Model-based Reinforcement Learning: A Survey

Suggested Citation: Thomas M. Moerland, Joost Broekens, Aske Plaat and Catholijn M. Jonker (2018), "Model-based Reinforcement Learning: A Survey", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Thomas M. Moerland
LIACS, Leiden University, The Netherlands
t.m.moerland@liacs.leidenuniv.nl

Joost Broekens
LIACS, Leiden University, The Netherlands

Aske Plaat
LIACS, Leiden University, The Netherlands

Catholijn M. Jonker
Interactive Intelligence, TU Delft, The Netherlands
LIACS, Leiden University, The Netherlands

now
the essence of knowledge
Boston — Delft

Contents

Model-based Reinforcement Learning: A Survey

Thomas M. Moerland[1], Joost Broekens[1], Aske Plaat[1] and Catholijn M. Jonker[1,2]

[1]*LIACS, Leiden University, The Netherlands*
[2]*Interactive Intelligence, TU Delft, The Netherlands*

ABSTRACT

Sequential decision making, commonly formalized as Markov Decision Process (MDP) optimization, is a important challenge in artificial intelligence. Two key approaches to this problem are reinforcement learning (RL) and planning. This paper 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 sections, 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. Along the way, the survey also draws connections to several related RL fields, like hierarchical RL and transfer learning. Altogether, the survey presents a

Thomas M. Moerland, Joost Broekens, Aske Plaat and Catholijn M. Jonker (2018), "Model-based Reinforcement Learning: A Survey", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

broad conceptual overview of the combination of planning and learning for MDP optimization.

1

Introduction

Sequential decision making, commonly formalized as Markov Decision Process (MDP) (Bellman, 1954; Puterman, 2014) optimization, is a key challenge in artificial intelligence. Two successful approaches to solve this problem are *planning* (Russell and Norvig, 2016; Bertsekas *et al.*, 1995) and *reinforcement learning* (Sutton and Barto, 2018). Planning and learning may actually be combined, in a field which is known as *model-based reinforcement learning*. We define model-based RL as: 'any MDP approach that i) uses a model (known or learned) and ii) uses learning to approximate a global value or policy function'.

While model-based RL has shown great success (Silver *et al.*, 2017b; Levine and Koltun, 2013; Deisenroth and Rasmussen, 2011), literature lacks a systematic review of the field (although Hamrick *et al.* (2020) does provide an overview of mental simulation in deep learning, see Sec. 9 for a detailed discussion of related work). Therefore, this article presents a survey of the combination of planning and learning. A general scheme of the possible connections between planning and learning, which we will use throughout the survey, is shown in Figure 1.1.

The survey is organized as follows. After a short introduction of the MDP optimization problem (Sec. 2), we first define the categories of

model-based reinforcement learning and their relation to the fields of planning and model-free reinforcement learning (Sec. 3). Afterwards, Sections 4-7 present the main body of this survey. The crucial first step of most model-based RL algorithms is *dynamics model learning* (Fig 1.1, arrow g) which we cover in Sec. 4. When we have obtained a model, the second step of model-based RL is to integrate planning and learning (Fig 1.1, arrows a-f), which we discuss in Sec. 5. Interestingly, some model-based RL approaches do not explicitly define one or both of these steps (model learning and integration of planning and learning), but rather wrap them into a larger (end-to-end) optimization. We call these methods *implicit model-based RL*, which we cover in Sec. 6. Finally, we conclude the main part of this survey with a discussion of the potential benefits of these approaches, and of model-based RL in general (Sec. 7).

While the main focus of this survey is on the practical/empirical aspects of model-based RL, we also shortly highlight the main theoretical results on the convergence properties of model-based RL algorithms (Sec. 8). Additionally, note that model-based RL is a fundamental approach to sequential decision making, and many other sub-disciplines 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-based way, where 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. Finally, the survey finishes with Related Work (Sec. 9), Discussion (Sec. 10), and Summary (Sec. 11) sections.

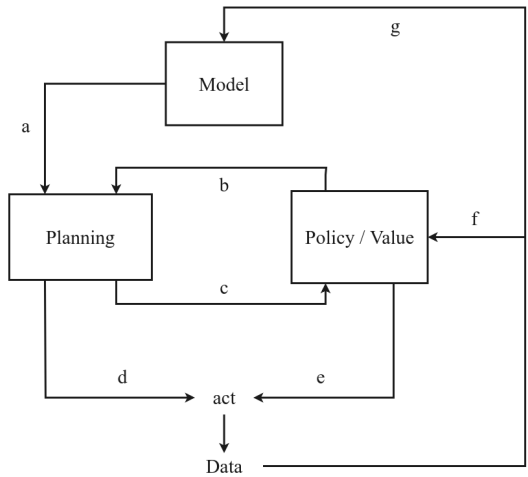

Figure 1.1: Overview of possible algorithmic connections between planning and learning. Learning can take place at two locations: in learning a dynamics model (arrow g), and/or in learning a policy/value function (arrows c and f). Most algorithms only implement a subset of the possible connections. Explanation of each arrow: 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.

2

Background

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 a trace through the environment is denoted by: $J_t = \sum_{k=0}^{K} \gamma^k \cdot r_{t+k}$, for a trace of length K. For $K = \infty$ we call this the infinite-horizon return.

Define the action-value function $Q^\pi(s, a)$ as the expectation of the

cumulative return given a certain policy π:

$$Q^{\pi}(s,a) \doteq \mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{k=0}^{K} \gamma^{k} r_{t+k} \middle| s_t = s, a_t = a\right] \tag{2.1}$$

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

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)}\Big[\mathcal{R}(s,a,s') + \gamma\, \mathbb{E}_{a' \sim \pi(\cdot|s')}\Big[Q^{\pi}(s',a')\Big]\Big] \tag{2.2}$$

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

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

There is *at least* one optimal policy, denoted by $\pi^{\star}$, which is better or equal than all other policies (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), instead of a reward maximization problem. That formulation is interchangeable with our presentation by negating the reward function.

3

Categories of Model-based Reinforcement Learning

To properly define model-based reinforcement learning, we first need individual definitions of planning and reinforcement learning. There are in principle two ways to distinguish both fields: based on their access to the MDP dynamics, and based on the way they represent the solution. Regarding the first distinction, planning methods tend to have *reversible* access to the MDP dynamics, which allows the agent to repeatedly plan forward from the same state (similar to the way humans plan in their mind). Thereby, reversible access to the MDP dynamics allows the agent to query the MDP at any preferred state-action pair (in any preferred order). We call such reversible access to the MDP dynamics a *model*.

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

In contrast, model-free reinforcement learning approaches typically have *irreversible* access to the MDP dynamics, which means that the agent has to move forward from the resulting next state after executing a particular action (similar to the way we act in the real world). This essentially restricts the order in which we can visit state-action pairs in the MDP.

Table 3.1: Distinction between planning, model-free RL, and model-based RL, based on 1) the presence of reversible access to the MDP dynamics (a model, known or learned) and 2) the presence of a global (learned) solution (e.g., policy or value function).

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

Based on this different type of access to the MDP dynamics, both fields have also focused on a different type of representation of the solution. Planning methods typically use a *local* representation of the solution, which only stores the solution for a subset of all states (for example around a current state, which gets discarded afterwards). In contrast, learning methods cannot repeatedly plan from the same state, and therefore have to store a *global* solution, which stores a value function or policy for the entire state space.

Unfortunately, the two distinctions between planning and learning (reversible versus irreversible MDP access and local versus global solution) do not always agree. For example, AlphaZero (Silver *et al.*, 2018) combines reversible access to the MDP dynamics (which would make it planning) with a global policy and value function (which would make it learning). Since many researchers consider AlphaZero a (model-based) reinforcement learning algorithm, we decide to define RL based on the presence of a global (learned) solution. This leads to the following definitions of (MDP) planning and learning (Table 3.1):

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

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

Starting from their different assumptions, both research fields have developed their own methodology. Along the way they started to meet, in

a field that became known as *model-based reinforcement learning* (Sutton, 1990). The principles beneath model-based RL were also discovered in the planning community, in the form of Learning Real-Time A* (Korf, 1990), while the underlying principles already date back to the Checkers programme by Samuel (1967). The key idea of model-based reinforcement learning is to combine a model and a global solution in one algorithm (Table 3.1):

Model-based reinforcement learning is a class of MDP algorithms that 1) use a model, and 2) store a global solution.

