase training time. Moreover, we need to perform MCTS operations and eventually train our networks, which is computationally demanding itself. Compared to deep learning in the supervised setting, the computational requirements of reinforcement learning, and especially of planning-reinforcement learning integrations, are much higher. Finally, planning-learning integrations have all the network hyperparameters involved in supervised learning, plus a range of hyperparameters specific for RL, like exploration constants, plus a range of hyperparameters specific to model-based RL, like the planning budget per timestep. Due to the many hyperparameters, we usually require many runs before we obtain a stable result, and this easily, depending on the problem type, scales the computational requirements by a factor $\sim$100x or more. In short, both the data requirement for a single run, as well as the sensitivity to hyperparameter settings (repeating the single runs) of model-based RL are worrying. At the least, both are limiting the access of many (university-based) researchers to study more complex problems.

To conclude, note that we do not argue to only study grid worlds, nor to rebuild a new environment for every paper, from both of which RL has suffered for long. Benchmarking and testing on the same environment is important. Moreover, the eventual application of these methods in the real world will require heavy machinery and large amounts of data. But, for the reasons discussed above, conceptual development and testing is equally valid (and probably better interpretable) on small

scale problems. This notion also seems to resurface in the broader RL community now (Osband et al., 2019).

9.4 RELATION TO PSYCHOLOGY RESEARCH

While related work from the planning and reinforcement learning communities were already extensively covered in Chapter 4, we do want do draw some further connections to research from psychology. Psychologist have for long investigated human decision making (Morris and Ward, 2004; Simon, 1944). Early models of human decision making mostly focused on *rational choice theory* (Edwards, 1954), of which utility theory (Stigler, 1950) is a prime example. These theories assume humans have the computational skill to evaluate all possible courses of action and select the one which is best according to some preference scale.

Simon (1957) provided major criticism on these theories, arguing that humans are in practice fundamentally limited in both their access to information and their computational capacities. They instead argue that humans use a form of *bounded rationality*, where they search for a solution that is *satisficing* (a portmanteau of satisfy and suffice) (Newell, Simon, et al., 1972). In other words, they search for a solution until some acceptance threshold is reached. Later work found that humans differ in the way they determine this threshold. For example, Schwartz et al. (2002) separates 'maximizers' and 'satisficers', where maximizers tend to search more exhaustively for a solution, while satisficers more quickly accept a reasonable solution.

Cognitive scientists have theorized about dual competing processes in human decision making as well (Evans, 1984). These ideas have been popularized by Kahneman (2011) as 'thinking fast and slow'. These ideas exhibit qualitative similarity to the trade-off between model-free and model-based reasoning in AlphaZero, which we studied in Chapter 7. It would be interesting to further explore these connections in future work.

We also want to argue for the importance of the reverse relation, i.e., the use of computational studies to better understand human psychology. Note that we are far from the first to come up with this idea, see, e.g., Boden (1988), Sun (2008) and Hamrick (2019). In the context of reinforcement learning, most work has focused on replication of results in animal learning (Holroyd and Coles, 2002; Niv, 2009). In some sense, the entire RL field is a computational study of the theory on instrumental conditioning, where the computational study outgrew

the original field in some directions. We believe that the requirement to implement a theory is the best way to identify its gaps and loose ends. As the expression says: 'the proof of the pudding is in the eating'. An initiative like *Psychlab* (Leibo et al., 2018), a RL test environment specifically designed to replicate a variety of psychological experiments, is an interesting effort in this direction from the machine learning community.

Finally, sequential decision-making research may help unveil the fundamental roots of intelligence in optimization. A frequently heard critique on deep (reinforcement) learning is that it does not help us understand anything about intelligence because we cannot interpret the millions of parameters. However, the human brain has approximately 86 billion neurons, an neither psychology, nor neuroscience, nor computational science, will ever comprehensively explain those interactions at a neuronal level. The required system for our intelligent behaviour simply seems too large. Instead, computational science shows how a combination of useful model specification and optimization can produce intelligent behaviour, where optimization of course has firm biological roots in Darwinian evolution (Darwin, 1859). The ability to show from which concepts AI can develop may well be our best bet at 'understanding' how we ended up with our current cognitive abilities.

9.5 FUTURE WORK IN PLANNING AND LEARNING

The research chapters in this book already included their own future work sections. Here, we will zoom out and identify promising directions for the planning-learning fields as a whole. These topics are summarized in Table 9.1.

NOVEL INTEGRATIONS OF PLANNING AND LEARNING First of all, there is much remaining work on novel integrations of planning and learning, of which we have shown one example in Chapter 6. In particular, we believe that multi-step approximate real-time dynamic programming (MSA-RTDP) (Efroni, Ghavamzadeh, and Mannor, 2019), as studied in Chapters 6 and 7 as well, deserves more attention. In MSA-RTDP we use a multi-step lookahead to estimate new targets for a value (or policy) approximation, and subsequently use these approximations to influence new planning iterations. The key question is: how should we specify both of these connections?

Table 9.1: Directions of future work in the planning-learning field (as discussed throughout Section 9.5). MSA-RTDP = multi-step approximate real-time dynamic programming, MBRL = model-based reinforcement learning, IM = intrinsic motivation.

Topic	**Main questions**
1. Planning-learning combinations	- In what ways can planning output influence learning, and vice versa? - In particular: in the context of MSA-RTDP
2. Trade-off (plan-learn-act)	- When and for how long should we plan (data dependent)? - For how long should we train (until convergence, with uncertainty)?
3. Asymptotic performance	- How may we obtain better uncertainty estimates on the model? - Can MBRL with learned model match asymptotic model-free RL performance?
4. Implicit model-based RL	- Will end-to-end optimization of (parts of) the model-based RL cycle, like value equivalent models or learning to plan, outperform explicit model-based RL approaches?
5. Hierarchical RL	- What are good endpoints (reward relevancy, coverage, bottlenecks, etc.)? - How can we use temporal abstraction to arrive at good policies?
6. Transfer	- To what extend can model transfer (with state and/or temporal abstraction) reduce sample complexity in new tasks? Should AI move more in this direction?
7. Prioritized sweeping	- Is backward planning equally important as forward planning? - How feasible is backward planning in high-dimensional problems?
8. Competence-based IM	- How can we learn good goal spaces? - How do we select the next goal from the goal space? - How do we plan back to a previously visited goal?
9. Safety	- How may we ensure safe exploration by using a model?

The current solutions to this problem are quite heuristically motivated. For example, the policy network of AlphaGo Zero is now trained on the counts at the root of the MCTS search (Silver et al., 2017c). This is a heuristic decision, since the counts do summarize some notion of action preference, but their actual value is highly dependent on the choice of other MCTS hyperparameters. We could hypothesize methods that map value estimates from a planning procedure to a policy function in different ways, or methods that only learn value functions. The latter was recently also proposed by Hamrick et al. (2020). While these methods only generalize value or policy information now, we could also attempt to generalize the uncertainty in these estimates. We could also study integration with other planning methods than MCTS, as for example done in Guided Policy Search (Levine and Abbeel, 2014). In short, we have only seen the tip of the iceberg of planning-learning integrations.

TRADE-OFF BETWEEN PLANNING, LEARNING AND ACTING Chapter 7 identified the trade-off between planning and acting as an important new research topic. It showed that, when we fix the planning budget per timestep, optimal performance requires a moderate planning budget per real environment step. However, humans seem to be more adaptive about when they start planning, and for how long they plan. Future research should investigate how similar principles may translate to RL. For example, the planning budget per timestep can be determined in a data-dependent way, instead of being treated as a fixed hyperparameter. We hypothesize that the planning budget should be relatively low at the start and end of learning. In the beginning, both the model and value/policy functions are highly uncertain, and therefore planning may not bring much benefit. Near convergence, the local value/policy estimates are close to convergence, and we likely only need some local planning when the value approximation is unsure. We expect planning is mostly beneficial halfway, when we have some idea about the dynamics, but the value and policy functions are not optimal yet.

Apart from the trade-off between planning and acting, we also need to determine how much budget to spend on learning (i.e., training of model and value approximation). Therefore, we actually need to trade-off three quantities: planning, acting and learning. Learning may involve model approximation and value/policy approximation. We will focus on the former, but similar principles apply to the latter. Ideally, we require model learning to: 1) scale computationally well

(i.e., we can train incrementally on small batches of new data), 2) assess convergence (i.e., verify when the predictions around observed data points are accurate), 3) provide reliable uncertainty estimates (i.e., assess when predictions are away from the observed data), and 4) not saturate (e.g., neural network training with ReLu non-linearities saturates, where certain network parts cannot be recovered once eliminated).

It is hard to obtain all the above criteria. We may iteratively fit Gaussian Process (GP) dynamics models, as for example used in PILCO (Deisenroth and Rasmussen, 2011). This satisfies requirements 2, 3 and 4, but does not scale well computationally. Neural network models do scale computationally well, but, especially when trained incrementally, it is hard to assess convergence, extract stable uncertainty estimates, and prevent saturation. In practice, the learning budget is typically fixed as a hyperparameter, which we want neither too low (then no learning takes place at all) nor too high (saturation and extrapolation errors on preliminary data).

However, what we really require are incremental neural network training methods with convergence checks, uncertainty estimates and prevention of saturation. These aspects are less of an issue in the supervised learning community, where we train the model once on a fixed dataset, and our new data is not dependent on our previous approximations. In contrast, RL is more unstable, since we use intermediate solutions to gather new data. When humans acquire new data, they appear to be able to quickly adjust their models, better known as *one-* or *few-shot learning* (Vinyals et al., 2016). The general idea of these approaches is usually to fix the initial representation layers, and perform the fast model adaptation and uncertainty inference in a high-level, small feature space. Thereby, these methods partially reintroduce tabular methods, which avoids the slow optimization to store new information in the entire network approximation. Similar ideas may also benefit (model-based) RL, and help adaptively tune the trade-off between planning, acting and learning.

ASYMPTOTIC PERFORMANCE OF MODEL-BASED RL WITH A LEARNED MODEL While model-based RL with a known model has shown asymptotic performance that equals or outperforms model-free RL (Levine and Koltun, 2013; Silver et al., 2017c), model-based RL with a learned model generally falls behind in long run performance[1] (Wang et al., 2019). Since model-based RL with a known model does equal

1 Although MuZero (Schrittwieser et al., 2019) recently provided a counter example.

model-free in performance, the main challenge of model-based RL with a learned model seems to be caused by the inaccuracy of the learned model. This could 1) be due to inaccurate predictions of the model (e.g., no properly estimating stochasticity, or incorporating partial observability and long-term dependencies), or 2) due to the inability to account for uncertainty. The latter explanation will probably play a big role, since 1) during planning we run the risk of moving to regions where we have not even observed data, and 2) uncertainty estimation in high-dimensional models is challenging, even in the supervised learning community (Kendall and Gal, 2017). Chua et al. (2018) made an important step in this direction, by showing that a bootstrap ensemble of neural networks can account for model uncertainty, up to the point where asymptotic performance matches model-free performance. Since uncertainty estimation in high-dimensional models is an important topic in the supervised learning community, research progress in those communities will likely have a big impact on model-based RL performance as well.

IMPLICIT MODEL-BASED RL An interesting research direction, which has surfaced in the planning-learning field in the last five years, is implicit model-based RL. This group of approaches, covered in Chapter 4, wrap (parts of) the model-based RL process in an end-to-end optimization, training it on the ability to predict an (optimal) action or value. For example, value equivalent models (Grimm et al., 2020; Schrittwieser et al., 2019) train a dynamics model on the ability to predict a future value, which is an implicit form of model learning.[2] As a second direction, we may also optimize entire planning algorithms for their ability to output the correct optimal action or value, which we call 'learning to plan' (Guez et al., 2018; Pascanu et al., 2017). Finally, both ideas may also be combined in one optimization, i.e., jointly optimize the dynamics model and the planning procedure on the ability to output an optimal value or policy (Farquhar et al., 2018; Guez et al., 2019).

There is a clear potential benefit of this approach. As we have seen in other fields of machine learning, like computer vision, human engineered features were eventually outperformed by features learned through optimization. While this is now commonly accepted for feature learning, it is not yet the common approach for algorithm design. Algorithms are usually designed explicitly, and so were the model-based RL

2 Note that the value equivalent loss can also be combined with a standard next state prediction loss, which would make it a hybrid of explicit and implicit model-based RL

algorithms in this dissertation. The implicit model-based RL approach follows the intuition that good algorithms may also be derived from optimization, i.e., we may actually optimize them against their ability to achieve what we want. The history of other research fields may predict that optimization will eventually beat human intuition and design. On the other hand, good priors will always reduce the computational complexity, and our computational budgets are of course not infinite. It is an interesting question where the sweet spot between prior knowledge and optimization will lie, both for model-based reinforcement learning in particular, as well as for artificial intelligence in general.

TEMPORAL ABSTRACTION AND HIERARCHICAL RL Temporal abstraction, i.e., hierarchical reinforcement learning (Barto and Mahadevan, 2003; Dayan and Hinton, 1993; Sutton, Precup, and Singh, 1999), will be crucial to advance sequential decision-making research. This book discussed methods to integrate planning and learning in a given MDP. Instead, in hierarchical RL we try to redefine the MDP itself, by compressing it in the temporal dimension. With good temporal abstraction, the planning depth (and depth of the credit assignment problem) will be reduced, which should make it easier to find a solution. As mentioned in Chapter 4, temporal abstraction is really a form of model learning with a higher-level action space that extends over multiple timesteps.

The main challenge in hierarchical RL is the identification of useful sub-routines, already covered in Chapter 4. We need to discover relevant end-points for the high-level actions. We cannot afford to learn a subroutine towards every possible state of the MDP, since this set (and thereby the high-level action space) becomes prohibitively large. Ideally, we want a compressed set of endpoints, in which the endpoints 1) are reward relevant and/or 2) have some crucial position in the MDP (like a bottleneck between densely connected subregions (McGovern and Barto, 2001) and/or 3) provide good coverage of the entire state space. Some researchers have argued that we can retrieve subroutines through end-to-end optimization (Bacon, Harb, and Precup, 2017), while others explicitly construct end-points from state space compression, MDP graph characteristics, or overlap in successful traces. The solution may well hide in a combination of these ideas. In any case, more research on temporal abstraction is definitely needed, since this approach could vastly scale the range of applications of (model-based) RL.