Regarding the combination of planning and learning, 'learning' is itself actually an overloaded term, since it can happen at two locations in the algorithm: 1) to learn a dynamics model, and 2) to learn a global solution (e.g., a policy or value function). Since learning can happen at two locations in algorithms that combine planning and learning, we actually end up with three subcategories of planning-learning integration (Table 3.2):

- *Model-based RL with a learned model*, where we both learn a model and learn a global solution. An example is Dyna (Sutton, 1991).
- *Model-based RL with a known model*, where we plan over a known model, and only use learning for the global solution. An example is AlphaZero (Silver *et al.*, 2018), while also Dynamic Programming (Bellman, 1966) technically belongs to this group.
- *Planning over a learned model*, where we do learn a model, but subsequently locally plan over it, without learning a global solution. An example is Embed2Control (Watter *et al.*, 2015).

Note that 'planning over a learned model' is not considered model-based RL, since it does not learn a global solution to the problem (see Table 3.1). However, it is a form of planning-learning integration, and we therefore still include this topic in the survey. It is important to

Table 3.2: Categories of planning-learning integration (as covered in this survey). In MDP algorithms that use planning, learning may happen at two locations: i) to learn a model, and 2) to learn the global solution (e.g., a policy or value function). These leads to three possible combinations of planning and learning, as shown in the table.

	Learned model	**Learned solution**
Model-based RL with a known model	-	+
Model-based RL with a learned model	+	+
Planning over a learned model	+	-

distinguish between the subcategories of planning-learning integration, because they also need to cope with different challenges. For example, approaches with a learned dynamics model typically need to account for model uncertainty, while approaches with a known/given dynamics model can ignore this issue, and put stronger emphasis on asymptotic performance. In the next section (Sec. 4) we will first discuss the various approaches to model learning (first column of Table 3.2), while the subsequent Sec. 5 will cover the ways to integrate planning with learning of a global value or policy function (second column of Table 3.2).

4

Dynamics Model Learning

The first step of model-based RL (with a learned model) 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. 4.1). Afterwards, we extensively cover the various challenges of model learning, and their possible solutions. These challenges are stochasticity (Sec. 4.2), uncertainty due to limited data (Sec. 4.3), partial observability (Sec. 4.4), non-stationarity (Sec. 4.5), multi-step prediction (4.6), state abstraction (Sec. 4.7) and temporal abstraction (Sec. 4.8). 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.

4.1 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 We will here focus on *dynamics models*, which attempt to learn the transition probabilities between states. Technically, the reward function is also part of the model, but it is usually easier to learn (since we only need to predict an additional scalar in our learned dynamics function). We therefore focus on the dynamics model here. Given a batch of one-step transition data $\{s_t, a_t, r_t, s_{t+1}\}$, there are three main types of dynamics functions 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 (Figure 4.1, arrow 1). It is by far the most common type of model, and can be used for lookahead planning. Since model-based RL has mostly focused on forward models, the remainder of this section will primarily focus on this type of model.
- *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. 4.7).

Estimation method We also 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

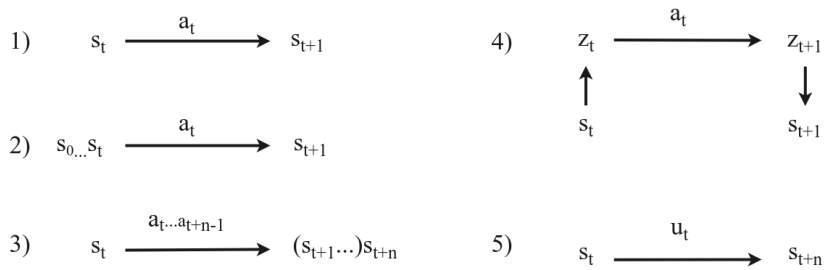

Figure 4.1: Overview of different types of mappings in model learning. **1)** Standard Markovian transition model $s_t, a_t \rightarrow s_{t+1}$. **2)** Partial observability (Section 4.4). We model $s_0...s_t, a_t \rightarrow s_{t+1}$, leveraging the state history to make an accurate prediction. **3)** Multi-step prediction (Section 4.6), 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 4.7), where we compress the state into a compact representation z_t and model the transition in this latent space. **5)** Temporal/action abstraction (Section 4.8), better known as hierarchical reinforcement learning, where we learn an abstract action u_t that brings us to s_{t+n}. Temporal 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.

is independent of the size of the observed dataset. There are two main subgroups:

– *Exact*: An important 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 therefore 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 $\mathcal{S}$ (the curse of dimensionality).

- *Approximate*: We may use function approximation methods, which lower the required number of parameters and allow for generalization. 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 (Sutton *et al.*, 2008; Parr *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 (Werbos, 1989; Narendra and Parthasarathy, 1990; Wahlström *et al.*, 2015; Oh *et al.*, 2015). Especially (deep) neural networks have become popular in the last decade, for function approximation in general (Goodfellow *et al.*, 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 transition models, and the line between model-based RL and replay buffer methods is indeed thin (Vanseijen and Sutton, 2015; Hasselt *et al.*, 2019).

  - *Approximate*: We may also apply non-parametric methods when we want to be able to generalize information to similar states. For example, Gaussian processes (Wang *et al.*, 2006; Deisenroth and Rasmussen, 2011) have been a popular non-parametric approach. Gaussian processes can also provide good uncertainty estimates, which we will further discuss in Sec. 4.3.

  The computational complexity of non-parametric methods depends on the size of the dataset, which makes them in general 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 (both parametric and non-parametric), i.e., all methods that can 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.
- *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 researchers frequently fit local linear approximations of the dynamics around some current state (Atkeson *et al.*, 1997; Bagnell and Schneider, 2001; Levine and Abbeel, 2014). A local model restricts the input domain in which the model is 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 therefore cannot continue to learn

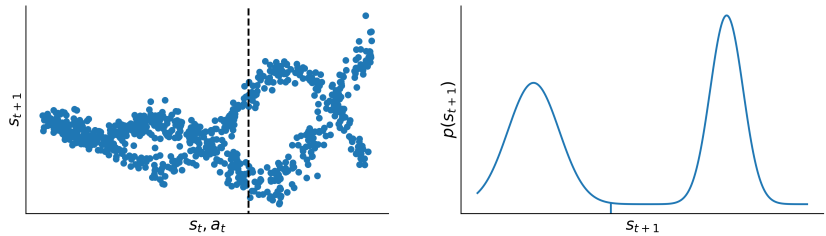

Figure 4.2: Illustration of stochastic transition dynamics. **Left**: 500 samples from an example transition function $\mathcal{T}(s'|s,a)$. The vertical dashed line indicates the cross-section distribution on the right. **Right**: distribution of s_{t+1} for a particular s, a. We observe a multimodal distribution. The conditional mean of this distribution, which would be predicted by mean squared error (MSE) training, is shown as a vertical line.

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 5 and 7.

This concludes our discussion of the three basic considerations (type of model, type of estimation method, region of validity) of model learning. In practice, most model learning methods actually focus on one particular combination: a forward model, with parametric function approximation, and global coverage. The remainder of this section will discuss several more more advanced challenges of model learning, in which this particular setting will also get the most attention.

4.2 Stochasticity

In a stochastic MDP the transition function specifies a distribution over the possible next states, instead of returning a single next state (Figure 4.2, 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 *et al.*, 2017b). This problem is illustrated in Figure 4.2, 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, most successful attempts are based on neural network approximation (deep generative models). One approach is to use variational inference (VI) to estimate dynamics models (Depeweg *et al.*, 2016; Moerland *et al.*, 2017b; Babaeizadeh *et al.*, 2017; Buesing *et al.*, 2018). Competing approaches 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 there is no clear consensus yet which deep generative modeling approach works best.

4.3 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 epistemic uncertainty can be reduced by observing more data, while stochasticity can never be reduced. We will here focus on methods to estimate (epistemic) uncertainty, which is an important topic in model-based RL, since we need to assess whether our plan is actually reliable. Note that uncertainty is even relevant in the absence of stochasticity, as illustrated in Figure 4.3.

We therefore want to estimate the uncertainty around our pre-

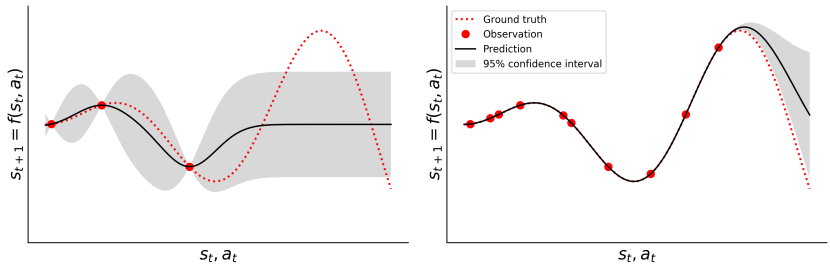

Figure 4.3: Illustration of uncertainty due to limited data. Red dotted line depicts an example ground truth transition function. **Left**: Gaussian Process fit after 3 observations. The predictions are clearly off in the right part of the figure, due to wrong extrapolation. The shaded area shows the 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.

dictions. 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 *et al.* (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 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 *et al.*, 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. We will discuss how to plan over an uncertain model in Sec. 5.

4.4 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 (Figure 4.1, arrow 2). For example, a first-person view agent can not see what is behind itself right now, but it can remember what it saw behind itself a few observations ago, which mitigates the partial observability.

So how do we incorporate information from previous observations? We largely identify four approaches: i) windowing, ii) belief states, iii) recurrency and iv) external memory (Figure 4.4).

- *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 a tabular solution to partial observability. Although this approach can be effective in small problems, they suffer from high memory requirements in bigger problems, and cannot profit from generalization.

- *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 is similar to the sequence modeling approach of *state-space models* (Bishop, 2006), such as hidden Markov models (HMM). This approach represents the dynamics model as a probabilistic graph, in which the parameters are for example estimated through expectation-maximization (EM) (Ghahramani and Hinton, 1996). There is also specific literature on planning for such belief state models, known as POMDP planners (Spaan and Spaan, 2004; Kurniawati *et al.*, 2008; Silver and Veness, 2010). The principles of belief state models have also been combined with neural networks (Krishnan

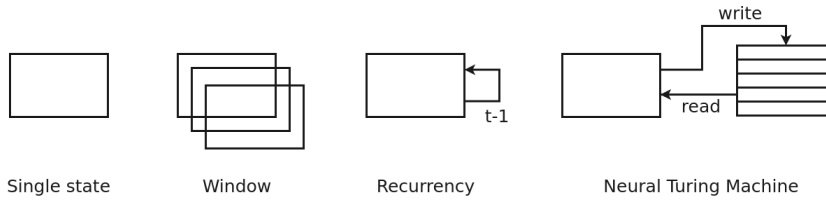

Figure 4.4: Example approaches to partial observability. The window approach concatenates the most recent n frames and treats this as a new state. The recurrent approach learns a recurrent mapping between timesteps to propagate information. The Neural Turing Machine uses an external memory to explicitly write away information and read it back when relevant, which is especially applicable to long-range dependencies.

et al., 2015; Karl *et al.*, 2016), which makes them applicable to high-dimensional problems as well.

- *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 *et al.* (1994). A variety of papers have studied RNNs in high-dimensional settings in recent years (Chiappa *et al.*, 2017; Ha and Schmidhuber, 2018; Gemici *et al.*, 2017). 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. 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 *et al.* (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 *et al.*, 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 focus on some other research question, still incorporate methodology to battle the partial observability in the domain. Note that the above partial observability methodology is equally applicable to a learned policy or value function.

4.5 Non-stationarity

Non-stationarity in an 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 (see Figure 4.5 for an illustration of 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 agent tries to detect a regime switch to subsequently switch between models as well. Da Silva *et al.* (2006) detect a switch based on the prediction errors in transition and reward models, while Nagabandi *et al.* (2018b) makes a soft assignment based on a Dirichlet process. Jaulmes *et al.* (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 higher learning rate may also make training unstable.

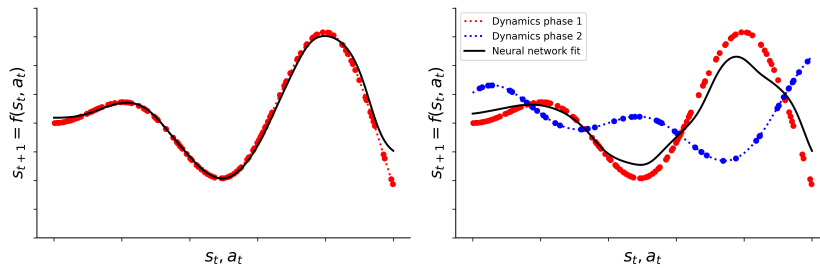

Figure 4.5: Illustration of non-stationarity. **Left**: First 150 data points sampled from initial dynamics. Black line shows the prediction of a neural network with 2 hidden layers of 50 units and tanh activations trained for 150 epochs. **Right**: Due to non-stationarity the dynamics changed to the 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.

4.6 Multi-step Prediction

The models we discussed so far made one-step predictions of the next state. However, we eventually intend to use these in models in multi-step planning procedures. Of course, we can make multi-step predictions with one-step 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 (Talvitie, 2014; Venkatraman *et al.*, 2015; Talvitie, 2017; Machado *et al.*, 2018).

We therefore require models that are robust at long range predictions (Figure 4.1, arrow 3). There are largely two approaches to this challenge: i) different loss functions and ii) separate dynamics functions for 1,2..n-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 *et al.*, 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.

4.7 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 is an important research field in machine learning itself, and many advances in state abstraction for model estimation actually build on results in the broader representation learning community.

Early application of representation learning in RL include (soft) state aggregation (Singh *et al.*, 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 (Oh *et al.*, 2015; Watter *et al.*, 2015; Chiappa *et al.*, 2017).

In the neural network-based approach, the dynamics model is typically factorized into 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 4.1, arrow 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 the more abstract 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 it allows for faster planning. 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 *et al.*, 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 (Watter *et al.*, 2015; Van Hoof *et al.*, 2016; Fraccaro *et al.*, 2017) linearize their learned model on the latent level, and subsequently apply iLQR to solve for a policy (Watter *et al.*, 2015; Zhang *et al.*, 2019; Van Hoof *et al.*, 2016). 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. 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 *et al.*, 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; Xu *et al.*, 2019; Watters *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 *et al.*, 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 *et al.*, 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), Thomas *et al.* (2018), and Sawada (2018).

There is another important research line which 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 *et al.* (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 *et al.*, 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 will cover this idea in Sec. 6 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. To summarize, 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. All these topics have their individual benefit, and future work may actually focus on ways to combine these approaches.

4.8 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 4.1, arrow 5 and Figure 4.6). Indeed, temporal abstraction can theoretically 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 *et al.*, 1999). Options

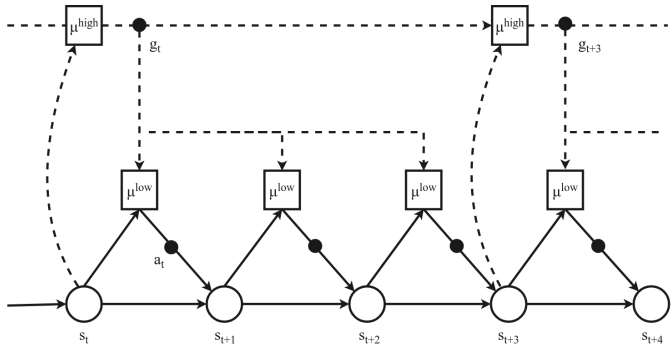

Figure 4.6: Conceptual illustration of a two-level hierarchy, partially based on Nachum *et al.* (2018). Standard low-level interaction is shown with solid lines, temporal abstraction is shown with dashed lines. The high-level controller picks a high-level action (goal) g_t according to π^{high}. After 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.

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 $Q(s, a, g)$, 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 getting closer to g in Euclidean distance (Nachum *et al.*, 2018). Afterwards, we can plan by chaining multiple subgoals (Eysenbach *et al.*, 2019a; Zhang *et al.*, 2021).

Options and goal-conditioned value functions are conceptually different. 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 can some sense interpolate from one-step models (plan a next subgoal which is only one step away) to model-free RL (directly give the final goal to the GCVF, without any planning), 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 often: 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 end-points for the subroutines. A bottleneck is a state that connects two densely interconnected subgraphs in the MDP graph (Menache *et al.*, 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 *et al.* (2005) run a graph partitioning algorithm 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 problems, but does not scale well to 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 (Mannor *et al.*, 2004; Lakshminarayanan *et al.*, 2016; Machado *et al.*, 2017). Instead of the raw state space, we may also cluster in a compressed representation of it (Ghosh *et al.*, 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. Gregor *et al.* (2016), Eysenbach *et al.* (2019b), and Achiam *et al.* (2018) associate the distribution of observed end-states with a noise 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 *et al.* (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 *et al.* (2017), Hausman *et al.* (2018), and Fox *et al.* (2016).