TRANSFER LEARNING Another clear direction of future work for planning-learning integrations is transfer learning (Taylor and Stone, 2009). We do not even need to come up with specific transfer methods, as we can directly transfer transition and reward functions, as well as learned state and temporal abstraction, to new tasks, as long as the input space still has the same dimensionality. Of course, in humans the sensory observation space is always the same (vision, auditory information, etc.). However, in most RL tasks we always learn from scratch. This is a computationally heavy and difficult task, incomparable to the prior representations and models humans already have when they face a new task. Therefore, it makes sense that RL algorithms are typically data hungry, even on small tasks, and have trouble with one-shot learning.

In RL, we should therefore aim to make the observation space over tasks more uniform, for example based on generic visual and auditory input. This way, we can re-use more information over tasks. Current RL research spends (too) much time on learning useful representations in each new task. In the computer vision community, there are many model libraries that allow one to re-use parts of networks trained in previous research (Kornblith, Shlens, and Le, 2019). Although this idea has already appeared in RL as well (Such et al., 2018), it is not a common approach yet. We believe that the transfer setting may reveal the true potential of model-based RL. On a single task, model-based RL tries to squeeze all information out of a small amount of data, which is by definition challenging. In contrast, the transfer setting allows us to re-use a learned model, possibly with good state and/or temporal abstraction, which may strongly reduce the sample requirements and training time in new tasks.

PRIORITIZED SWEEPING WITH FUNCTION APPROXIMATION As mentioned in Chapters 3 and 4, prioritized sweeping (Moore and Atkeson, 1993) is an example of a powerful idea from early RL literature. The key idea is to propagate relevant new information towards all possible precursors of a state. This idea is intuitive in everyday life. If you burn your hand on the thee kettle as a child, you will directly realize that this negative state can also be reached by approaching the kettle from a different direction. Backward planning has also been extensively studied in psychological literature (Holmberg and Robèrt, 2000; Rollier and Turner, 1994; Wiese, Buehler, and Griffin, 2016).

While a tabular dynamics model is trivial to revert, prioritized sweeping in high-dimensional problems requires explicit estimation of a

backwards model. Agostinelli et al. (2019), Edwards, Downs, and Davidson (2018), and Corneil, Gerstner, and Brea (2018) provide example approaches with neural network approximation. Ideally, we would design function approximation models that learn the correlations structure between current state s, action a and next state s', and which can be queried for any third variable conditioned on two others. In any case, backward planning might be just as important as forward planning, and it should be a promising direction for future RL research.

COMPETENCE-BASED INTRINSIC MOTIVATION Exploration has always been a central topic in sequential decision making. We believe competence-based intrinsic motivation (Oudeyer, Kaplan, and Hafner, 2007; Oudeyer, Kaplan, et al., 2008), already discussed in Chapter 4, is the most promising direction in exploration research. Most exploration research has focused on step-wise methods, which locally perturb the chosen action based on random noise or value uncertainty. However, humans incorporate a different exploration mechanism as well (Oudeyer and Kaplan, 2009), where they first fix a new goal that is close to their current abilities, exploit previous information to get close to the specific goal, and only then explore to reach it. In this approach, we do not explore at every step in the trace, but only at the end (when we reach the target state for exploration).

Compared to standard RL approaches, these methods have a 'goal space' which consists of their current competences, i.e., the states they already managed to reach. Ideas about competence acquisition have been long known in psychology, dating at least back to White (1959). It nevertheless remains a relatively little studied topic in RL research, likely because there are many challenges. First, we need to be able to learn a compressed goal space, since the entire state space is too large. Second, we need to design a criterion to sample a next goal from this space, for example based on learning progress. And third, we need to be able to return to a previous goal, re-using the knowledge from how we previously got there. Since we cannot afford to store all individual traces, this will also involve some type of function approximation. In short, there are many remaining challenges (Péré et al., 2018), but the principle has a strong motivation in human psychology, and holds the promise of strong improvement over step-based perturbation methods, especially in tasks with sparse rewards (Ecoffet et al., 2019).

SAFETY Finally, we believe that research on model-based reinforcement learning will also benefit research on safety. Safe exploration is an

important topic for real-world applications, and has especially received attention in the robotics RL community (Garcia and Fernández, 2015). There are two main ideas. First, we want to be able to explore while also retaining reasonable confidence that we will be able to return to a safe state (Berkenkamp et al., 2017). This is clearly a model-based endeavor, where we plan an exploratory step, but also an expected way to get back. Thereby, model-based RL can also contribute to safety and real-world deployment of AI.

In short, the combination of planning and learning (model-based reinforcement learning) is related to many crucial topics in sequential decision-making research, like data efficiency, transfer, hierarchy and exploration. A variety of research topics, each worth several PhD projects, lie ahead. We summarized some important directions above.

10 CONCLUSION

Pooh floated gracefully up into the sky, and stayed there – level with the top of the tree and about twenty feet away from it.
"Hooray!" Christopher Robin shouted.
"Isn't that fine?" shouted Winnie-the-Pooh down to Christopher Robin. "What do I look like?"
"You look like a bear holding on to a balloon," Christopher Robin said.
"Not . . ." said Pooh anxiously, "not like a small black cloud in the blue sky?"
"Not very much?"
"Ah, well, perhaps from up here it looks different. And, as I say, you never can tell with bees."

A.A. Milne, *Winnie-the-Pooh and Some Bees* (1926)

This book studied the integration of planning and learning. Backed-up by concurrent research results in recent years (Levine and Abbeel, 2014; Schrittwieser et al., 2019; Silver et al., 2017c), it identifies planning-learning integrations as a vital research area in artificial intelligence. Both fields are closely intertwined, have the same underlying dimensions, but use different approaches and conventions for their challenges. Integration of both approaches can take place in a variety of ways, which are extensively covered throughout the book. There are many new ways in which both fields can be integrated. We illustrated potential new combinations by showing a novel learning method to deal with stochastic dynamics models, an extension of a well-known planning-learning paradigm to continuous action spaces, a study of the computational trade-off present in planning-learning integrations, and a novel tree search method based on concepts from the learning community. There are numerous remaining directions for future research, like hierarchical RL, transfer learning, and implicit model-based RL. Altogether, the book provides both a theoretical and empirical view on planning-learning integrations. It can serve as an introduction to either or both fields, illustrates the vast remaining potential of the integration of planning and learning, and open up towards future research.

To conclude, the epigraph above this chapter shows the outcome of the planning process started by Winnieh-the-Pooh in the epigraph

of the introduction. Eventually, Pooh decides to take the blue balloon, hoping that the bees will take him for a dark cloud. Clearly, Pooh stops planning after a while and decides on a real action, in this work referred to as a real step. Humans, and fictional bears, are fundamentally limited in their computational budgets: the full planning problems we face are simply too big to every fully enumerate. Our work shows intelligent decision making does requires some planning like Pooh does, but also requires us to stop and move forward at some point as well. It will bring new experience, which can improve the learned models of the world, and, with additional planning, improve the learned value of different actions. Planning and learning are above all complementary phenomena.

BIBLIOGRAPHY

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. (2016). "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16, pp. 265–283.

Abbeel, Pieter and Andrew Y Ng (2004). "Apprenticeship learning via inverse reinforcement learning." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 1.

– (2005). "Learning first-order Markov models for control." In: *Advances in neural information processing systems*, pp. 1–8.

Achiam, Joshua, Harrison Edwards, Dario Amodei, and Pieter Abbeel (2018). "Variational option discovery algorithms." In: *arXiv preprint arXiv:1807.10299*.

Achiam, Joshua and Shankar Sastry (2017). "Surprise-based intrinsic motivation for deep reinforcement learning." In: *arXiv preprint arXiv:1703.01732*.

Agostinelli, Forest, Stephen McAleer, Alexander Shmakov, and Pierre Baldi (2019). "Solving the Rubik's cube with deep reinforcement learning and search." In: *Nature Machine Intelligence* 1.8, pp. 356–363.

Agrawal, Pulkit, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine (2016). "Learning to poke by poking: Experiential learning of intuitive physics." In: *Advances in Neural Information Processing Systems*, pp. 5074–5082.

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (2016). "Concrete problems in AI safety." In: *arXiv preprint arXiv:1606.06565*.

Anand, Ankesh, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm (2019). "Unsupervised state representation learning in atari." In: *Advances in Neural Information Processing Systems*, pp. 8766–8779.

Anderson, Brian DO and John B Moore (2007). *Optimal control: linear quadratic methods*. Courier Corporation.

Anthony, Thomas, Robert Nishihara, Philipp Moritz, Tim Salimans, and John Schulman (2019). "Policy Gradient Search: Online Planning and Expert Iteration without Search Trees." In: *arXiv preprint arXiv:1904.03646*.

Anthony, Thomas, Zheng Tian, and David Barber (2017). "Thinking fast and slow with deep learning and tree search." In: *Advances in Neural Information Processing Systems*, pp. 5360–5370.

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath (2017). "Deep reinforcement learning: A brief survey." In: *IEEE Signal Processing Magazine* 34.6, pp. 26–38.

Asadi, Kavosh, Evan Cater, Dipendra Misra, and Michael L Littman (2018). "Towards a Simple Approach to Multi-step Model-based Reinforcement Learning." In: *arXiv preprint arXiv:1811.00128*.

Asmuth, John, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate (2009). "A Bayesian sampling approach to exploration in reinforcement learning." In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, pp. 19–26.

Åström, Karl Johan and Peter Eykhoff (1971). "System identification—a survey." In: *Automatica* 7.2, pp. 123–162.

Aswani, Anil, Humberto Gonzalez, S Shankar Sastry, and Claire Tomlin (2013). "Provably safe and robust learning-based model predictive control." In: *Automatica* 49.5, pp. 1216–1226.

Atiya, Amir F, Alexander G Parlos, and Lester Ingber (2003). "A reinforcement learning method based on adaptive simulated annealing." In: *2003 46th Midwest Symposium on Circuits and Systems*. Vol. 1. IEEE, pp. 121–124.

Atkeson, Christopher G, Andrew W Moore, and Stefan Schaal (1997). "Locally weighted learning for control." In: *Lazy learning*. Springer, pp. 75–113.

Atkeson, Christopher G and Juan Carlos Santamaria (1997). "A comparison of direct and model-based reinforcement learning." In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. IEEE, pp. 3557–3564.

Audibert, Jean-Yves and Sébastien Bubeck (2010). "Best arm identification in multi-armed bandits." In:

Auer, Peter (2002). "Using confidence bounds for exploitation-exploration trade-offs." In: *Journal of Machine Learning Research* 3.Nov, pp. 397–422.

Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer (2002). "Finite-time analysis of the multiarmed bandit problem." In: *Machine learning* 47.2-3, pp. 235–256.

Avila Belbute-Peres, Filipe de, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter (2018). "End-to-end differentiable physics for learning and control." In: *Advances in Neural Information Processing Systems*, pp. 7178–7189.

Babaeizadeh, Mohammad, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine (2017). "Stochastic variational video prediction." In: *arXiv preprint arXiv:1710.11252*.

Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2017). "The option-critic architecture." In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Bagnell, J Andrew and Jeff G Schneider (2001). "Autonomous helicopter control using reinforcement learning policy search methods." In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. Vol. 2. IEEE, pp. 1615–1620.

Baranes, Adrien and Pierre-Yves Oudeyer (2009). "R-iac: Robust intrinsically motivated exploration and active learning." In: *IEEE Transactions on Autonomous Mental Development* 1.3, pp. 155–169.

– (2013). "Active learning of inverse models with intrinsically motivated goal exploration in robots." In: *Robotics and Autonomous Systems* 61.1, pp. 49–73.

Barreto, André, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver (2017). "Successor features for transfer in reinforcement learning." In: *Advances in neural information processing systems*, pp. 4055–4065.

Barto, Andrew G, Steven J Bradtke, and Satinder P Singh (1995). "Learning to act using real-time dynamic programming." In: *Artificial intelligence* 72.1-2, pp. 81–138.

Barto, Andrew G and Sridhar Mahadevan (2003). "Recent advances in hierarchical reinforcement learning." In: *Discrete event dynamic systems* 13.1-2, pp. 41–77.

Barto, Andrew G, Richard S Sutton, and Charles W Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems." In: *IEEE transactions on systems, man, and cybernetics* 5, pp. 834–846.

Battaglia, Peter W, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. (2018). "Relational inductive biases, deep learning, and graph networks." In: *arXiv preprint arXiv:1806.01261*.

Battaglia, Peter, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. (2016). "Interaction networks for learning about objects, relations and physics." In: *Advances in neural information processing systems*, pp. 4502–4510.

Baxter, Jonathan, Andrew Tridgell, and Lex Weaver (1999). "TDLeaf (lambda): Combining temporal difference learning with game-tree search." In: *arXiv preprint cs/9901001*.

Beck, Jacob, Kamil Ciosek, Sam Devlin, Sebastian Tschiatschek, Cheng Zhang, and Katja Hofmann (2020). "Amrl: Aggregated memory for reinforcement learning." In:

Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The arcade learning environment: An evaluation platform for general agents." In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellemare, Marc, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos (2016). "Unifying count-based exploration and intrinsic motivation." In: *Advances in Neural Information Processing Systems*, pp. 1471–1479.

Bellman, Richard (1966). "Dynamic programming." In: *Science* 153.3731, pp. 34–37.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston (2009). "Curriculum learning." In: *Proceedings of the 26th annual international conference on machine learning*. ACM, pp. 41–48.

Berkenkamp, Felix, Matteo Turchetta, Angela Schoellig, and Andreas Krause (2017). "Safe model-based reinforcement learning with stability guarantees." In: *Advances in neural information processing systems*, pp. 908–918.

Berliner, Hans (1981). "The B* tree search algorithm: A best-first proof procedure." In: *Readings in Artificial Intelligence*. Elsevier, pp. 79–87.

Bertsekas, Dimitri P (1995). *Dynamic programming and optimal control*. Vol. 1. 2.

Bertsekas, Dimitri P and John N Tsitsiklis (1991). "An analysis of stochastic shortest path problems." In: *Mathematics of Operations Research* 16.3, pp. 580–595.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Bock, Hans Georg and Karl-Josef Plitt (1984). "A multiple shooting algorithm for direct solution of optimal control problems." In: *IFAC Proceedings Volumes* 17.2, pp. 1603–1608.

Boden, Margaret A (1988). *Computer models of mind: Computational approaches in theoretical psychology*. Cambridge University Press.

Boone, Gary (1997). "Efficient reinforcement learning: Model-based acrobot control." In: *Proceedings of International Conference on Robotics and Automation*. Vol. 1. IEEE, pp. 229–234.

Botvinick, Matthew and Marc Toussaint (2012). "Planning as inference." In: *Trends in cognitive sciences* 16.10, pp. 485–488.

Bradtke, Steven J and Andrew G Barto (1996). "Linear least-squares algorithms for temporal difference learning." In: *Machine learning* 22.1-3, pp. 33–57.

Brafman, Ronen I and Moshe Tennenholtz (2002). "R-max-a general polynomial time algorithm for near-optimal reinforcement learning." In: *Journal of Machine Learning Research* 3.Oct, pp. 213–231.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "OpenAI Gym." In: *arXiv preprint arXiv:1606.01540*.

Browne, Cameron B, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton (2012). "A survey of monte carlo tree search methods." In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Brunskill, Emma and Lihong Li (2014). "Pac-inspired option discovery in lifelong reinforcement learning." In: *International conference on machine learning*, pp. 316–324.

Buckman, Jacob, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee (2018). "Sample-efficient reinforcement learning with stochastic ensemble value expansion." In: *Advances in Neural Information Processing Systems*, pp. 8224–8234.

Buesing, Lars, Theophane Weber, Sebastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. (2018). "Learning and querying fast generative models for reinforcement learning." In: *arXiv preprint arXiv:1802.03006*.

Burda, Yuri, Roger Grosse, and Ruslan Salakhutdinov (2015). "Importance weighted autoencoders." In: *arXiv preprint arXiv:1509.00519*.

Busoniu, L, R Babuska, and B De Schutter (2008). "A Comprehensive Survey of Multiagent Reinforcement Learning." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 2.38, pp. 156–172.

Campbell, Murray, A Joseph Hoane Jr, and Feng-hsiung Hsu (2002). "Deep blue." In: *Artificial intelligence* 134.1-2, pp. 57–83.

Carmel, David and Shaul Markovitch (1999). "Exploration strategies for model-based learning in multi-agent systems: Exploration strategies." In: *Autonomous Agents and Multi-agent systems* 2.2, pp. 141–172.

Caruana, Rich (1997). "Multitask learning." In: *Machine learning* 28.1, pp. 41–75.

Castro, Pablo Samuel and Doina Precup (2007). "Using Linear Programming for Bayesian Exploration in Markov Decision Processes." In: *IJCAI*. Vol. 24372442.

Cazenave, Tristan and Nicolas Jouandeau (2007). "On the parallelization of UCT." In: *proceedings of the Computer Games Workshop*. Citeseer, pp. 93–101.

Cazenave, Tristan and Abdallah Saffidine (2010). "Score bounded Monte-Carlo tree search." In: *International Conference on Computers and Games*. Springer, pp. 93–104.

Chang, Kai-Wei, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford (2015). "Learning to Search Better than Your Teacher." In: *International Conference on Machine Learning*, pp. 2058–2066.

Chang, Michael B, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum (2016). "A compositional object-based approach to learning physical dynamics." In: *arXiv preprint arXiv:1612.00341*.

Chaslot, Guillaume M JB, Mark HM Winands, H Jaap Van Den Herik, Jos WHM Uiterwijk, Bruno Bouzy, et al. (2008a). "Progressive Strategies For Monte-Carlo Tree Search." In: *New Mathematics and Natural Computation (NMNC)* 4.03, pp. 343–357.

Chaslot, Guillaume, Sander Bakkes, Istvan Szita, and Pieter Spronck (2008b). "Monte-Carlo Tree Search: A New Framework for Game AI." In: *AIIDE*.

Chen, Xi, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel (2016). "Variational Lossy Autoencoder." In: *arXiv preprint arXiv:1611.02731*.

Chentanez, Nuttapong, Andrew G Barto, and Satinder P Singh (2005). "Intrinsically motivated reinforcement learning." In: *Advances in neural information processing systems*, pp. 1281–1288.

Chiappa, Silvia, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed (2017). "Recurrent environment simulators." In: *arXiv preprint arXiv:1704.02254*.

Choi, Jongwook, Yijie Guo, Marcin Moczulski, Junhyuk Oh, Neal Wu, Mohammad Norouzi, and Honglak Lee (2018). "Contingency-aware exploration in reinforcement learning." In: *arXiv preprint arXiv:1811.01483*.

Chrisman, Lonnie (1992). "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach." In: *AAAI*. Vol. 1992. Citeseer, pp. 183–188.

Christiano, Paul, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba (2016).

"Transfer from simulation to real world through learning deep inverse dynamics model." In: *arXiv preprint arXiv:1610.03518*.

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine (2018). "Deep reinforcement learning in a handful of trials using probabilistic dynamics models." In: *Advances in Neural Information Processing Systems*, pp. 4754–4765.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio (2015). "A recurrent latent variable model for sequential data." In: *Advances in neural information processing systems*, pp. 2980–2988.

Clavera, Ignasi, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel (2018). "Model-Based Reinforcement Learning via Meta-Policy Optimization." In: *Conference on Robot Learning*, pp. 617–629.

Corneil, Dane, Wulfram Gerstner, and Johanni Brea (2018). "Efficient model-based deep reinforcement learning with variational state tabulation." In: *arXiv preprint arXiv:1802.04325*.

Couëtoux, Adrien, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard (2011). "Continuous upper confidence trees." In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 433–445.

Coulom, Rémi (2006). "Efficient selectivity and backup operators in Monte-Carlo tree search." In: *International conference on computers and games*. Springer, pp. 72–83.

– (2007). "Computing elo ratings of move patterns in the game of go." In: *Computer games workshop*.

Coumans, Erwin and Yunfei Bai (2016). "Pybullet, a python module for physics simulation for games, robotics and machine learning." In: *GitHub repository*.

Craik, Kenneth James Williams (1943). "The Nature of Explanation." In:

Da Silva, Bruno C, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel (2006). "Dealing with non-stationary environments using context detection." In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 217–224.

Daniel, Christian, Herke Van Hoof, Jan Peters, and Gerhard Neumann (2016). "Probabilistic inference for determining options in reinforcement learning." In: *Machine Learning* 104.2-3, pp. 337–357.

Darwin, Charles (1859). "On the Origin of Species." In:

Dayan, Peter (1993). "Improving generalization for temporal difference learning: The successor representation." In: *Neural Computation* 5.4, pp. 613–624.

Dayan, Peter and Geoffrey E Hinton (1993). "Feudal reinforcement learning." In: *Advances in neural information processing systems*, pp. 271–278.

Dearden, Richard, Nir Friedman, and David Andre (1999). "Model based Bayesian exploration." In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 150–159.

Dearden, Richard, Nir Friedman, and Stuart Russell (1998). "Bayesian Q-learning." In: *AAAI/IAAI*, pp. 761–768.

Degrave, Jonas, Michiel Hermans, Joni Dambre, et al. (2019). "A differentiable physics engine for deep learning in robotics." In: *Frontiers in neurorobotics* 13.

Deisenroth, Marc and Carl E Rasmussen (2011). "PILCO: A model-based and data-efficient approach to policy search." In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472.

Depeweg, Stefan, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft (2016). "Learning and policy search in stochastic dynamical systems with bayesian neural networks." In: *arXiv preprint arXiv:1605.07127*.

Der Kiureghian, Armen and Ove Ditlevsen (2009). "Aleatory or epistemic? Does it matter?" In: *Structural Safety* 31.2, pp. 105–112.

Dijkstra, Edsger W (1959). "A note on two problems in connexion with graphs." In: *Numerische mathematik* 1.1, pp. 269–271.

Dilokthanakul, Nat, Christos Kaplanis, Nick Pawlowski, and Murray Shanahan (2019). "Feature control as intrinsic motivation for hierarchical reinforcement learning." In: *IEEE transactions on neural networks and learning systems*.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). "Density estimation using Real NVP." In: *arXiv preprint arXiv:1605.08803*.

Diuk, Carlos, Andre Cohen, and Michael L Littman (2008). "An object-oriented representation for efficient reinforcement learning." In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 240–247.

Doll, Bradley B, Dylan A Simon, and Nathaniel D Daw (2012). "The ubiquity of model-based reinforcement learning." In: *Current opinion in neurobiology* 22.6, pp. 1075–1081.

Doya, Kenji, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato (2002). "Multiple model-based reinforcement learning." In: *Neural computation* 14.6, pp. 1347–1369.

Duff, Michael O'Gordon and Andrew Barto (2002). "Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes." PhD thesis. University of Massachusetts at Amherst.

Ecoffet, Adrien, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune (2019). "Go-explore: a new approach for hard-exploration problems." In: *arXiv preprint arXiv:1901.10995*.

Edwards, Ashley D, Laura Downs, and James C Davidson (2018). "Forward-backward reinforcement learning." In: *arXiv preprint arXiv:1803.10227*.

Edwards, Daniel James and TP Hart (1961). "The alpha-beta heuristic." In:

Edwards, Ward (1954). "The theory of decision making." In: *Psychological bulletin* 51.4, p. 380.

Efroni, Yonathan, Gal Dalal, Bruno Scherrer, and Shie Mannor (2018). "Beyond the One-Step Greedy Approach in Reinforcement Learning." In: *International Conference on Machine Learning*, pp. 1386–1395.

– (2019). "How to combine tree-search methods in reinforcement learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 3494–3501.

Efroni, Yonathan, Mohammad Ghavamzadeh, and Shie Mannor (2019). "Multi-Step Greedy and Approximate Real Time Dynamic Programming." In: *arXiv preprint arXiv:1909.04236*.

El Hihi, Salah and Yoshua Bengio (1996). "Hierarchical recurrent neural networks for long-term dependencies." In: *Advances in neural information processing systems*, pp. 493–499.

Evans, Jonathan St BT (1984). "Heuristic and analytic processes in reasoning." In: *British Journal of Psychology* 75.4, pp. 451–468.

Eysenbach, Benjamin, Abhishek Gupta, Julian Ibarz, and Sergey Levine (2019). "Diversity is All You Need: Learning Skills without a Reward Function." In: *International Conference on Learning Representations*.

Fairbank, Michael and Eduardo Alonso (2012). "Value-gradient learning." In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Farquhar, Gregory, Tim Rocktäschel, Maximilian Igl, and SA Whiteson (2018). "Treeqn and atreec: Differentiable tree planning for deep reinforcement learning." In: International Conference on Learning Representations.

Finn, Chelsea, Ian Goodfellow, and Sergey Levine (2016). "Unsupervised learning for physical interaction through video prediction." In: *Advances in neural information processing systems*, pp. 64–72.

Florensa, Carlos, Yan Duan, and Pieter Abbeel (2017). "Stochastic neural networks for hierarchical reinforcement learning." In: *arXiv preprint arXiv:1704.03012*.

Florensa, Carlos, David Held, Xinyang Geng, and Pieter Abbeel (2018). "Automatic Goal Generation for Reinforcement Learning Agents." In: *International Conference on Machine Learning*, pp. 1514–1523.

Fox, Roy, Sanjay Krishnan, Ion Stoica, and Ken Goldberg (2017). "Multi-level discovery of deep options." In: *arXiv preprint arXiv:1703.08294*.

Fox, Roy, Michal Moshkovitz, and Naftali Tishby (2016). "Principled option learning in Markov decision processes." In: *arXiv preprint arXiv:1609.05524*.

Fraccaro, Marco, Simon Kamronn, Ulrich Paquet, and Ole Winther (2017). "A disentangled recognition and nonlinear dynamics model for unsupervised learning." In: *Advances in Neural Information Processing Systems*, pp. 3601–3610.

Fragkiadaki, Katerina, Pulkit Agrawal, Sergey Levine, and Jitendra Malik (2015). "Learning visual predictive models of physics for playing billiards." In: *arXiv preprint arXiv:1511.07404*.

François-Lavet, Vincent, Yoshua Bengio, Doina Precup, and Joelle Pineau (2019). "Combined reinforcement learning via abstract representations." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 3582–3589.

François-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. (2018). "An introduction to deep reinforcement learning." In: *Foundations and Trends® in Machine Learning* 11.3-4, pp. 219–354.

Frans, Kevin, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman (2018). "Meta learning shared hierarchies." In: *International Conference on Learning Representations*.

Fröhlich, Fabian, Fabian J Theis, and Jan Hasenauer (2014). "Uncertainty analysis for non-identifiable dynamical systems: Profile likelihoods, bootstrapping and more." In: *International Conference on Computational Methods in Systems Biology*. Springer, pp. 61–72.

Fu, Justin, Sergey Levine, and Pieter Abbeel (2016). "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4019–4026.

Gal, Yarin, Rowan McAllister, and Carl Edward Rasmussen (2016). "Improving PILCO with Bayesian neural network dynamics models." In: *Data-Efficient Machine Learning workshop, ICML*. Vol. 4.

Garcia, Javier and Fernando Fernández (2015). "A comprehensive survey on safe reinforcement learning." In: *Journal of Machine Learning Research* 16.1, pp. 1437–1480.

Garnelo, Marta, Kai Arulkumaran, and Murray Shanahan (2016). "Towards deep symbolic reinforcement learning." In: *arXiv preprint arXiv:1609.05518*.

Geffner, Hector and Blai Bonet (2013). "A concise introduction to models and methods for automated planning." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8.1, pp. 1–141.

Gelly, Sylvain and Yizao Wang (2006). "Exploration exploitation in go: UCT for Monte-Carlo go." In: *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*.

Gemici, Mevlana, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap (2017). "Generative temporal models with memory." In: *arXiv preprint arXiv:1702.04649*.

Ghahramani, Zoubin and Geoffrey E Hinton (1996). *Parameter estimation for linear dynamical systems*. Tech. rep. Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science.

Ghahramani, Zoubin and Sam T Roweis (1999). "Learning nonlinear dynamical systems using an EM algorithm." In: *Advances in neural information processing systems*, pp. 431–437.

Ghallab, Malik, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). "PDDL—the planning domain definition language." In: *AIPS-98 planning committee* 3, p. 14.

Ghavamzadeh, Mohammad, Shie Mannor, Joelle Pineau, Aviv Tamar, et al. (2015). "Bayesian reinforcement learning: A survey." In: *Foundations and Trends® in Machine Learning* 8.5-6, pp. 359–483.

Ghosh, Dibya, Abhishek Gupta, and Sergey Levine (2018). "Learning Actionable Representations with Goal-Conditioned Policies." In: *arXiv preprint arXiv:1811.07819*.

Goel, Sandeep and Manfred Huber (2003). "Subgoal discovery for hierarchical reinforcement learning using learned policies." In: *FLAIRS conference*, pp. 346–350.

Goodfellow, Ian (2016). "NIPS 2016 Tutorial: Generative Adversarial Networks." In: *arXiv preprint arXiv:1701.00160*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. MIT press.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural turing machines." In: *arXiv preprint arXiv:1410.5401*.