- *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 *et al.*, 2017; Riemer *et al.*, 2018) and Feudal Networks (Vezhnevets *et al.*, 2017), with more examples in Frans *et al.* (2018), Levy *et al.* (2019), Heess *et al.* (2016), 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 to prevent degeneration, 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 (Tessler *et al.*, 2017; Heess *et al.*, 2016). 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 (Konidaris *et al.*, 2012; Fox *et al.*, 2017; Hamidi *et al.*, 2015), which is of course also a form of prior knowledge.

This concludes our discussion of temporal abstraction, and thereby also our discussion of model learning as a whole. In summary, we have seen a variety of important challenges in model learning, which several research papers have addressed. In all directions important progress has been made, but most papers tend to focus on a specific problem in isolation. Since complex real-world tasks likely require many of the discussed aspects, the combination of these methods in more complex tasks seems an important future research direction.

5

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 (Tolman, 1948; Doll *et al.*, 2012) and behavioural psychology (Craik, 1943; Wolpert *et al.*, 1995; Doll *et al.*, 2012). 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 taxonomy that disentangles four central questions of the integration of planning and learning. The four main questions we need to answer are:

1. At which state do we start planning? (Sec. 5.1)
2. How much planning budget do we allocate for planning and real data collection? (Sec. 5.2)
3. How to plan? (Sec. 5.3)
4. How to integrate planning in the learning and acting loop? (Sec. 5.4)

The first three questions mostly focus on the planning method itself (Fig. 1.1, arrow a), while the last question covers the way planning

is integrated in the learning and acting loop (Fig. 1.1, arrows b-g). Note that each of the above questions actually have several important subconsiderations, which leads to the overall taxonomy summarized in Table 5.1. The next sections will discuss each of these subconsiderations.

5.1 At which state do we start planning?

The natural first question of planning is: at which state do we start? There are several options:

- *Uniform*: A straightforward approach is to uniformly select states throughout the 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 has also been used in replay database approaches (Schaul *et al.*, 2016).

- *Current*: Finally, a common approach is to only spend planning effort at the current state of the real environment. This puts 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 *et al.*, 2017b), sometimes (because of the problem size) 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 *et al.*, 2020b as well).

5.2 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) when we start planning for a particular state, what planning budget do 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 irreversible 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 *et al.*, 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 next 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.*, 2017b) 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 full planning algorithm itself (which in AlphaGo Zero for example contains expensive neural network evaluations).

Some approaches, especially the ones that target high data efficiency (see Sec. 7.1) in the real environment, allow for a 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. 7.1.

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 make a more adaptive trade-off (Keramati *et al.*, 2011), 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). See Hamrick (2019) for a more detailed discussion, which also incorporates more literature from human psychology. We will also return to this topic in Sec. 7.3.

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 *et al.* (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.

5.3 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 *et al.* (2020b) 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 *et al.*, 2008), Monte Carlo Tree Search (Silver *et al.*, 2017b; Anthony *et al.*, 2017; Jiang *et al.*, 2018; Moerland *et al.*, 2018b), minimax-search (Samuel, 1967;

Baxter *et al.*, 1999), 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 simulator is by definition not differentiable, our learned model of these dynamics (for example a neural network) usually is differentiable. Therefore, model-based RL can 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 (Degrave *et al.*, 2019; Avila Belbute-Peres *et al.*, 2018).

  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 *et al.*, 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. 4.1):

- *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 this approach here.

- *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 (Moore and Atkeson, 1993; Dearden *et al.*, 1999; Wiering and Schmidhuber, 1998), which are trivial to invert. Example that use function approximation include linear approximation (Sutton *et al.*, 2012), nearest-neighbour models (Jong and Stone, 2007), and neural network approximation (Agostinelli *et al.*, 2019; Edwards *et al.*, 2018; Corneil *et al.*, 2018). The benefit of prioritized sweeping can be large, due to its ability to quickly propagate relevant new information, but in combination with function approximation it can also be unstable.

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 5.1.

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 (Kocsis and Szepesvári, 2006; Coulom, 2006; 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 also used in AlphaGo Zero (Silver *et al.*, 2017b).

- *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 choose 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 *et al.* (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, an 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 *et al.*, 2016).

This was a shallow treatment of the crucial breadth versus depth balancing in planning, which has a close relation to exploration methods as well. 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 (i.e., backtrack over multiple actions in a state). Nevertheless, many model-based RL methods still choose to stick to a breadth of one in their model samples, likely because this gives seamless integration with model-free updates. We further discuss this topic in Sec. 7.1.

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 constrain the

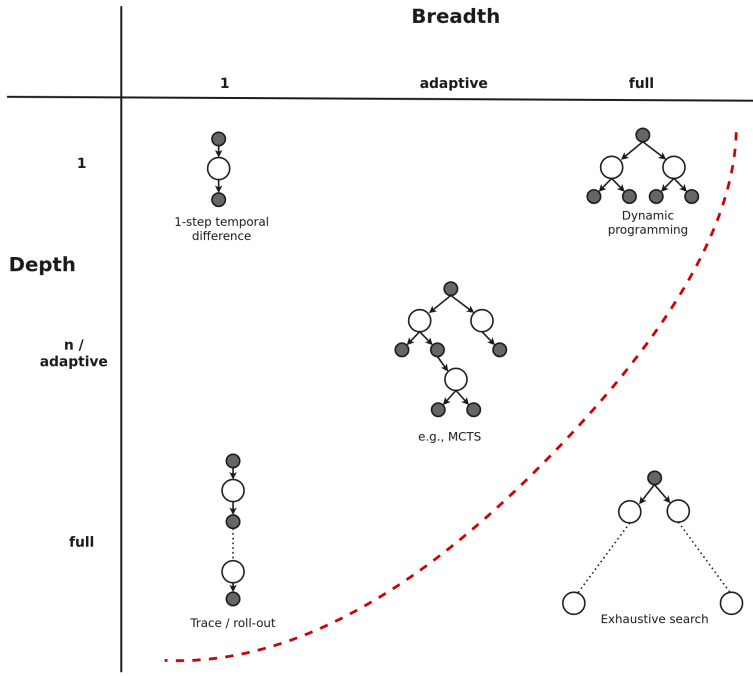

Figure 5.1: Breadth and depth of a single planning iteration. For every planning iteration, we need to decide on the breadth and depth of the lookahead. In practice, planning iterations usually reside somewhere left of the red dashed line, since we cannot afford a full breadth, full depth (exhaustive) 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.

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. 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:

- *Analytic*: This propagation method fits a parametric distribution to the uncertainty at every timestep, and tries to analytically propagate the distribution over passes through the model. A well-known example is PILCO (Deisenroth and Rasmussen, 2011), which derives closed form analytic expressions to propagate uncertainty through a Gaussian Process model. However, analytic propagation is usually 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 *et al.*, 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 *et al.*, 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, which may lead to more stable learning.

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, such as managing exploration (i.e., balancing breadth and depth in the planning tree). However, these challenges are not a specific aspect of planning-learning integration (but rather of RL and planning in general), and are therefore not further discussed in this survey (although we do discuss model-based exploration in Sec. 7.2).

5.4 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. We now get back to Figure 1.1, which presented a conceptual overview of the overall training loop in planning-learning integration. 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. 1.1, arrow b), ii) using planning output to update learned value or policy functions (Fig. 1.1, arrow c), and iii) using planning output to select actions in the real world (Fig. 1.1, arrow d).

Planning input from learned functions The learned value or policy functions may store much information about the current environment, which may direct 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 algorithms use bootstrapping in their planning approach, for example Baxter *et al.* (1999), Silver *et al.* (2017b), Jiang *et al.* (2018), and Moerland *et al.* (2018b), while is is technically also part of Dynamic Programming (Bellman, 1966). Note that bootstrapping is also a 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 (Silver *et al.*, 2008; Hamrick *et al.*, 2020), 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.*, 2017b) 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 another 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. 5.1, a form of prioritization). In short, there are various ways in which we may incorporate prior knowledge from a policy into planning, although it is not clear yet which approach works best.