Gregor, Karol, Danilo Jimenez Rezende, and Daan Wierstra (2016). "Variational intrinsic control." In: *arXiv preprint arXiv:1611.07507*.

Grimm, Christopher, André Barreto, Satinder Singh, and David Silver (2020). "The Value Equivalence Principle for Model-Based Reinforcement Learning." In: *Advances in Neural Information Processing Systems*.

Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine (2016). "Continuous deep q-learning with model-based acceleration." In: *International Conference on Machine Learning*, pp. 2829–2838.

Guestrin, Carlos, Daphne Koller, Chris Gearhart, and Neal Kanodia (2003). "Generalizing plans to new environments in relational MDPs." In: *Proceedings of the 18th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., pp. 1003–1010.

Guez, Arthur, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. (2019). "An Investigation of Model-Free Planning." In: *International Conference on Machine Learning*, pp. 2464–2473.

Guez, Arthur, David Silver, and Peter Dayan (2012). "Efficient Bayes-adaptive reinforcement learning using sample-based search." In: *Advances in neural information processing systems*, pp. 1025–1033.

Guez, Arthur, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver (2018). "Learning to search with MCTSnets." In: *arXiv preprint arXiv:1802.04697*.

Guo, Xiaoxiao, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang (2014). "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning." In: *Advances in neural information processing systems*, pp. 3338–3346.

Ha, David and Jürgen Schmidhuber (2018). "Recurrent world models facilitate policy evolution." In: *Advances in Neural Information Processing Systems*, pp. 2450–2462.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor." In: *arXiv preprint arXiv:1801.01290*.

Hafner, Danijar, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi (2019a). "Dream to Control: Learning Behaviors by Latent Imagination." In: *International Conference on Learning Representations*.

Hafner, Danijar, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2019b). "Learning latent

dynamics for planning from pixels." In: *International Conference on Machine Learning*. PMLR, pp. 2555–2565.

Hamidi, Mandana, Prasad Tadepalli, Robby Goetschalckx, and Alan Fern (2015). "Active imitation learning of hierarchical policies." In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Hamrick, Jessica B (2019). "Analogues of mental simulation and imagination in deep learning." In: *Current Opinion in Behavioral Sciences* 29, pp. 8–16.

Hamrick, Jessica B, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia (2017). "Metacontrol for adaptive imagination-based optimization." In: *arXiv preprint arXiv:1705.02670*.

Hamrick, Jessica B, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W Battaglia (2020). "Combining q-learning and search with amortized value estimates." In: *International Conference on Learning Representations (ICLR)*.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hasselt, Hado P van, Matteo Hessel, and John Aslanides (2019). "When to use parametric models in reinforcement learning?" In: *Advances in Neural Information Processing Systems*, pp. 14322–14333.

Haugeland, John (1989). *Artificial intelligence: The very idea*. MIT press.

Hausman, Karol, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller (2018). "Learning an Embedding Space for Transferable Robot Skills." In: *International Conference on Learning Representations*.

Heess, Nicolas, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver (2016). "Learning and transfer of modulated locomotor controllers." In: *arXiv preprint arXiv:1610.05182*.

Heess, Nicolas, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa (2015). "Learning continuous control policies by stochastic value gradients." In: *Advances in Neural Information Processing Systems*, pp. 2944–2952.

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2018). "Deep reinforcement learning that matters." In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hengst, Bernhard (2017). "Hierarchical reinforcement learning." In: *Encyclopedia of Machine Learning and Data Mining*, pp. 611–619.

Hess, Eckhard H (1959). "Imprinting." In: *Science* 130.3368, pp. 133–141.

Hester, Todd and Peter Stone (2012a). "Intrinsically motivated model learning for a developing curious agent." In: *2012 IEEE international*

conference on development and learning and epigenetic robotics (ICDL). IEEE, pp. 1–6.

Hester, Todd and Peter Stone (2012b). "Learning and using models." In: *Reinforcement learning*. Springer, pp. 111–141.

– (2013). "TEXPLORE: real-time sample-efficient reinforcement learning for robots." In: *Machine learning* 90.3, pp. 385–429.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Holland, G Zacharias, Erin J Talvitie, and Michael Bowling (2018). "The effect of planning shape on dyna-style planning in high-dimensional state spaces." In: *arXiv preprint arXiv:1806.01825*.

Holmberg, John and K-H Robèrt (2000). "Backcasting—A framework for strategic planning." In: *International Journal of Sustainable Development & World Ecology* 7.4, pp. 291–308.

Holroyd, Clay B and Michael GH Coles (2002). "The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity." In: *Psychological review* 109.4, p. 679.

Houthooft, Rein, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "Vime: Variational information maximizing exploration." In: *Advances in Neural Information Processing Systems*, pp. 1109–1117.

Howard, Ronald A (1960). "Dynamic programming and markov processes." In:

Huang, Shih-Chieh, Remi Coulom, and Shun-Shii Lin (2010). "Time management for Monte-Carlo tree search applied to the game of Go." In: *2010 International Conference on Technologies and Applications of Artificial Intelligence*. IEEE, pp. 462–466.

Jaakkola, Tommi and David Haussler (1999). "Exploiting generative models in discriminative classifiers." In: *Advances in neural information processing systems*, pp. 487–493.

Jaderberg, Max, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. (2019). "Human-level performance in 3D multiplayer games with population-based reinforcement learning." In: *Science* 364.6443, pp. 859–865.

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu (2016). "Reinforcement learning with unsupervised auxiliary tasks." In: *arXiv preprint arXiv:1611.05397*.

Jang, Eric, Shixiang Gu, and Ben Poole (2016). "Categorical Reparameterization with Gumbel-Softmax." In: *arXiv preprint arXiv:1611.01144*.

Janner, Michael, Justin Fu, Marvin Zhang, and Sergey Levine (2019). "When to trust your model: Model-based policy optimization." In: *Advances in Neural Information Processing Systems*, pp. 12519–12530.

Jaulmes, Robin, Joelle Pineau, and Doina Precup (2005). "Learning in non-stationary partially observable Markov decision processes." In: *ECML Workshop on Reinforcement Learning in non-stationary environments*. Vol. 25, pp. 26–32.

Jayaraman, Dinesh, Frederik Ebert, Alexei A Efros, and Sergey Levine (2018). "Time-agnostic prediction: Predicting predictable video frames." In: *arXiv preprint arXiv:1808.07784*.

Jiang, Daniel, Emmanuel Ekwedike, and Han Liu (2018). "Feedback-Based Tree Search for Reinforcement Learning." In: *International Conference on Machine Learning*, pp. 2289–2298.

Jiang, Nan, Alex Kulesza, Satinder Singh, and Richard Lewis (2015). "The dependence of effective planning horizon on model accuracy." In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. Citeseer, pp. 1181–1189.

Jong, Nicholas K and Peter Stone (2007). "Model-based function approximation in reinforcement learning." In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, p. 95.

Jonschkowski, Rico and Oliver Brock (2015). "Learning state representations with robotic priors." In: *Autonomous Robots* 39.3, pp. 407–428.

Jordan, Michael I and David E Rumelhart (1992). "Forward models: Supervised learning with a distal teacher." In: *Cognitive science* 16.3, pp. 307–354.

Kaelbling, Leslie Pack (1993). *Learning in embedded systems*. MIT press.

Kahneman, Daniel (2011). *Thinking, fast and slow*. Macmillan.

Kakade, Sham Machandranath et al. (2003). "On the sample complexity of reinforcement learning." PhD thesis. University of London London, England.

Kalchbrenner, Nal, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu (2017). "Video pixel networks." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1771–1779.

Kalman, Rudolph Emil (1960). "Contributions to the Theory of Optimal Control." In: *Bol. Soc. Mat. Mex.* 5, pp. 102–199.

Kalweit, Gabriel and Joschka Boedecker (2017). "Uncertainty-driven imagination for continuous deep reinforcement learning." In: *Conference on Robot Learning*, pp. 195–206.

Kamthe, Sanket and Marc Peter Deisenroth (2017). "Data-efficient reinforcement learning with probabilistic model predictive control." In: *arXiv preprint arXiv:1706.06491*.

Kansky, Ken, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George (2017). "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1809–1818.

Kappen, Hilbert J, Vicenç Gómez, and Manfred Opper (2012). "Optimal control as a graphical model inference problem." In: *Machine learning* 87.2, pp. 159–182.

Karl, Maximilian, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt (2016). "Deep variational bayes filters: Unsupervised learning of state space models from raw data." In: *arXiv preprint arXiv:1605.06432*.

Kaufmann, Emilie and Wouter M Koolen (2017). "Monte-carlo tree search by best arm identification." In: *Advances in Neural Information Processing Systems*, pp. 4897–4906.

Ke, Nan Rosemary, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra (2019). "Learning Dynamics Model in Reinforcement Learning by Incorporating the Long Term Future." In: *arXiv preprint arXiv:1903.01599*.

Keller, Thomas (2015). "Anytime optimal MDP planning with trial-based heuristic tree search." PhD thesis. University of Freiburg, Freiburg im Breisgau, Germany.

Keller, Thomas and Malte Helmert (2013). "Trial-based heuristic tree search for finite horizon MDPs." In: *Twenty-Third International Conference on Automated Planning and Scheduling*.

Kempka, Michał, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski (2016). "Vizdoom: A doom-based ai research platform for visual reinforcement learning." In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Kendall, Alex and Yarin Gal (2017). "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Advances in neural information processing systems*, pp. 5574–5584.

Keramati, Mehdi, Amir Dezfouli, and Payam Piray (2011). "Speed/accuracy trade-off between the habitual and the goal-directed processes." In: *PLoS computational biology* 7.5.

Khansari-Zadeh, S Mohammad and Aude Billard (2011). "Learning stable nonlinear dynamical systems with gaussian mixture models." In: *IEEE Transactions on Robotics* 27.5, pp. 943–957.

Kingma, Diederik P, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling (2016). "Improved Variational Inference with Inverse Autoregressive Flow." In: *Advances in Neural Information Processing Systems*, pp. 4743–4751.

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes." In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kingma, Diederik and Jimmy Ba (Dec. 2014). "Adam: A Method for Stochastic Optimization." In: *International Conference on Learning Representations*.

Kipf, Thomas, Elise van der Pol, and Max Welling (2020). "Contrastive Learning of Structured World Models." In: *International Conference on Learning Representations*.

Knuth, Donald E and Ronald W Moore (1975). "An analysis of alpha-beta pruning." In: *Artificial intelligence* 6.4, pp. 293–326.

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey." In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit based monte-carlo planning." In: *ECML*. Vol. 6. Springer, pp. 282–293.

Kolobov, Andrey (2012). "Planning with Markov decision processes: An AI perspective." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 1, pp. 1–210.

Kolter, J Zico and Andrew Y Ng (2009). "Near-Bayesian exploration in polynomial time." In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pp. 513–520.

Konidaris, George D (2006). "A framework for transfer in reinforcement learning." In: *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*.

Konidaris, George and Andrew G Barto (2007). "Building Portable Options: Skill Transfer in Reinforcement Learning." In: *IJCAI*. Vol. 7, pp. 895–900.

Konidaris, George, Scott Kuindersma, Roderic Grupen, and Andrew Barto (2012). "Robot learning from demonstration by constructing skill trees." In: *The International Journal of Robotics Research* 31.3, pp. 360–375.

Korf, Richard E (1990). "Real-time heuristic search." In: *Artificial intelligence* 42.2-3, pp. 189–211.

Kornblith, Simon, Jonathon Shlens, and Quoc V Le (2019). "Do better imagenet models transfer better?" In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2661–2671.

Korte, Bernhard, Jens Vygen, B Korte, and J Vygen (2012). *Combinatorial optimization*. Vol. 2. Springer.

Krishnan, Rahul G., Uri Shalit, and David A Sontag (2015). "Deep Kalman Filters." In: *ArXiv* abs/1511.05121.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*, pp. 1097–1105.

Kulkarni, Tejas D, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation." In: *Advances in neural information processing systems*, pp. 3675–3683.

Kurniawati, Hanna, David Hsu, and Wee Sun Lee (2008). "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces." In: *Robotics: Science and systems*. Vol. 2008. Zurich, Switzerland.

Kurutach, Thanard, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel (2018). "Learning plannable representations with causal infogan." In: *Advances in Neural Information Processing Systems*, pp. 8733–8744.

LaValle, Steven M (1998). "Rapidly-exploring random trees: A new tool for path planning." In:

Lai, Matthew (2015). "Giraffe: Using deep reinforcement learning to play chess." In: *arXiv preprint arXiv:1509.01549*.

Lakshminarayanan, Aravind S, Ramnandan Krishnamurthy, Peeyush Kumar, and Balaraman Ravindran (2016). "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering." In: *arXiv preprint arXiv:1605.05359*.

Lange, Sascha, Thomas Gabel, and Martin Riedmiller (2012). "Batch reinforcement learning." In: *Reinforcement learning*. Springer, pp. 45–73.

Laversanne-Finot, Adrien, Alexandre Pere, and Pierre-Yves Oudeyer (2018). "Curiosity Driven Exploration of Learned Disentangled Goal Spaces." In: *Conference on Robot Learning*, pp. 487–504.

Lazaric, Alessandro (2012). "Transfer in reinforcement learning: a framework and a survey." In: *Reinforcement Learning*. Springer, pp. 143–173.

Leibo, Joel Z, Cyprien de Masson d'Autume, Daniel Zoran, David Amos, Charles Beattie, Keith Anderson, Antonio García Castañeda, Manuel Sanchez, Simon Green, Audrunas Gruslys, et al. (2018). "Psychlab: a psychology laboratory for deep reinforcement learning agents." In: *arXiv preprint arXiv:1801.08116*.

Lesort, Timothée, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat (2018). "State representation learning for control: An overview." In: *Neural Networks* 108, pp. 379–392.

Levine, Sergey and Pieter Abbeel (2014). "Learning neural network policies with guided policy search under unknown dynamics." In: *Advances in Neural Information Processing Systems*, pp. 1071–1079.

Levine, Sergey and Vladlen Koltun (2013). "Guided policy search." In: *International Conference on Machine Learning*, pp. 1–9.

Levy, Andrew, Robert Platt, and Kate Saenko (2019). "Hierarchical Reinforcement Learning with Hindsight." In: *International Conference on Learning Representations*.

Li, Lihong, Michael L Littman, Thomas J Walsh, and Alexander L Strehl (2011). "Knows what it knows: a framework for self-aware learning." In: *Machine learning* 82.3, pp. 399–443.

Li, Yingzhen and Richard E Turner (2016). "Rényi divergence variational inference." In: *Advances in Neural Information Processing Systems*, pp. 1073–1081.

Li, Zhenguo, Fengwei Zhou, Fei Chen, and Hang Li (2017). "Meta-SGD: Learning to learn quickly for few-shot learning." In: *arXiv preprint arXiv:1707.09835*.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971*.

Lin, Long-Ji (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching." In: *Machine learning* 8.3-4, pp. 293–321.

– (1993). *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lin, Long-Ji and Tom M Mitchell (1992). *Memory approaches to reinforcement learning in non-Markovian domains*. Citeseer.

Ljung, Lennart (2001). "System identification." In: *Wiley Encyclopedia of Electrical and Electronics Engineering*.

Lopes, Manuel, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer (2012). "Exploration in model-based reinforcement learning by empir-

ically estimating learning progress." In: *Advances in neural information processing systems*, pp. 206–214.

Lowrey, Kendall, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch (2018). "Plan online, learn offline: Efficient learning and exploration via model-based control." In: *arXiv preprint arXiv:1811.01848*.

Lu, Kevin, Igor Mordatch, and Pieter Abbeel (2019). "Adaptive Online Planning for Continual Lifelong Learning." In: *arXiv preprint arXiv:1912.01188*.

Machado, Marios C, Marc G Bellemare, and Michael Bowling (2017). "A laplacian framework for option discovery in reinforcement learning." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2295–2304.

Machado, Marlos C, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling (2018). "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents." In: *Journal of Artificial Intelligence Research* 61, pp. 523–562.

Mackintosh, Nicholas John (1983). *Conditioning and associative learning*. Clarendon Press Oxford.

Maddison, Chris J, Andriy Mnih, and Yee Whye Teh (2016). "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables." In: *arXiv preprint arXiv:1611.00712*.

Mahadevan, Sridhar (2009). "Learning Representation and Control in Markov Decision Processes: New Frontiers." In: *Foundations and Trends® in Machine Learning* 1.4, pp. 403–565. ISSN: 1935-8237. DOI: 10.1561/2200000003.

Mann, Timothy and Shie Mannor (2014). "Scaling up approximate value iteration with options: Better policies with fewer iterations." In: *International conference on machine learning*, pp. 127–135.

Mannor, Shie, Ishai Menache, Amit Hoze, and Uri Klein (2004). "Dynamic abstraction in reinforcement learning via clustering." In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 71.

Mannor, Shie, Reuven Y Rubinstein, and Yohai Gat (2003). "The cross entropy method for fast policy search." In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 512–519.

Matiisen, Tambet, Avital Oliver, Taco Cohen, and John Schulman (2017). "Teacher-student curriculum learning." In: *arXiv preprint arXiv:1707.00183*.

Mayne, David Q and Hannah Michalska (1990). "Receding horizon control of nonlinear systems." In: *IEEE Transactions on automatic control* 35.7, pp. 814–824.

McCallum, R (1997). "Reinforcement learning with selective perception and hidden state." In:

McGovern, Amy and Andrew G Barto (2001). "Automatic discovery of subgoals in reinforcement learning using diverse density." In:

Menache, Ishai, Shie Mannor, and Nahum Shimkin (2002). "Q-cut—dynamic discovery of sub-goals in reinforcement learning." In: *European Conference on Machine Learning*. Springer, pp. 295–306.

Michalowicz, Joseph Victor, Jonathan M Nichols, and Frank Bucholtz (2013). *Handbook of differential entropy*. Crc Press.

Miller, Neal Elgar and John Dollard (1941). "Social learning and imitation." In:

Mishra, Nikhil, Pieter Abbeel, and Igor Mordatch (2017). "Prediction and control with temporal segment models." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2459–2468.

Mnih, Andriy and Karol Gregor (2014). "Neural variational inference and learning in belief networks." In: *arXiv preprint arXiv:1402.0030*.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, p. 529.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker (2016). "Fear and Hope Emerge from Anticipation in Model-Based Reinforcement Learning." In: *IJCAI*, pp. 848–854.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker (2017a). "Efficient exploration with Double Uncertain Value Networks." In: *Deep Reinforcement Learning Symposium @ NIPS 2017*. arXiv preprint arXiv:1711.10789.

– (2017b). "Learning Multimodal Transition Dynamics for Model-Based Reinforcement Learning." In: *arXiv preprint arXiv:1705.00470*.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker (2018a). "Emotion in reinforcement learning agents and robots: a survey." In: *Machine Learning* 107.2, pp. 443–480.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker (2018b). "The Potential of the Return Distribution for Exploration in RL." In: *arXiv preprint arXiv:1806.04242*.

Moerland, Thomas M., Joost Broekens, and Catholijn M. Jonker (2020a). "A Framework for Reinforcement Learning and Planning." In: *arXiv preprint arXiv:2006.15009*.

Moerland, Thomas M, Joost Broekens, and Catholijn M Jonker (2020b). "Model-based Reinforcement Learning: A Survey." In: *arXiv preprint arXiv:2006.16712*.

Moerland, Thomas M, Joost Broekens, Aske Plaat, and Catholijn M Jonker (2018a). "AoC: Alpha zero in continuous action space." In: *arXiv preprint arXiv:1805.09613*.

– (2018b). "Monte Carlo Tree Search for Asymmetric Trees." In: *arXiv preprint arXiv:1805.09218*.

Moerland, Thomas M, Anna Deichler, Simone Baldi, Joost Broekens, and Catholijn M Jonker (2020). "Think Too Fast Nor Too Slow: The Computational Trade-off Between Planning And Reinforcement Learning." In: *arXiv preprint arXiv:2005.07404*.

Mohamed, Shakir and Danilo Jimenez Rezende (2015). "Variational information maximisation for intrinsically motivated reinforcement learning." In: *Advances in neural information processing systems*, pp. 2125–2133.

Momennejad, Ida, Evan M Russek, Jin H Cheong, Matthew M Botvinick, Nathaniel Douglass Daw, and Samuel J Gershman (2017). "The successor representation in human reinforcement learning." In: *Nature Human Behaviour* 1.9, p. 680.

Moore, Andrew W and Christopher G Atkeson (1993). "Prioritized sweeping: Reinforcement learning with less data and less time." In: *Machine learning* 13.1, pp. 103–130.

Moore, Edward F (1959). "The shortest path through a maze." In: *Proc. Int. Symp. Switching Theory, 1959*, pp. 285–292.

Morari, Manfred and Jay H Lee (1999). "Model predictive control: past, present and future." In: *Computers & Chemical Engineering* 23.4-5, pp. 667–682.

Moriarty, David E, Alan C Schultz, and John J Grefenstette (1999). "Evolutionary algorithms for reinforcement learning." In: *Journal of Artificial Intelligence Research* 11, pp. 241–276.

Morris, Robin and Geoff Ward (2004). *The cognitive psychology of planning*. Psychology Press.

Müller, K-R, Alexander J Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik (1997). "Predicting time series

with support vector machines." In: *International Conference on Artificial Neural Networks*. Springer, pp. 999–1004.

Munos, Rémi, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare (2016). "Safe and efficient off-policy reinforcement learning." In: *Advances in Neural Information Processing Systems*, pp. 1054–1062.

Munos, Rémi et al. (2014). "From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning." In: *Foundations and Trends® in Machine Learning* 7.1, pp. 1–129.

Nachum, Ofir, Shixiang Shane Gu, Honglak Lee, and Sergey Levine (2018). "Data-efficient hierarchical reinforcement learning." In: *Advances in Neural Information Processing Systems*, pp. 3303–3313.

Nagabandi, Anusha, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018a). "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning." In: *International Conference on Learning Representations*.

Nagabandi, Anusha, Chelsea Finn, and Sergey Levine (2018). "Deep online learning via meta-learning: Continual adaptation for model-based RL." In: *arXiv preprint arXiv:1812.07671*.

Nagabandi, Anusha, Gregory Kahn, Ronald S Fearing, and Sergey Levine (2018b). "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7559–7566.

Narendra, Kumpati S and Kannan Parthasarathy (1990). "Identification and control of dynamical systems using neural networks." In: *IEEE Transactions on neural networks* 1.1, pp. 4–27.

Neitz, Alexander, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf (2018). "Adaptive skip intervals: Temporal abstraction for recurrent dynamical models." In: *Advances in Neural Information Processing Systems*, pp. 9816–9826.

Newell, Allen, Herbert Alexander Simon, et al. (1972). *Human problem solving*. Vol. 104. 9.

Ng, Andrew Y, Daishi Harada, and Stuart Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping." In: *ICML*. Vol. 99, pp. 278–287.

Nguyen-Tuong, Duy and Jan Peters (2011). "Model learning for robot control: a survey." In: *Cognitive processing* 12.4, pp. 319–340.

Niv, Yael (2009). "Reinforcement learning in the brain." In: *Journal of Mathematical Psychology* 53.3, pp. 139–154.

Nouri, Ali and Michael L Littman (2010). "Dimension reduction and its application to model-based exploration in continuous spaces." In: *Machine Learning* 81.1, pp. 85–98.

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh (2015). "Action-conditional video prediction using deep networks in atari games." In: *Advances in neural information processing systems*, pp. 2863–2871.

Oh, Junhyuk, Satinder Singh, and Honglak Lee (2017). "Value prediction network." In: *Advances in Neural Information Processing Systems*, pp. 6118–6128.

Oord, Aaron van den, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu (2016). "Conditional image generation with PixelCNN decoders." In: *arXiv preprint arXiv:1606.05328*.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy (2016). "Deep exploration via bootstrapped DQN." In: *Advances in Neural Information Processing Systems*, pp. 4026–4034.

Osband, Ian, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepezvari, Satinder Singh, et al. (2019). "Behaviour Suite for Reinforcement Learning." In: *arXiv preprint arXiv:1908.03568*.

Osband, Ian, Benjamin Van Roy, and Zheng Wen (2016). "Generalization and Exploration via Randomized Value Functions." In: *International Conference on Machine Learning*, pp. 2377–2386.

Ostafew, Chris J, Angela P Schoellig, and Timothy D Barfoot (2016). "Robust constrained learning-based NMPC enabling reliable mobile robot path tracking." In: *The International Journal of Robotics Research* 35.13, pp. 1547–1563.

Ostrovski, Georg, Marc G Bellemare, Aäron van den Oord, and Rémi Munos (2017). "Count-based exploration with neural density models." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2721–2730.

Oudeyer, Pierre-Yves, Frdric Kaplan, and Verena V Hafner (2007). "Intrinsic motivation systems for autonomous mental development." In: *IEEE transactions on evolutionary computation* 11.2, pp. 265–286.

Oudeyer, Pierre-Yves and Frederic Kaplan (2009). "What is intrinsic motivation? A typology of computational approaches." In: *Frontiers in neurorobotics* 1, p. 6.

Oudeyer, Pierre-Yves, Frederic Kaplan, et al. (2008). "How can we define intrinsic motivation." In: *Proc. of the 8th Conf. on Epigenetic Robotics*. Vol. 5, pp. 29–31.

Parlos, Alexander G, Kil To Chong, and Amir F Atiya (1994). "Application of the recurrent multilayer perceptron in modeling complex process dynamics." In: *IEEE Transactions on Neural Networks* 5.2, pp. 255–266.

Parr, Ronald, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman (2008). "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning." In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 752–759.

Pascanu, Razvan, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia (2017). "Learning model-based planning from scratch." In: *arXiv preprint arXiv:1707.06170*.

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell (2017). "Curiosity-driven exploration by self-supervised prediction." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17.

Pavlov, Ivan Petrovitch and William Gantt (1928). "Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals." In:

Pearl, Judea (1984). "Heuristics: intelligent search strategies for computer problem solving." In:

Peeke, Harman (2012). *Habituation, sensitization, and behavior*. Elsevier.

Péré, Alexandre, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer (2018). "Unsupervised learning of goal spaces for intrinsically motivated goal exploration." In: *arXiv preprint arXiv:1803.00781*.

Peshkin, Leonid, Nicolas Meuleau, and Leslie Pack Kaelbling (1999). "Learning Policies with External Memory." In: *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., pp. 307–314.

Peters, Jan, Katharina Mulling, and Yasemin Altun (2010). "Relative entropy policy search." In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Plaat, Aske, Walter Kosters, and Mike Preuss (2020). "Model-Based Deep Reinforcement Learning for High-Dimensional Problems, a Survey." In: *arXiv preprint arXiv:2008.05598*.

Plaat, Aske, Jonathan Schaeffer, Wim Pijls, and Arie De Bruin (1996). "Exploiting graph properties of game trees." In: *AAAI/IAAI, Vol. 1*, pp. 234–239.

Plappert, Matthias, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin

Andrychowicz (2017). "Parameter space noise for exploration." In: *arXiv preprint arXiv:1706.01905*.

Pohl, Ira (1969). *Bidirectional and heuristic search in path problems*. Tech. rep.

Polydoros, Athanasios S and Lazaros Nalpantidis (2017). "Survey of model-based reinforcement learning: Applications on robotics." In: *Journal of Intelligent & Robotic Systems* 86.2, pp. 153–173.

Pong, V, S Gu, M Dalal, and S Levine (2018). "Temporal Difference Models: Model-Free Deep RL for Model-Based Control." In: *International Conference on Learning Representations (ICLR 2018)*. OpenReview.net.

Ponsen, Marc, Matthew E Taylor, and Karl Tuyls (2009). "Abstraction and generalization in reinforcement learning: A summary and framework." In: *International Workshop on Adaptive and Learning Agents*. Springer, pp. 1–32.

Precup, Doina (2000). "Eligibility traces for off-policy policy evaluation." In: *Computer Science Department Faculty Publication Series*, p. 80.

Pritzel, Alexander, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell (2017). "Neural Episodic Control." In: *International Conference on Machine Learning*, pp. 2827–2836.

Puterman, Martin L (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Racanière, Sébastien, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. (2017). "Imagination-augmented agents for deep reinforcement learning." In: *Advances in neural information processing systems*, pp. 5690–5701.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational inference with normalizing flows." In: *arXiv preprint arXiv:1505.05770*.

Richalet, Jacques, André Rault, JL Testud, and J Papon (1978). "Model predictive heuristic control." In: *Automatica (Journal of IFAC)* 14.5, pp. 413–428.

Riemer, Matthew, Miao Liu, and Gerald Tesauro (2018). "Learning abstract options." In: *Advances in Neural Information Processing Systems*, pp. 10424–10434.