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 only sample a single action (i.e., use a breadth of one), like Dyna (Sutton, 1990), we may use a classic Q-learning target (one-step, off-policy). For planning cycles that do consider multiple actions in a state (that go wide and possibly deep), we can combine on- and off-policy back-ups throughout the

tree in various ways. Willemsen *et al.* (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 (Veness *et al.*, 2009; Moerland *et al.*, 2018b). 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.*, 2017b) 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.* (2018b) cover an extension to a loss between discrete counts and a continuous policy network. As a 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. 5.3) 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 (Ha and Schmidhuber, 2018; Kurutach *et al.*, 2018; Depeweg *et al.*, 2016; Deisenroth and Rasmussen, 2011), using real data only to train the dynamics model.

Note that arrows b and c in Figure 1.1 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 algorithm in this class is AlphaGo Zero (Silver *et al.*, 2017b), which is an instance of *multi-step approximate real-time dynamic programming* (MSA-RTDP) (Efroni *et al.*, 2019a). 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 *et al.* (2019a) 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 is an intuitive result, it does prove 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. 7.3.

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. 1.1, arrow e), model-based RL may also select actions directly from the planning output (Fig. 1.1, arrow d). Some methods only use planning for action selection, not for value/policy updating (Tesauro and Galperin, 1997; Silver *et al.*, 2008), for example because planning updates can have uncertainty. However, many methods actually combine both uses (Silver *et al.*, 2017b; Silver *et al.*, 2018; Anthony *et al.*, 2017; Moerland *et al.*, 2018b).

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 3.2). 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 the 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 (Nagabandi *et al.*, 2018c; Chua *et al.*, 2018), Dijkstra's algorithm (Kurutach *et al.*, 2018), or repeated application of an inverse model (Agrawal *et al.*, 2016). MPC is robust to (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 a planning cycle ends, but can introduce additional exploration noise (like Dirichilet noise in Silver *et al.* (2017b)). We can also explicitly incorporate exploration criteria in the planning process, which we may call 'plan to explore' (Sekar *et al.*, 2020; Lowrey *et al.*, 2018). For example, Dearden *et al.* (1998) explore based on the value of perfect information' (VPI), which estimates from the model which action has the highest potential to change the greedy policy. Indeed, planning may identify action sequences that perform 'deep exploration' towards new reward regions, which one-step exploration methods would fail to identify due to jittering behaviour (Osband *et al.*, 2016).

This concludes our discussion of the main considerations in planning-learning integration. Table 5.1 summarizes the framework, showing the potential decisions on each dimension.

5.5 Conceptual comparison of approaches

This section 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 5.2 summarizes the different types of connectivity that may be present in planning-learning integration. The figure is based on the scheme of Figure 1.1, as used throughout this section, and the classification of model-based RL methods described in Table 3.2.

We see how well-known model-based RL algorithms like Dyna (Sutton, 1991) and AlphaGo Zero (Silver *et al.*, 2017b) use different connectivity. For example, Dyna learns a model, which AlphaGo Zero assumes

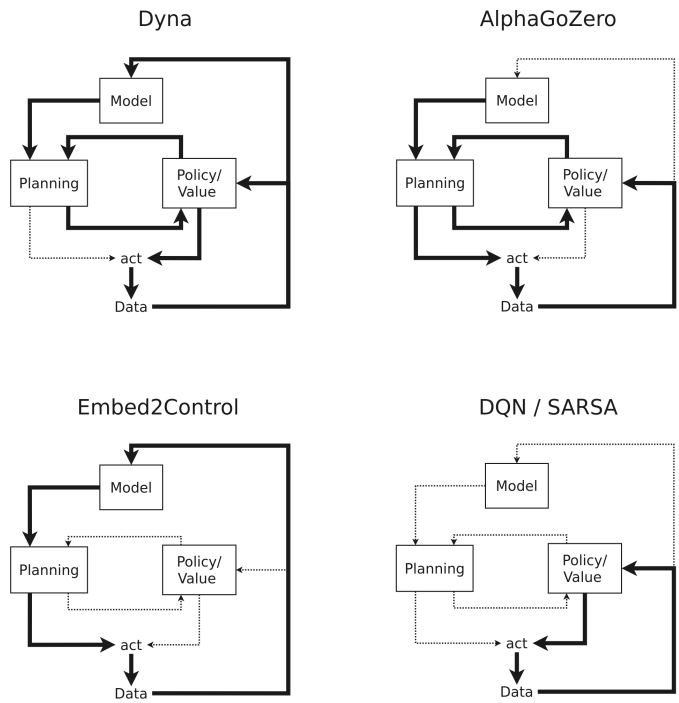

Figure 5.2: Comparison of planning and learning algorithms, based on the general visualization of learning/planning integration from Figure 1.1. Thick lines are used by an algorithm. Dyna (Sutton, 1991) (top-left) is an example of model-based RL with a learned model. AlphaGo Zero (Silver *et al.*, 2017b) (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

.

to be known, and AlphaGo Zero selects 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 5.2 compares several well-known model-based RL algorithms on the dimensions of our framework for planning-learning integration (Table 5.1). We see how different integration approaches make different choices on each of the dimensions. It is hard to judge whether some integration approaches are better than others, since they are generally evaluated on different types of tasks (more on benchmarking in Sec. 10. The preferred choices of course also depend on the specific problem and available resources. For example, a problem like Go requires are relatively high planning budget per timestep (Silver *et al.*, 2017b), while for smaller problems a lower planning budget per timestep may suffice. Gradient-based planning can be useful, but is mostly applicable to continuous control tasks, due to the relatively smooth dynamics. For many considerations, there are both pros and cons. Usually, the eventual decisions depends on hyperparameter optimization and the type of benefit (of model-based RL) we aim for, which we will discuss in Sec 7.

Table 5.1: Overview of taxonomy of planning-learning integration. These considerations are discussed throughout Sec. 5. Table 5.2 summarizes several model-based RL algorithms on these dimensions.

Dimension	**Consideration**	**Choices**
1. Start state (5.1)	- Start state	Uniform ↔ visited ↔ prioritized ↔ current
2. Budget (5.2)	- Number of real steps before planning	1 ↔ n, episode, etc.
	- Effort per planning cycle	1 ↔ n ↔ convergence
3. Planning approach (5.3)	- 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 (5.4)	- 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

Table 5.2: Systematic comparison of different model-based RL algorithms on the dimensions of planning-learning integration (Sec. 5). 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 3.2). 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 one 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.*, 2017b)	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	-	✓

6

Implicit Model-based Reinforcement Learning

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 specified each part of the algorithm. 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 model-based RL algorithms, 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 single optimization objective, since we want it 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