Roijers, Diederik M, Peter Vamplew, Shimon Whiteson, and Richard Dazeley (2013). "A survey of multi-objective sequential decision-making." In: *Journal of Artificial Intelligence Research* 48, pp. 67–113.

Roijers, Diederik M and Shimon Whiteson (2017). "Multi-objective decision making." In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 11.1, pp. 1–129.

Rollier, Bruce and Jon A Turner (1994). "Planning forward by looking backward: Retrospective thinking in strategic decision making." In: *Decision Sciences* 25.2, pp. 169–188.

Rosin, Christopher D (2011). "Multi-armed bandits with episode context." In: *Annals of Mathematics and Artificial Intelligence* 61.3, pp. 203–230.

Rubinstein, Reuven Y and Dirk P Kroese (2013). *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media.

Rummery, Gavin A and Mahesan Niranjan (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England.

Russell, Stuart J and Peter Norvig (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

Saffiotti, Alessandro, Kurt Konolige, and Enrique H Ruspini (1995). "A multivalued logic approach to integrating planning and control." In: *Artificial intelligence* 76.1-2, pp. 481–526.

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever (2017). "Evolution strategies as a scalable alternative to reinforcement learning." In: *arXiv preprint arXiv:1703.03864*.

Samuel, Arthur L (1967). "Some studies in machine learning using the game of checkers. II - Recent progress." In: *IBM Journal of research and development* 11.6, pp. 601–617.

Sawada, Yoshihide (2018). "Disentangling Controllable and Uncontrollable Factors of Variation by Interacting with the World." In: *arXiv preprint arXiv:1804.06955*.

Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015). "Universal value function approximators." In: *International Conference on Machine Learning*, pp. 1312–1320.

Schmidhuber, Jürgen (1991a). "A possibility for implementing curiosity and boredom in model-building neural controllers." In: *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227.

– (1991b). "Curious model-building control systems." In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. IEEE, pp. 1458–1463.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lock-

hart, Demis Hassabis, Thore Graepel, et al. (2019). "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model." In: *arXiv preprint arXiv:1911.08265*.

Schulman, John, Xi Chen, and Pieter Abbeel (2017). "Equivalence between policy gradients and soft q-learning." In: *arXiv preprint arXiv:1704.06440*.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). "Trust region policy optimization." In: *International conference on machine learning*, pp. 1889–1897.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2016). "High-Dimensional Continuous Control Using Generalized Advantage Estimation." In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347*.

Schwartz, Barry, Andrew Ward, John Monterosso, Sonja Lyubomirsky, Katherine White, and Darrin R Lehman (2002). "Maximizing versus satisficing: Happiness is a matter of choice." In: *Journal of personality and social psychology* 83.5, p. 1178.

Sekar, Ramanan, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak (2020). "Planning to Explore via Self-Supervised World Models." In: *arXiv preprint arXiv:2005.05960*.

Sequeira, Pedro, Francisco S Melo, and Ana Paiva (2014). "Learning by appraising: an emotion-based approach to intrinsic reward design." In: *Adaptive Behavior* 22.5, pp. 330–349.

Sermanet, Pierre, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain (2018). "Time-contrastive networks: Self-supervised learning from video." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1134–1141.

Shalev-Shwartz, Shai, Shaked Shammah, and Amnon Shashua (2016). "Safe, multi-agent, reinforcement learning for autonomous driving." In: *arXiv preprint arXiv:1610.03295*.

Sharma, Archit, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman (2019). "Dynamics-Aware Unsupervised Discovery of Skills." In: *International Conference on Learning Representations*.

Sheppard, Brian (2002). "World-championship-caliber Scrabble." In: *Artificial Intelligence* 134.1-2, pp. 241–275.

Shu, Tianmin, Caiming Xiong, and Richard Socher (2017). "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning." In: *arXiv preprint arXiv:1712.07294*.

Shyam, Pranav, Wojciech Jaśkowski, and Faustino Gomez (2019). "Model-Based Active Exploration." In: *International Conference on Machine Learning*, pp. 5779–5788.

Sigaud, Olivier, Camille Salaün, and Vincent Padois (2011). "On-line regression algorithms for learning mechanical models of robots: a survey." In: *Robotics and Autonomous Systems* 59.12, pp. 1115–1129.

Silver, David, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. (2017a). "The predictron: End-to-end learning and planning." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 3191–3199.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587, p. 484.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2017b). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." In: *arXiv preprint arXiv:1712.01815*.

– (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419, pp. 1140–1144.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). "Deterministic Policy Gradient Algorithms." In: *International Conference on Machine Learning*, pp. 387–395.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017c). "Mastering the game of go without human knowledge." In: *Nature* 550.7676, p. 354.

Silver, David, Richard S Sutton, and Martin Müller (2008). "Sample-based learning and search with permanent and transient memories." In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 968–975.

Silver, David and Joel Veness (2010). "Monte-Carlo planning in large POMDPs." In: *Advances in neural information processing systems*, pp. 2164–2172.

Simon, Herbert A (1944). "Decision-making and administrative organization." In: *Public Administration Review* 4.1, pp. 16–30.

– (1957). "Models of man; social and rational." In:

Şimşek, Özgür, Alicia P Wolfe, and Andrew G Barto (2005). "Identifying useful subgoals in reinforcement learning by local graph partitioning." In: *Proceedings of the 22nd international conference on Machine learning*. ACM, pp. 816–823.

Singh, Satinder P, Tommi Jaakkola, and Michael I Jordan (1995). "Reinforcement learning with soft state aggregation." In: *Advances in neural information processing systems*, pp. 361–368.

Skinner, Burrhus F (1937). "Two types of conditioned reflex: A reply to Konorski and Miller." In: *The Journal of General Psychology* 16.1, pp. 272–279.

Sohn, Kihyuk, Honglak Lee, and Xinchen Yan (2015). "Learning structured output representation using deep conditional generative models." In: *Advances in Neural Information Processing Systems*, pp. 3483–3491.

Sønderby, Casper Kaae, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther (2016). "Ladder variational autoencoders." In: *Advances in Neural Information Processing Systems*, pp. 3738–3746.

Spaan, Matthijs TJ and Nikos Vlassis (2004). "A point-based POMDP algorithm for robot planning." In: *Proc. IEEE Int. Conf. on Robotics and Automation, New Orleans, Louisiana*, pp. 2399–2404.

Spelke, Elizabeth S and Katherine D Kinzler (2007). "Core knowledge." In: *Developmental science* 10.1, pp. 89–96.

Srinivas, Aravind, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). "Universal planning networks." In: *arXiv preprint arXiv:1804.00645*.

Stadie, Bradly C, Sergey Levine, and Pieter Abbeel (2015). "Incentivizing exploration in reinforcement learning with deep predictive models." In: *arXiv preprint arXiv:1507.00814*.

Stigler, George J (1950). "The development of utility theory. I." In: *Journal of political economy* 58.4, pp. 307–327.

Such, Felipe Petroski, Vashisht Madhavan, Rosanne Liu, Rui Wang, Pablo Samuel Castro, Yulun Li, Jiale Zhi, Ludwig Schubert, Marc G Bellemare, Jeff Clune, et al. (2018). "An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents." In: *arXiv preprint arXiv:1812.07069*.

Sun, Ron (2008). *The Cambridge handbook of computational psychology*. Cambridge University Press.

Sun, Yi, Faustino Gomez, and Jürgen Schmidhuber (2011). "Planning to be surprised: Optimal bayesian exploration in dynamic environments." In: *International Conference on Artificial General Intelligence*. Springer, pp. 41–51.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences." In: *Machine learning* 3.1, pp. 9–44.

– (1990). "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming." In: *Machine Learning Proceedings 1990*. Elsevier, pp. 216–224.

– (1991). "Dyna, an integrated architecture for learning, planning, and reacting." In: *ACM Sigart Bulletin* 2.4, pp. 160–163.

– (1996). "Generalization in reinforcement learning: Successful examples using sparse coarse coding." In: *Advances in neural information processing systems*, pp. 1038–1044.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, David A McAllester, Satinder P Singh, and Yishay Mansour (2000). "Policy gradient methods for reinforcement learning with function approximation." In: *Advances in neural information processing systems*, pp. 1057–1063.

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning." In: *Artificial intelligence* 112.1-2, pp. 181–211.

Sutton, Richard S, Csaba Szepesvári, Alborz Geramifard, and Michael P Bowling (2012). "Dyna-style planning with linear function approximation and prioritized sweeping." In: *arXiv preprint arXiv:1206.3285*.

Sutton, Richard S., Csaba Szepesvári, Alborz Geramifard, and Michael Bowling (2008). "Dyna-style Planning with Linear Function Approximation and Prioritized Sweeping." In: *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*. UAI'08. Arlington, Virginia, United States: AUAI Press, pp. 528–536. ISBN: 0-9749039-4-9.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Tadepalli, Prasad, Robert Givan, and Kurt Driessens (2004). "Relational reinforcement learning: An overview." In: *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, pp. 1–9.

Talvitie, Erik (2014). "Model Regularization for Stable Sample Rollouts." In: *UAI*, pp. 780–789.

– (2017). "Self-correcting models for model-based reinforcement learning." In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel (2016). "Value iteration networks." In: *Advances in Neural Information Processing Systems*, pp. 2154–2162.

Tang, Yunhao and Shipra Agrawal (2018). "Exploration by distributional reinforcement learning." In: *arXiv preprint arXiv:1805.01907*.

Taylor, Matthew E and Peter Stone (2009). "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.Jul, pp. 1633–1685.

Tesauro, Gerald and Gregory R. Galperin (1997). "On-line Policy Improvement using Monte-Carlo Search." In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, pp. 1068–1074.

Tesauro, Gerald, VT Rajan, and Richard Segal (2012). "Bayesian inference in monte-carlo tree search." In: *arXiv preprint arXiv:1203.3519*.

Tessler, Chen, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor (2017). "A deep hierarchical approach to lifelong learning in minecraft." In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Thomas, Valentin, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio (2018). "Disentangling the independently controllable factors of variation by interacting with the world." In: *arXiv preprint arXiv:1802.09484*.

Thomas, YA (1975). "Linear quadratic optimal estimation and control with receding horizon." In: *Electronics Letters* 11.1, pp. 19–21.

Thompson, William R (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples." In: *Biometrika* 25.3/4, pp. 285–294.

Thrun, Sebastian B (1992). "Efficient exploration in reinforcement learning." In:

Thrun, Sebastian and Anton Schwartz (1995). "Finding structure in reinforcement learning." In: *Advances in neural information processing systems*, pp. 385–392.

Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017*

IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, pp. 23–30.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control." In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, pp. 5026–5033.

Todorov, Emanuel and Weiwei Li (2005). "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems." In: *Proceedings of the 2005, American Control Conference, 2005.* IEEE, pp. 300–306.

Tolman, Edward C (1948). "Cognitive maps in rats and men." In: *Psychological review* 55.4, p. 189.

Toussaint, Marc (2009). "Robot trajectory optimization using approximate inference." In: *Proceedings of the 26th annual international conference on machine learning*. ACM, pp. 1049–1056.

Van Hoof, Herke, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters (2016). "Stable reinforcement learning with autoencoders for tactile and visual data." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3928–3934.

Van Otterlo, Martijn (2005). "A survey of reinforcement learning in relational domains." In: *Centre for Telematics and Information Technology (CTIT) University of Twente, Tech. Rep*.

Van Seijen, Harm, Hadi Nekoei, Evan Racah, and Sarath Chandar (2020). "The LoCA Regret: A Consistent Metric to Evaluate Model-Based Behavior in Reinforcement Learning." In: *Advances in Neural Information Processing Systems* 33.

Van Seijen, Harm, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering (2009). "A theoretical and empirical analysis of Expected Sarsa." In: *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, pp. 177–184.

Van Steenkiste, Sjoerd, Michael Chang, Klaus Greff, and Jürgen Schmidhuber (2018). "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions." In: *arXiv preprint arXiv:1802.10353*.

Vanschoren, Joaquin (2019). "Meta-learning." In: *Automated Machine Learning*. Springer, Cham, pp. 35–61.

Vanseijen, Harm and Rich Sutton (2015). "A deeper look at planning as learning from replay." In: *International conference on machine learning*, pp. 2314–2322.

Veness, Joel, David Silver, Alan Blair, and William Uther (2009). "Bootstrapping from game tree search." In: *Advances in neural information processing systems*, pp. 1937–1945.

Venkatraman, Arun, Martial Hebert, and J Andrew Bagnell (2015). "Improving multi-step prediction of learned time series models." In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Vezhnevets, Alexander Sasha, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu (2017). "Feudal networks for hierarchical reinforcement learning." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 3540–3549.

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature*, pp. 1–5.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. (2016). "Matching networks for one shot learning." In: *Advances in neural information processing systems*, pp. 3630–3638.

Waa, Jasper van der, Jurriaan van Diggelen, Karel van den Bosch, and Mark Neerincx (2018). "Contrastive explanations for reinforcement learning in terms of expected consequences." In: *arXiv preprint arXiv:1807.08706*.

Wahlström, Niklas, Thomas B Schön, and Marc Peter Deisenroth (2015). "From pixels to torques: Policy learning with deep dynamical models." In: *arXiv preprint arXiv:1502.02251*.

Walker, Jacob, Carl Doersch, Abhinav Gupta, and Martial Hebert (2016). "An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders." In: *European Conference on Computer Vision*. Springer, pp. 835–851.

Wang, Jack, Aaron Hertzmann, and David J Fleet (2006). "Gaussian process dynamical models." In: *Advances in neural information processing systems*, pp. 1441–1448.

Wang, Tingwu, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba (2019). "Benchmarking Model-Based Reinforcement Learning." In: *CoRR* abs/1907.02057.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning." In: *Machine learning* 8.3-4, pp. 279–292.

Watson, John S (1966). "The development and generalization of" contingency awareness" in early infancy: Some hypotheses." In: *Merrill-Palmer Quarterly of Behavior and Development* 12.2, pp. 123–135.

Watter, Manuel, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller (2015). "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Advances in neural information processing systems*, pp. 2746–2754.

Watters, Nicholas, Loic Matthey, Matko Bosnjak, Christopher P Burgess, and Alexander Lerchner (2019). "Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration." In: *arXiv preprint arXiv:1905.09275*.

Werbos, Paul J (1989). "Neural networks for control and system identification." In: *Proceedings of the 28th IEEE Conference on Decision and Control,* IEEE, pp. 260–265.

White, Robert W (1959). "Motivation reconsidered: The concept of competence." In: *Psychological review* 66.5, p. 297.

Whiteson, Shimon and Peter Stone (2006). "Evolutionary function approximation for reinforcement learning." In: *Journal of Machine Learning Research* 7.May, pp. 877–917.

Wiering, Marco A, Maikel Withagen, and Mădălina M Drugan (2014). "Model-based multi-objective reinforcement learning." In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, pp. 1–6.

Wiering, Marco and Jürgen Schmidhuber (1998). "Efficient model-based exploration." In: *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*. Vol. 6, pp. 223–228.

Wiese, Jessica, Roger Buehler, and Dale Griffin (2016). "Backward planning: Effects of planning direction on predictions of task completion time." In: *Judgment and Decision Making* 11.2, p. 147.

Willemsen, Daniel, Hendrik Baier, and Michael Kaisers (2020). "Value targets in off-policy AlphaZero: a new greedy backup." In: *Adaptive and Learning Agents (ALA) Workshop*.

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Reinforcement Learning*. Springer, pp. 5–32.

Wilson, Aaron, Alan Fern, Soumya Ray, and Prasad Tadepalli (2007). "Multi-task reinforcement learning: a hierarchical Bayesian approach." In: *Proceedings of the 24th international conference on Machine learning*. ACM, pp. 1015–1022.

Winands, Mark HM, Yngvi Björnsson, and Jahn-Takeshi Saito (2008). "Monte-Carlo tree search solver." In: *International Conference on Computers and Games*. Springer, pp. 25–36.

Wolpert, Daniel M, Zoubin Ghahramani, and Michael I Jordan (1995). "An internal model for sensorimotor integration." In: *Science* 269.5232, pp. 1880–1882.

Wu, Jiajun, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum (2015). "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning." In: *Advances in neural information processing systems* 28, pp. 127–135.

Xu, Zhenjia, Zhijian Liu, Chen Sun, Kevin Murphy, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu (2019). "Unsupervised discovery of parts, structure, and dynamics." In: *arXiv preprint arXiv:1903.05136*.

Yamaguchi, Tomohiro, Shota Nagahama, Yoshihiro Ichikawa, and Keiki Takadama (2019). "Model-Based Multi-objective Reinforcement Learning with Unknown Weights." In: *International Conference on Human-Computer Interaction*. Springer, pp. 311–321.

Yoon, Sung Wook, Alan Fern, and Robert Givan (2007). "FF-Replan: A Baseline for Probabilistic Planning." In: *ICAPS*. Vol. 7, pp. 352–359.

Younes, Håkan LS and Michael L Littman (2004). "PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects." In: *Techn. Rep. CMU-CS-04-162* 2, p. 99.

Yu, Lantao, Weinan Zhang, Jun Wang, and Yong Yu (2017). "Seqgan: Sequence generative adversarial nets with policy gradient." In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Zhang, Marvin, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine (2019). "SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning." In: *International Conference on Machine Learning*, pp. 7444–7453.

Zhu, Zhuangdi, Kaixiang Lin, and Jiayu Zhou (2020). "Transfer Learning in Deep Reinforcement Learning: A Survey." In: *arXiv preprint arXiv:2009.07888*.

Ziegler, Zachary M and Alexander M Rush (2019). "Latent normalizing flows for discrete sequences." In: *arXiv preprint arXiv:1901.10548*.

SUMMARY

Sequential decision making, commonly formalized as Markov Decision Process (MDP) optimization, is a key challenge in artificial intelligence. Two successful approaches to solve this problem are *planning*, a classic subfield in artificial intelligence research, and *reinforcement learning*, a subfield of machine learning. The key difference between planning and learning is whether a model of the environment dynamics is known (planning) or unknown (reinforcement learning).

Planning and learning may actually be combined, in a field which is better known as *model-based reinforcement learning*. Model-based reinforcement learning has recently shown important empirical success, for example defeating the human world champion in the classic boardgame Go, a long-standing challenge in AI research. This emphasizes the importance of a combined planning and learning approach.

While there is communication between the research fields of planning and learning, they still largely operate as separate communities, each with their own methodology. More important, literature still lacks a systematic view on the relation between both fields, and how these may be combined. This is the main topic of this dissertation. We discuss two main questions: 1) how are planning and learning related (what are the similarities and differences between their algorithms), and 2) how can planning and learning be combined (and what advantage may this provide)?

The first half of this thesis provides a conceptual answer to both questions, where Chapter 3 discusses the first research question, and Chapter 4 the second. Chapter 3 introduces the Framework for Reinforcement Learning and Planning (FRAP). FRAP shows that both approaches share exactly the same algorithmic design space. We disentangle this space into six key dimensions: 1) for which states do we seek a solution, 2) how do we select the next trial in the environment, 3) how do we estimate the cumulative reward after the trial, 4) how do we back-up the acquired information to the begin state of the trial, 5) how do we store the solution, and 6) how do we update the solution based on the new back-up estimate obtained in step 4. At the end of the chapter, we compare a variety of planning and learning algorithms along the dimensions of our framework. The table reads like a patchwork, where similar solutions appear throughout both planning and learning algorithms.

Altogether, FRAP shows that the lines between both fields are blurry, and usually based on convention. On a fundamental level, they face the same challenges and possible solutions. As such, the framework provides a bridge between both research fields, and a fundamental way to categorize algorithms in both fields.

Chapter 4 provides a conceptual answer to the second research question. Learning can be added to planning in two ways: 1) learning a model of the transition dynamics, and 2) learning a value or policy function to approximate the global solution to the MDP. Chapter 4 first extensively discusses model learning, focusing on the specific challenges involved. Examples of challenges are environment stochasticity, uncertainty due to limited data, partial observability, non-stationarity, and state and temporal abstraction. The next part of Chapter 4 systematically structures the combination of planning and learning to arrive at a policy or value function. We again dissect the possible combinations into several key questions, like 1) where to start planning, 2) what planning budgets to allow for, 3) how to actually plan, and 4) how to integrate the plan into the larger learning and acting loop. The last part of Chapter 4 discusses the possible benefits of the combination of planning and learning: 1) enhanced data efficiency (lower sample complexity in the real world), 2) targeted (and safe) exploration, 3) improved training stability, 4) better transfer between tasks, and 5) explainability.

There appears to be a tension between Chapters 3 and 4. Whereas Chapter 3 argues that planning and learning essentially do the same thing, Chapter 4 shows that it may still be beneficial to combine both. There are two main reasons why planning should be added to learning. The first benefit originates from the difference between the real world and internal simulation. Internal simulation (planning) is much safer, and allows us to have fewer interactions with the real world (which can be dangerous and/or time-consuming). The second benefit originates from a convention difference in the type of representation used in both fields. Learning algorithms use global approximation of policy and value functions, which will inevitably make approximation errors. The local tabular representation of planning can locally correct these errors. During learning planning can thereby provide more stable back-ups, but even after the global solution has converged, it will often still contain approximation errors (due to capacity constraints), and planning may still be beneficial. We further develop this idea in Chapter 7. Altogether, Chapters 3 and 4 provide a systematic view on 1) the relation between planning and learning, and 2) the potential ways to combine both. As such, it forms the core part of this thesis.

The second half of the dissertation provides experimental illustration of the first conceptual half. We present four papers, each of which deal with a different aspect of the above conceptual ideas. The first three papers study new ways in which planning and learning can be combined. Chapter 5 presents a new method to learn a dynamics model in the context of stochasticity in the environment, which is usually present in real-world environments. To accurately predict the possible futures in the presence of multimodal stochasticity, we require models that flexibly approximate distributions in high-dimensional spaces. We present a novel model learning method based on conditional variational inference in neural networks, which does allow for these properties. This is an important preliminary to model-based reinforcement learning in stochastic environments.

Chapter 6 studies a novel way to combine planning and learning. We built upon a successful previous algorithm, AlphaGo Zero, which combined planning (through search trees) and learning (through deep neural networks) to achieve superhuman performance in the game of Go. However, board games like Go have a discrete action space, for which the algorithm was designed as well. Real-world problems frequently have a continuous action space, like many robotics tasks. Chapter 6 extends the successful AlphaGo Zero algorithm to also work in continuous action spaces. With a different action space, we also have a different representation of our policy network, which requires a different way of training (planning to learning influence) and a different way to bias new tree searches (learning to planning influence). As such, Chapter 6 illustrates a new way to integrate planning and learning.

In Chapter 7 we take a deeper look at a specific benefit of the combination of planning and learning. Our hypothesis is that planning and learning provide mutual benefit, due to the different ways in which they represent their solutions. Learning allows us to store a global solution to the entire problem, and generalize information between similar states. However, learned predictions will have approximation errors. The tabular representation of planning allows for exact local separation of different futures, but we cannot afford to exhaustively solve the entire planning problem, which makes it hard to scale the approach to large problems. We hypothesize that the combination of both provides the best of both worlds, since local planning can locally correct the errors made in the function approximation. For example, in a game of Chess, the current values of all available actions may be hard to distinguish, but a few steps ahead the pay-off of certain futures might be much more clear. We experimentally validate this idea in a

planning-learning integration, in which we vary the planning budget per timestep. On one extreme we do not plan at all (model-free RL), and on the other extreme we extensively plan at every timestep (up to exhaustive search). Our experiments show that both approaches are suboptimal, and the best result is achieved when we make a moderate planning effort per timestep. The identification of this trade-off was not identified before as crucial to the performance of these algorithms, and could be an important future research direction.

After these three experimental illustrations of the second research question (the combination of planning and learning), we finish the experimental section with an illustration of research question one (the commonalities between planning and learning). If both approaches really make the same algorithmic choices, then we should be able to design a new algorithm in one field based on inspiration from the other. Chapter 8 presents a new planning algorithm, MCTS-T+, which extends the well-known planning algorithm Monte Carlo Tree Search (MCTS). We took inspiration from the reinforcement learning literature, which has frequently studied sparse exploration problems in problem with asymmetry in the MDP tree. Sparse reward tasks are challenging for exploration, since we do not get any indication of the task goal for a long time. We show that standard MCTS performs strongly suboptimal on some well-known example tasks from the reinforcement learning community. The problem is that MCTS only weights how often a particular action was tried (the classical statistical uncertainty), but ignore how many different futures are possible after that action. If there are many possible future, then the uncertainty should decrease more slowly, and vice versa. Our algorithm, MCTS-T+, describes a way to heuristically estimate this second type of uncertainty, and as such reaches much higher planning efficiency in these tasks with asymmetry and sparse rewards.

The dissertation concludes in Chapter 9 with a broad discussion of the research field of planning and learning. We summarize where the field stands, what this dissertation adds, and extensively highlight promising directions for future work in model-based RL. In short, the combination of planning and learning promises to be key research field in AI in the forthcoming years. This dissertation presents a broad conceptual view on the individual fields of planning and learning, their similarities, and the possible ways in which both can be combined. In the end, planning and learning are complementary phenomena.

SAMENVATTING

Een kernprobleem in de kunstmatige intelligentie (*artificial intelligence*, AI) is het kunnen maken van een serie intelligente beslissingen. Zo'n sequentieel beslisprobleem kan wiskundig beschreven worden als een Markoviaans Beslissingsprobleem (*Markov Decision Process, MDP*). De beslissingen in zo'n MDP worden genomen door een kunstmatige *agent*. Zo'n agent kan een robot zijn, of een zelfrijdende auto, een virtuele tegenstander in een bordspel, of eigenlijk elke kunstmatige entiteit die een serie van beslissingen moet maken.

Op elke tijdstap in het probleem observeert de agent de huidige staat van de omgeving, op basis van sensorische input. Vervolgens kiest de agent een actie en voert deze uit. Op de volgende tijdstap observeert de agent het effect hiervan. Allereerst ziet hij de nieuwe staat van de omgeving, die verandert kan zijn door de actie, en daarnaast krijgt de agent een beloning of straf, afhankelijk van hoe goed of slecht de uitkomst van de actie was. Vervolgens herhaalt het proces zich, waarbij de agent opnieuw een actie kiest.

Elke beloning of straf is voor de agent simpelweg een getal. Vaak wordt een positief getal voor een beloning gebruikt, en een negatief getal voor een straf. Het doel van de AI agent is om de optelsom van alle beloning en straf over alle tijdstappen tesamen zo positief mogelijk te maken. Het is cruciaal dat het hier om de optelsom van alle beloningen gaat. In veel taken moeten we immers eerst moeite investeren (kleine straffen) om uiteindelijk een grotere beloning te verwerven.

Het bovenstaande framework is generiek en toepasbaar op feitelijk alle soorten beslisproblemen. Verschillende onderzoeksgemeenschappen hebben daarom over de jaren oplossingen voor dit probleem gezocht. In de klassieke kunstmatige intelligentie is er vooral gekeken naar een groep van algoritmes die bekend staan als 'plannen'. Bij plannen krijgen we een *model* van de omgeving, wat ons vertelt hoe de omgeving zal veranderen door een bepaalde actie. Belangrijk is dat we een model ook terug kunnen spoelen, en dan een andere actie vanaf een bepaalde staat kunnen proberen. Op die manier kunnen we een zoekboom bouwen, waarin we verschillende toekomstige paden proberen om uiteindelijk tot de beste beslissing te komen. Dit valt te vergelijken met de manier waarop mensen in gedachten vooruit kunnen plannen.

In de machine learning tak van kunstmatige intelligentie wordt ook al meerdere decennia naar hetzelfde probleem gekeken, onder de term 'reinforcement learning'. Hier wordt echter een leerperspectief aangenomen. We nemen aan dat de omgeving niet meer terug te draaien is, en dat bij elke actie die we uitvoeren we daadwerkelijk verder moeten vanaf de staat die we vervolgens bereiken. Dit valt te vergelijken met hoe we moeten handelen in de echte wereld, waarbij een actie een permanent effect heeft. Omdat we nu niet meer vooruit kunnen plannen, proberen dit soort algoritmes een snelle benadering te leren van de waarde van verschillende acties, op basis van patroonherkenning. Dit kun je vergelijken met reactief, geleerd gedrag.

Mensen gebruiken voor hun beslissingen duidelijk een combinatie van plannen en leren. In de kunstmatige intelligentie hebben beide velden zich echter grotendeels zelfstandig ontwikkeld. Uiteraard is er ook kruisbestuiving geweest, in een veld dat bekend staat als 'model-based reinforcement learning'. Dit veld heeft belangrijke empirische successen laten zien, zoals recentelijk als 's werelds sterkste speler in klassieke bordspellen als Go, Schaken en Shogi. Dit illustreert het belang van de combinatie van beide benaderingen.