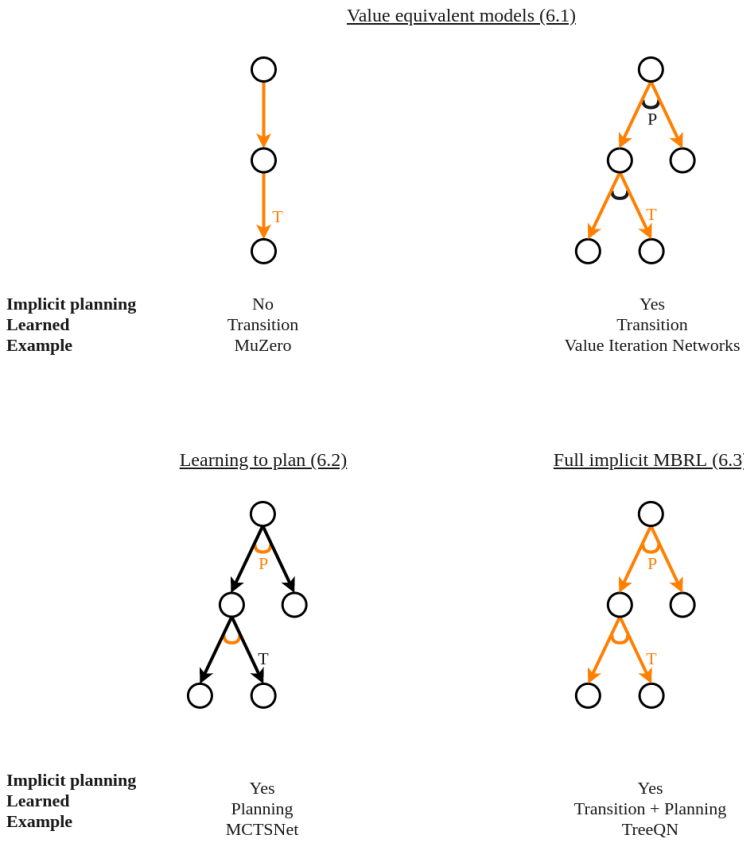

Figure 6.1: Categories of implicit model-based RL (Sec. 6). Graphs schematically illustrate a differentiable computation graph, with transitions (T) denoted as lines, and planning/policy improvement (P) denoted as an arc between alternative actions (which are not explicitly drawn). The depiction of planning (policy improvement in the graph) is of course conceptual, and may in practice involve several networks (e.g., an action selection network and a back-up network). For each graph, black lines are known (in differentiable form), while orange lines are learned. Top-left: Value equivalent model in the policy evaluation setting, of which MuZero (Schrittwieser *et al.*, 2019) is an example. Top-right: Value equivalent model with implicit planning, of which Value Iteration Networks (Tamar *et al.*, 2016) are an example. Bottom-left: Learning to plan, of which MCTSNet (Guez *et al.*, 2018) is an example. Bottom-right: Full implicit model-based RL, of which TreeQN (Farquhar *et al.*, 2018) is an example.

optimization. In those cases, we embed (parts of) the model-based RL process within a differentiable 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 an eventual value prediction or action recommendation. One would be tempted to therefore call the field 'end-to-end model-based RL', but note 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 optimize a transition model and 2) to optimize for the actually planning procedure (i.e., some form of policy optimization). This leads to the possible settings shown in Fig. 6.1. The figure schematically shows the different types of computational graphs we may construct, and which elements of this graph are optimized for (shown in orange). We will discuss these possible settings shown in the later part of this section.

The differentiable computational graphs of course need to be optimized against some outer objective, for which there are two options. First, we may train the graph for its ability to predict the correct (optimal) value (an *RL loss*). This value is frequently obtained from a standard model-free RL target constructed from observed traces. The second option is to train the graph against its ability to output the correct (optimal) action or policy (an *imitation loss*). Such knowledge may either be available from expert demonstrations, or can be obtained from running a separate model-free RL agent. The underling intuition is to first optimize the model and/or planning procedure against correct value or action targets in a task, which may afterwards lead to superior performance in the same task, or generalization to other tasks.

The remainder of this section will discuss the three possible forms of computational graphs shows in Fig. 6.1: value equivalent models (Sec. 6.1), where we only optimize the transition dynamics in the graph, learning to plan (Sec. 6.2), where we only optimize the planning operations, and full implicit model-based RL (Sec. 6.3), where we jointly optimize the transition model and planning operations. A structured overview of the papers we discuss is provided in Table 6.1.

6.1 Value equivalent models

Standard model learning approaches, as discussed in Section 4, 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 simpler. Grimm *et al.* (2020) theoretically study this idea, which they name *value equivalent models*. Value equivalent models are unrolled inside the computation graph to predict a future value (or action), instead of a future state. As such, these models are enforced to emphasize value-relevant characteristics of the environment.

An example of a successful value-equivalent approach is MuZero (Schrittwieser *et al.*, 2019). To learn a transition function, MuZero internally unrolls a model to predict a multi-step, action-conditional value (Fig. 6.1, top-left). The graph is then optimized for its ability to predict a model-free value estimate at each step. The obtained value-equivalent model is subsequently use in an MCTS procedure, which achieved state-of-the-art performance in the Chess, Go and Shogi. Other successful examples of this approach are Value Prediction Networks (VPN) (Oh *et al.*, 2017) and the Predictron (Silver *et al.*, 2017a).

A crucial aspect of the above methods is that they unroll a single trace, and therefore train in the policy evaluation setting. In contrast, we may also optimize the transition model in an implicit planning graph, which does contain policy improvement (Fig. 6.1, top-right). Two example papers that take this approach are Value Iteration Networks (VIN) (Tamar *et al.*, 2016) and Universal Planning Networks (UPN) (Srinivas *et al.*, 2018). Both papers specify a known, differentiable planning procedure in the graph (VIN embeds value iteration, UPN embeds value-gradient planning). The entire planning procedure consist of multiple cycles through the transition model and planner, which is then optimized for its ability to predict a correct optimal action or value. Since VINs and UPNs do incorporate policy improvement in the computational graph, they may learn slightly different aspects of the dynamics than MuZero and VPN (i.e., VIN/UPN will only

emphasize aspects necessary to predict the optimal action/value, while MuZero/VPN will emphasize aspects necessary to make correct multi-step predictions for all observed action sequences).

6.2 Learning to plan

We may also use the implicit planning idea to optimize the planning operations themselves (Fig. 6.1, bottom-left). So far, we encountered two ways in which learning may enter model-based RL: i) to learn a dynamics model (Sec. 4), and ii) to learn a value or policy function (from planning output) (Sec. 5). We now encounter a third level in which learning may enter model-based RL: to learn the actual planning operations (and its integration with a learned value or policy function). End-to-end learning has of course shown success in other machine learning fields, such as computer vision, where it has gradually replaced manually constructed features. A similar trend starts to appear for entire algorithms, which we may better learn through optimization than manually specify. This idea is known as *algorithmic function approximation* (Guez *et al.*, 2019), where our learned approximator makes multiple internal cycles (like a recurrent neural network) to improve the quality of its prediction. Note that this differs from standard RNNs, where the recurrence is typically used to deal with additional inputs or outputs (e.g., in the time dimension), while in algorithmic function approximation the *additional internal cycles improve the quality of the predictions.*

An example of learning to plan are MCTSNets (Guez *et al.*, 2018). This approach specifies elements of the MCTS algorithm, like selection, back-up and final recommendation, as neural networks, and optimizes these against the ability to predict the correct optimal action in the game Sokoban. While MCTSNets assume a known dynamics model, Imagination-augmented agents (I2A) (Racanière *et al.*, 2017) first separately learn a differentiable dynamics model (in the standard way), and then learn how to aggregate information from roll-outs in this model through the implicit-model-based RL approach (which shows we can actually combine conventional and implicit model-based RL approaches). While both these algorithms (MCTSNets and I2A) still include some manual design in their planning procedure (like the order of node ex-

Table 6.1: Comparison of implicit model-based RL approaches. **Rows**: algorithm colour coding, yellow = explicit model-based RL (for comparison), green = value equivalent models (Sec. 6.1), red = learning to plan (Sec. 6.2), blue = combination of value equivalent models and learning to plan (Sec. 6.3). **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 *et al.*, 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)

pansion), Imagination-based planner (IBP) (Pascanu *et al.*, 2017) also includes a differentiable manager network that in each iteration decides whether we continue planning and from which state we will plan. This gives the algorithm almost full freedom in the algorithmic planning space, and the authors therefore also include a cost for simulation, which ensures that the agent will not continue to plan forever. Results show that the agent indeed learns both how to plan and for how long to plan.

6.3 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 (Fig. 6.1, bottom-right). 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 and 5 get wrapped into a single optimization.

An example approach in this category is TreeQN (Farquhar *et al.*, 2018). From the outside, TreeQN looks like a standard value network, but internally it is 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, which outputs the value estimate for the input state. We then optimize both the dynamics model and the planning procedure (back-up network) against a standard RL loss. While TreeQN still has some internal structure, full algorithmic freedom is provided by the Deep Repeated ConvLSTM (DRC) (Guez *et al.*, 2019). DRC is a high-capacity recurrent neural network without any planning or MDP specific internal structure. It is therefore entirely up to the DRC to approximate both an appropriate (value-equivalent) model and an appropriate planning procedure, which the authors call *model-free planning*. Indeed, DRC does show characteristics of planning after training, like an increase in test performance with increasing computational budget.

An overview of the discussed papers in this section is provided in Table 6.1. The strength of the implicit model-based RL approach is tied to the strength of optimization in general, which, as other fields of machine learning have shown, may outperform manual design. Moreover, value equivalent models may be beneficial in tasks where the dynamics are complicated, but the dynamics relevant for value estimation are easier to learn. On the other hand, implicit model-based RL has its challenges as well. For value-equivalent transition models, all learned predictions focus on the value and reward information, which is derived from a scalar signal. These methods will therefore likely not capture all relevant aspects of the environment, which may be problematic for transfer. A similar problem occurs for learning to plan, where we risk that our planner will learn to exploit task-specific characteristics, which does not generalize to other tasks. The true solution to these problems is of course to train on a wide variety of tasks, which is computationally demanding, while implicit model-based RL is already computationally demanding in itself (the computational graphs grow 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*).

7

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 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 7.1). First, we will discuss enhanced data efficiency (Sec. 7.1), which uses planning (increased model sample complexity) to reduce the real-world sample complexity. Second, we discuss exploration methods that use model characteristics (Sec. 7.2). 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. 7.3). A fourth potential benefit is transfer (Sec. 7.4), which attempts to reduce the sample complexity on a sequence of tasks by exploiting commonalities. Finally, we also shortly touch upon safety (Sec. 7.5), and explainability (Sec. 7.6).

7.1 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. 5.2). 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 *et al.*, 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 *et al.*, 2012). All data efficiency approaches

crucially need to deal with model uncertainty, which may be estimated with a Bayesian approach (Guez *et al.*, 2012; Asmuth *et al.*, 2009; Castro and Precup, 2007). 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. 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 *et al.*, 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.* (2018c) 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

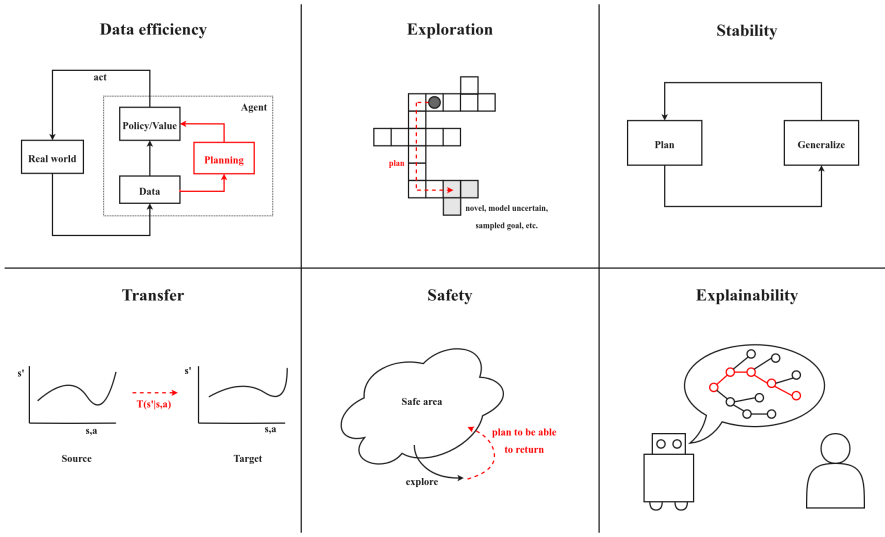

Figure 7.1: Benefits of model-based reinforcement learning, as discussed in Section 7.

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. 7.3) 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 *et al.*, 2013), and is a major motivation for the model-based approach.

7.2 Exploration

The trade-off between exploration and exploitation is a crucial topic within reinforcement learning, in which model-based approaches can play an important role. There are two important considerations that determine whether a particular exploration approach is model-based, as visualized in Table 7.1. First, we need to distinguish *one-phase versus two-phase exploration* (Table 7.1, columns). Model-free RL methods and pure planning methods use 'one-phase exploration': they use the same exploration principle in the entire algorithm, i.e., either within a trace (model-free RL) or within a planning tree. 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/greedy) 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.*, 2017b). However, with a known model (in which case planning and real steps both happen in the same reversible model), the second phase rather seems a pruning technique, to ensure that we stop planning at some point and advance to a next state.

The second important distinction is between *value-based and state-based exploration (intrinsic motivation)* (Table 7.1, rows). Value-based methods based 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, where the perturbation may for example be random (Plappert *et al.*, 2017; Mnih *et al.*, 2015) or based on uncertainty estimates around these values (Auer, 2002; Osband *et al.*, 2016; Moerland *et al.*, 2017a). 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

Table 7.1: 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 on value function	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

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 *et al.*, 2005).

The two above distinctions together lead to four possible combinations, as visualized in the cells of Table 7.1. 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 7.1). We therefore consider all state-based exploration methods as model-based RL. State-based exploration methods often use model-based characteristics or a density model over state space (which in the tabular setting can directly be derived from a tabular model), and therefore have a close relation to model-based RL, even when it is applied without actual planning (one-phase).

Considerations in exploration To get a better understanding of the main challenges in (model-based) exploration, we will first discuss the most important general challenges in exploration:

- *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 rather want to fix an interesting exploration target in the future, and commit to a sequence of actions to actually get there. This approach is known as *deep* exploration (Osband *et al.*, 2016) (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 (Osband *et al.*, 2016; Moerland *et al.*, 2017a). However, deep exploration is 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 replan (a receding-horizon), 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 potentially 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 *et al.*, 2005) (e.g., for novelty), and simply add these as a bonus to the extrinsic reward. The exploration potential of a state is then propagated inside the global value/policy function, together with information about the extrinsic reward. We will call this *task-conflated propagation*, since exploration information (intrinsic rewards) is merged with information about the true task (extrinsic rewards). A potential downside of this approach is that exploration information modifies the global solution, and, after an intrinsic reward has worn out, it may take time before its effect on the value function has faded out.

  As an alternative, we may also use *task-separated* exploration back-ups. In this case, 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 *et al.* (2019) propose to train separate value functions for the intrinsic and extrinsic rewards. We can also use a goal-conditioned policy or value function, which automatically separates information for each potential goal state. In general, task-separated exploration back-ups come at additional computational (and memory) cost, but they do allow for better separation of exploration information and the true extrinsic task.

- *Parametric versus non-parametric (deep) exploration back-ups*: Similar to the depth of exploration in the forward sense, the depth of the back-up also plays a crucial role for exploration. Imagine our agent just discovered an interesting novel state, which we would like to visit again in a next episode. However, we use a one-step back-up (which only propagates information about this state one step back), and we store this information in a deep policy or value network, to which we make small updates. A a consequence of these choices, information may not propagate far enough (only one step), and/or the change to the global value or policy function may not be large enough to change the behaviour of the agent at the start states. The effect is that the agent has actually found an interesting new region, but due to its type of back-up is not able to directly visit this region again. Ecoffet *et al.* (2019) call this the 'detachment' problem, since the information does not propagate far enough, and the agent therefore detaches from it in its initial states.

  A potential solution to this problem is the use of deeper back-ups, especially in combination with *semi-parametric* or *non-parametric* representation for the exploratory information. In the context of reinforcement learning, non-parametric representations are better known as *episodic memory* (Blundell *et al.*, 2016). In episodic memory, we store the exact action sequence towards a particular state, based on a non-parametric overwrite of information in a table. Note that this approach can also be extended to the semi-parametric setting, where we train a neural network to read and write to this table (Graves *et al.*, 2014; Pritzel *et al.*, 2017).

The benefit of episodic memory is a fast change of information, which can help the agent to directly get back to an interesting region (by simply copying the actions that previously brought it there). Indeed, episodic memory is known to play an important role in human and animal learning as well (Gershman and Daw, 2017). Go-Explore (Ecoffet *et al.*, 2019) implicitly uses this idea, but directly resets the agent to a previously seen state (without actually replaying the action sequence from the start). Note that there is a variety of other research into episodic memory for reinforcement learning (Blundell *et al.*, 2016; Pritzel *et al.*, 2017; Lin *et al.*, 2018; Loynd *et al.*, 2018; Ramani, 2019; Fortunato *et al.*, 2019; Hu *et al.*, 2021).

With our understanding of the above concepts, we are now ready to discuss model-based exploration. We will focus on the intrinsic motivation (IM) literature (i.e., state-based exploration, the bottom row of Table 7.1) (Chentanez *et al.*, 2005). This field is traditionally split up in two sub-fields (Oudeyer, Kaplan, *et al.*, 2008): *knowledge-based* and *competence-based* intrinsic motivation (Fig. 7.2), which differ in the way they define the exploration potential of a certain state.