Hoewel de plan en leer onderzoeksvelden al geruime tijd communiceren, is er in de literatuur geen systematische blik op de relatie tussen beide velden, en hoe deze zijn te combineren. Dit is daarom het onderwerp van dit proefschrift. We stellen twee hoofdvragen: 1) hoe zijn plannen en leren gerelateerd (wat zijn de overeenkomsten en verschillen in de algoritmische benaderingen), en 2) hoe kunnen plannen en leren worden gecombineerd (en wat voor voordelen kan dit bieden)?

De eerste helft van dit proefschrift geeft een conceptueel antwoord op beide vragen, waarbij hoofdstuk 3 de eerste en hoofdstuk 4 de tweede vraag behandelt. Hoofdstuk 3 introduceert een Framework voor Reinforcement Learning en Plannen (FRAP). FRAP laat zien dat plannen en leren exact dezelfde onderliggende algoritmische beslissingen moeten maken. We snijden een algoritme op in 6 hoofdvragen, elke met meerdere subdimensies. Deze vragen zijn: 1) voor welke staten zoeken we een oplossing, 2) hoe selecteren we de volgende stap in de omgeving, 3) hoe schatten we de som van beloningen na deze stap, 4) hoe verwerken we deze informatie terug naar de staat waar we vandaan kwamen, 5) hoe bewaren we de oplossing voor het probleem, en 6) hoe verbeteren we onze oplossing op basis van de verwerkte informatie uit vraag 4. Aan het eind van het hoofdstuk vergelijken we in een tabel een breed spectrum aan plan- en leeralgoritmes op basis van de dimensies van het framework. De tabel leest als een lappendeken, en toont dat

beide velden grotendeels soortgelijke oplossingen voor dezelfde onderliggende problemen hebben bedacht. Het illustreert daarmee dat de lijnen tussen beide velden methodologisch grotendeels op conventies zijn gebaseerd. Het framework verschaft nu een gemeenschappelijke taal voor beide velden, en een manier om algoritmes in deze velden meer systematisch te categoriseren.

Hoofdstuk 4 geeft een conceptueel overzicht van de manieren waarop plannen en leren te combineren zijn. Leren kan op twee manieren aan plannen toegevoegd worden: 1) leren hoe de omgeving werkt (het *model*) zodat we daarin vooruit kunnen plannen, en 2) leren wat de verwachte waarde (*value*) is van acties (of hoe we op basis daarvan moeten handelen). Wat betreft het eerste punt categoriseert het hoofdstuk uitgebreid de verschillende uitdagingen van het leren van een model van de omgeving. Voorbeelden van uitdagingen zijn onder andere stochasticiteit in de omgeving, onzekerheid door een beperking in de hoeveelheid geobserveerde data, incomplete informatie, en het leren van informatieve abstracties uit complexe, hoog-dimensionale informatie. Ten tweede behandelen we de manieren waarop plannen en leren gecombineerd kunnen worden om een oplossing voor het probleem te vinden. Wederom vergelijken we een brede set literatuur volgens deze stappen in een tabel, wat de diversiteit aan verschillende plan-leer integraties toont. Tot slot toont hoofdstuk 4 ook de manieren waarop de combinatie van plannen en leren voordelen biedt: 1) zodat we minder stappen in de echte wereld hoeven te maken, 2) zodat we beter (en veiliger) kunnen exploreren (nieuwe dingen ontdekken), 3) zodat we stabieler kunnen leren, 4) zodat we beter informatie kunnen hergebruiken tussen taken, en 5) zodat de agent zijn beslissingen beter kan uitleggen aan een mens.

Er lijkt een spanningsveld tussen hoofdstuk 3 en 4 te bestaan. Enerzijds zijn plannen en leren onder de motorkap dus hetzelfde (hoofdstuk 3), anderszijds blijkt het wel degelijk voordelig om beide te combineren. In grote lijnen zijn er twee redenen waarom de combinatie desondanks nuttig is. Het eerste voordeel komt voort uit het onderscheid tussen een echte wereld en interne simulatie. Het intern kunnen simuleren van toekomsten lijkt weliswaar sterk op hoe we leren van de echte wereld, maar het is veel veiliger, en doorgaans veel sneller. Het tweede fundamentele voordeel komt voort uit een conventie in beide velden over de manier waarop ze de oplossing van het probleem opslaan. Leeralgoritmes maken een globale, snelle benadering van de waardes van acties, welke doorgaans benaderingsfouten bevatten. Plannen kan helpen deze lokale fouten te corrigeren. We werken dit idee verder uit

in hoofdstuk 7. Tezamen geven hoofdstuk 3 en 4 een systematische blik op 1) de relatie tussen plannen en leren, en 2) de mogelijke manieren om beide te combineren.

De tweede helft van dit proefschrift is een experimentele illustratie van het conceptuele deel. We bespreken vier papers, die elk een ander aspect van de bovenstaande conceptuele ideeen verder uitwerken. De eerste drie hoofdstukken gaan over nieuwe manieren om plannen en leren te combineren. Hoofdstuk 5 focust op het leren van een model van de omgeving, wanneer de omgeving zich *stochastisch* gedraagt. Stochasticiteit doet zich voor wanneer een bepaalde actie in een huidige staat verschillende toekomsten kan hebben. De echte wereld is in hoge mate stochastisch, bijvoorbeeld door het gedrag van anderen. Als je in een auto rijdt, is het gedrag van andere weggebruikers bijvoorbeeld niet altijd hetzelfde. Het correct leren van dit soort kansverdelingen in grotere problemen is uitdagend, vooral wanneer de kansverdeling massa heeft op veel verschillende toekomsten. We beschrijven een nieuwe methode om neural netwerken te trainen door middel van conditionele variationele inferentie, waarmee we complexe vormen van multimodale stochasticiteit correct kunnen leren. Dit is een belangrijke eerste stap om te kunnen plannen over een geleerd model in stochastische omgevingen, zoals de echte wereld.

In hoofdstuk 6 kijken we vervolgens naar een nieuwe manier om plannen en leren te combineren. We bouwen voort op een bekend recent algoritme, AlphaGo Zero, wat plannen (op basis van zoekbomen) en leren (op basis van diepe neurale netwerken) combineert en zo de menselijke wereldkampioen in het bordspel Go versloeg. In bordspellen zijn de acties discreet, we kunnen immers kiezen uit een eindig aantal zetten. In de echte wereld, bijvoorbeeld in de robotica, zijn acties vaak continu. Op de motoren van een robot kunnen we een voltage zetten, wat varieert tussen een bepaald maximum en minimum, maar daar zitten oneindig veel mogelijke voltages tussen. In hoofdstuk 6 laten we zien hoe het succesvolle AlphaGo Zero algoritme aangepast kan worden zodat het ook werkt in taken met een continue actie ruimte, zoals we tonen op een gesimuleerd robotica experiment.

In hoofdstuk 7 kijken we dieper naar een specifiek voordeel van de combinatie van plannen en leren. Onze hypothese is dat plannen en leren een wederzijds voordeel bieden, door de manier waarop ze hun oplossing representeren. De leerbenadering zorgt dat we een oplossing voor het hele probleem kunnen opslaan, en dat we informatie kunnen delen tussen soortgelijke situaties. De geleerde voorspelling kan echter lokaal fout zijn. Plannen zorgt lokaal voor betere scheiding van de

mogelijke toekomsten, maar we hebben doorgaans niet de rekenkracht of tijd om het gehele probleem door te plannen. Onze hypothese is dat de combinatie van beide een wederzijds voordeel biedt, doordat planning de lokale fouten in geleerde voorspellingen kan corrigeren. We testen dit experimenteel door de mate van planning per tijdstap in een plan-leer algoritme te varieren. In het ene extreme geval wordt er helemaal niet gepland, en in het andere extreme geval plannen we elke tijdstap zeer uitgebreid. We laten experimenteel zien dat beide benaderingen suboptimaal zijn, en dat het beste resultaat wordt bereikt als we *een beetje* plannen per tijdstap. Deze trade-off tussen plannen en leren stond nog niet op de onderzoeksradar, maar blijkt cruciaal te zijn voor het resultaat van het algoritme. Pschologisch onderzoek verkende al soortgelijke ideeen, en toonde dat mensen adaptief beslissen wanneer ze starten met plannen, en voor hoe lang. Dit kan een belangrijke richting zijn voor toekomstig werk in dit veld.

Na deze drie illustraties van de combinatie van plannen en leren (onderzoeksvraag 2), sluiten we in hoofdstuk 8 af met een illustratie van de overeenkomsten tussen plannen en leren (onderzoeksvraag 1). Als beide benaderingen onderliggend dezelfde keuzes maken, dan moet het mogelijk zijn om een nieuw algoritme in een onderzoeksveld te ontwerpen gebaseerd op inspiratie uit het andere onderzoeksveld. In hoofdstuk 8 beschrijven we een nieuw plan algoritme, MCTS-T+, wat een uitbreiding is van het succesvolle plan algoritme Monte Carlo Tree Search (MCTS). We halen hiervoor inspiratie uit de reinforcement learning literatuur, met name van werk over exploratie wanneer de beloningen schaars zijn. Het is uitdagend om een taak met schaarse beloning op te lossen, omdat we lange tijd geen indicatie krijgen wat een goede uitkomst is. We laten zien dat standaard MCTS zeer slecht presteert in een aantal van dit soort taken met schaarse beloning. Het probleem is dat MCTS wel meeweegt hoe vaak een actie is geprobeerd (de standaard statisische onzekerheid), maar negeert hoeveel mogelijke toekomsten er op een actie kunnen volgen. Als een actie veel verschillende toekomsten heeft, moet onze onzekerheid langzamer afnemen. Ons algoritme, MCTS-T+, beschrijft een manier om deze onzekerheden heuristisch te schatten, en komt zo tot veel grotere plan efficientie.

Het boek sluit af met een overkoepelende bespreking van het onderzoeksveld van plannen en leren (hoofdstuk 9). We kijken waar het veld staat, wat dit proefschrift daar aan toevoegt, en besteden uitgebreid aandacht aan mogelijke toekomstige onderzoeksrichtingen. De combinatie van plannen en leren belooft in de komende jaren een cruciaal onderzoeksveld in de AI te worden. Dit proefschrift biedt een brede

conceptuele blik op de individuele velden van plannen en leren, hun overeenkomsten, en de manieren waarop beide gecombineerd kunnen worden. Uiteindelijk zijn plannen en leren complementaire fenomenen.

ACKNOWLEDGMENTS

CURRICULUM VITAE

Thomas Marinus Moerland (1988)

EDUCATION

2006-2014 **Medical Doctor (MD)**, *Leiden University, The Netherlands*
Propedeuse - cum laude
Doctoraal - cum laude
Artsexamen - cum laude
Final rotation: Neurology department, Diaconessenhuis Leiden

2010-2015 **Master of Science (MSc), Mathematics**, *Leiden University, The Netherlands*
Summa cum laude
Specialization: Machine learning
Master's thesis: Vision-based Robotics, TU Delft

2016-2020 **Doctor of Philosophy (PhD), Computer Science**, *TU Delft, The Netherlands*
Topic: Intersection of planning and learning.

AWARDS & GRANTS

2018 Qualcomm PhD fellowship

2018 Best reviewer award, Conference on Neural Information Processing Systems

2016 Jan Hemelrijk Award, best Dutch master's thesis in statistics of 2016

2009-2011 Excellent student traject, Pulmonology department, Leiden University Medical Center

LIST OF PUBLICATIONS

- Moerland TM, Broekens J, Jonker CM. A Framework for Reinforcement Learning and Planning. *In submission*. 2020.
- Moerland TM, Broekens J, Jonker CM. Model-based Reinforcement Learning: A Survey. *In submission*. 2020.
- Moerland TM, Broekens J, Jonker CM. Model-based Reinforcement Learning: A Compressed Survey. Deep Reinforcement Learning Workshop @ NeurIPS. 2020.
- Moerland TM, Broekens J, Jonker CM. A Framework for Reinforcement Learning and Planning: Extended Abstract. Bridging the Gap Between AI Planning and Reinforcement Learning (PRL) workshop @ ICAPS. 2020.
- Moerland TM, Broekens J, Jonker CM. Think Too Fast Nor Too Slow: The computational trade-off between planning and reinforcement learning. Bridging the Gap Between AI Planning and Reinforcement Learning (PRL) workshop @ ICAPS. 2020.
- Moerland TM, Broekens J, Plaat A, Jonker CM. AoC: Alpha Zero in Continuous Action Space. European Workshop on Reinforcement Learning. 2018.
- Moerland TM, Broekens J, Plaat A, Jonker CM. Monte Carlo Tree Search for Asymmetric Trees. 2018. Planning and Learning (PAL) Workshop @ ICML. 2018.
- Moerland TM, Broekens J, Jonker CM. The Potential of the Return Distribution for Exploration in RL. 2018. Exploration in Reinforcement Learning Workshop @ ICML. 2018.
- W. J. Wolfslag, M. Bharatheesha, T. M. Moerland and M. Wisse. RRT-CoLearn: Towards Kinodynamic Planning Without Numerical Trajectory Optimization. IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 1655-1662. 2018.

- Wolfslag WJ, Bharatheesha M, Moerland TM, Wisse M. RRT-CoLearn: towards kinodynamic planning without numerical trajectory optimization. International Conference on Robotics and Automation (ICRA). 2018.
- Moerland TM, Broekens J, Jonker CM. Emotion in Reinforcement Learning Agents and Robots: A Survey. Machine Learning 107(2), 443-480. 2017.
- Moerland TM, Broekens J, Jonker CM. Efficient Exploration with Double Uncertain Value Networks. Deep Reinforcement Learning Symposium @ Conference on Neural Information Processing Systems (NIPS). 2017.
- Moerland TM, Broekens J, Jonker CM. Learning Multimodal Transition Dynamics for Model-Based Reinforcement Learning. Scaling Up Reinforcement Learning (SURL) Workshop @ European Conference on Machine Learning (ECML). 2017.
- Bharatheesha M, Wolfslag W, Moerland TM. A dataset bias problem for learning-RRT, with two potential solutions. Delft Workshop on Robot Learning. 2017. (Extended abstract)
- Wolfslag WJ, Bharatheesa M, Moerland TM, Wisse M. Learning indirect optimal control for dynamic motion planning with RRT. Dynamic Walking. 2017. (Extended abstract)
- Moerland TM, Broekens J, Jonker CM. Fear and Hope Emerge From Anticipation in Model-Based Reinforcement Learning. International Joint Conference on Artificial Intelligence (IJCAI). 2016.
- Moerland TM, Chandarr A, Rudinac M and Jonker P. Knowing What You Don't Know: Novelty Detection for Action Recognition in Personal Robots. VISAPP. 2016.