Knowledge-based intrinsic motivation Knowledge-based intrinsic motivation prioritizes states for exploration when they *provide new information about the MDP*. Most approaches in this category specific a specific intrinsic reward, which is then propagated together with the extrinsic reward (task-conflated exploration back-ups). Writing $r^i(s)$ or $r^i(s, a, s')$ for specific intrinsic reward of a certain state or transition, these methods use a total reward that combines the intrinsic and extrinsic part:

$$r_t(s, a, s') = r^e(s, a, s') + \eta \cdot r^i(s, a, s'), \tag{7.1}$$

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

Most knowledge-based IM literature focuses on different ways to specify r^i. By far the largest category uses the concept of *novelty* (Hester and Stone, 2012a; Bellemare *et al.*, 2016; Sequeira *et al.*, 2014). For

example, the Bayesian Exploration Bonus (BEB) (Kolter and Ng, 2009) uses

$$r^i(s,a,s') \propto 1/(1+n(s,a)), \tag{7.2}$$

where $n(s,a)$ denotes the number of visits to state-action pair (s,a). Novelty ideas were 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*:

$$r^i(s,a,s') = \sqrt{l(s,a)}, \tag{7.3}$$

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* (Stadie *et al.*, 2015; Pathak *et al.*, 2017), *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. 4.5) 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 can be related to emotion theory, which was surveyed for RL agents by Moerland *et al.* (2018a).

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 *et al.* (2019) do learn separate value functions for the intrinsic and extrinsic rewards. Note that novelty is also an important concept in theoretical work on the sample complexity of exploration (Kakade *et al.*, 2003; Brafman and Tennenholtz, 2002), which we further discuss in Sec. 8.

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 agent's competence to achieve something, rather than knowledge about the MDP (e.g., we may have visited a state often, which would make it uninteresting for knowledge-based IM, but if we are still getting better/faster at actually reaching the state, i.e., we still make learning progress, then the state does remain interesting for competence-based IM). In competence-based IM the intention is usually to generate an automatic *curriculum* of tasks, guided by learning progress (Bengio *et al.*, 2009).

A popular formulation of compentence-based IM methods are Intrinsically Motivated Goal Exploration Processes (IMGEP) (Baranes and Oudeyer, 2009), which consist of three steps: 1) learn a goal space, 2) sample a goal, and 3) plan/get towards the goal. Goal space learning was already discussed in Sec. 4.7 and 4.8. The general aim is to learn a representation that captures the salient directions of variation in a task. 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 *et al.*, 2018).

The second step, goal space sampling, is a crucial part of competence-based IM, since we want to select a goal that has high potential for *learning progress* (Oudeyer *et al.*, 2007; Baranes and Oudeyer, 2013). One approach is to track a set of goals, and reselect those goals for which the achieved return has shown positive change recently (Matiisen *et al.*, 2017; Laversanne-Finot *et al.*, 2018). 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, 1991).

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-

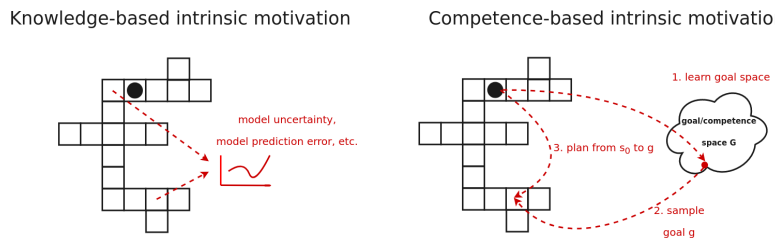

Figure 7.2: Knowledge-based versus competence-based intrinsic motivation. Solid circle identifies the current agent position. **Left**: In knowledge-based intrinsic motivation, every state (the arrows show two examples) in the domain gets associated with an intrinsic reward based on local characteristics, 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.

conditioned value functions (discussed in Sec. 4.8) 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 *et al.*, 2018). Episodic memory methods could also be applied here.

In short, model-based exploration is an active research topic, which has already made important contributions to exploration research. An important next step would be to show that these methods can also outperform model-free RL in large applications. Another important aspect, which we have not discussed in this section, is the potential benefit of hierarchical RL for exploration. We already covered the challenge of learning good hierarchical actions in Sec. 4.8. However, once good abstract actions are available, they will likely be a crucial component of (model-based) exploration as well, due to a reducing of the lookahead and propagation depth.

7.3 Optimality

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 lose asymptotic performance in the long run. However, recent attempts of model-based RL with a known model, like AlphaGo Zero (Silver *et al.*, 2017b) and Guided Policy Search (Levine and Koltun, 2013), manage to outperform model-free attempts on long-run empirical performance. MuZero (Schrittwieser *et al.*, 2019), which uses a (value-equivalent) learned model, further outperforms the results of AlphaGo Zero. This suggests that with a perfect (or good) model, model-based RL may actually lead to better (empirical) asymptotic performance.

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 *et al.* (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.* (2020a) and Wang *et al.* (2019) recently studied the trade-off between planning and learning (already mentioned in Sec. 5.2), finding that optimal performance requires an intermediate planning budget per real step, and not a 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 *et al.* (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'.

7.4 Transfer

In *transfer learning* (Taylor and Stone, 2009; Lazaric, 2012) 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 and Whiteson, 2017; Roijers *et al.*, 2013). 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 *et al.* (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 (Dayan, 1993; Barreto *et al.*, 2017). 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. 4.7, 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 *et al.* (2020).

7.5 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 *et al.* (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.

7.6 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 *et al.*, 2018a), which largely builds on model-based methods. Finally, Shu *et al.* (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.

7.7 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 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.

8

Theory of Model-based Reinforcement Learning

There is also a large body of literature on the theoretical convergence properties of model-based reinforcement learning. Although the primary focus of this survey was on the practical/empirical aspects of model-based RL, we will shortly highlight some main theoretical results. Classic convergence results in dynamic programming are for example available for policy iteration (Puterman, 2014), approximate policy iteration (Kakade and Langford, 2002; Munos, 2003), and real-time dynamic programming (RTDP) (Barto *et al.*, 1995). Efroni *et al.* (2018) show multi-step policy iteration also converges, as does multi-step and approximate RTDP (Efroni *et al.*, 2019a).

Much theoretical work tries to quantify the rate at which algorithms converge, which we can largely split up in sample complexity bounds (PAC bounds) and regret bounds. Sample complexity is typically assessed through the Probably Approximately Correct (PAC) framework. Here, we try to bound the number of timesteps an algorithm may select an actions whose value is not near-optimal (Strehl *et al.*, 2006). An algorithm is PAC if this number is bounded by a function polynomial in the problem characteristics, like the MDP horizon H and the dimensionality of the state ($|\mathcal{S}|$) and action space ($|\mathcal{A}|$). There are a variety

of papers that provide PAC bounds for MDP algorithms (Kakade *et al.*, 2003; Strehl *et al.*, 2006; Dann and Brunskill, 2015; Asmuth *et al.*, 2009; Szita and Szepesvári, 2010).

An alternative approach is to bound the *regret* during the learning process. The regret measures the average total loss of reward of the learned policy compared to the optimal policy. The regret at timestep T is therefore defined as

$$\text{Regret}(T) = \mathbb{E}_{\pi^\star}[\sum_{t=1}^{T} r_t] - \mathbb{E}_{\pi}[\sum_{t=1}^{T} r_t], \tag{8.1}$$

where $\pi^\star$ denotes the optimal policy, and π denotes the (changing) policy of our learning algorithm. While PAC bounds the *total number* of sub-optimal actions a learned policy will take, regret bounds limit the *total size* of the mistakes. The *lower* bound of the above regret is known to be $\Omega(\sqrt{H|\mathcal{S}||\mathcal{A}|T})$ (Jaksch *et al.*, 2010; Osband and Van Roy, 2016). Table 8.1 lists several model-based RL algorithms with proven *upper* regret bounds. UCRL2 (Jaksch *et al.*, 2010) obtains a regret bound of $\tilde{O}(H|\mathcal{S}|\sqrt{|\mathcal{A}|T})$, which Agrawal and Jia (2017) improve to $\tilde{O}(H\sqrt{|\mathcal{S}||\mathcal{A}|T})$ for large T. UCBVI (Azar *et al.*, 2017) and vUCQ (Kakade *et al.*, 2018) further improve these results, with UCBVI achieving $\tilde{O}(\sqrt{H|\mathcal{S}||\mathcal{A}|T} + \sqrt{H^2T})$. While these algorithms provide *worst-case* regret bounds, EULER (Zanette and Brunskill, 2019) actually matches the theoretical lower bound under additional assumptions on the variance of the optimal value function.

All previously discussed algorithms use a variant of *optimism in the face of uncertainty* (Lai and Robbins, 1985) in their algorithms, usually through upper confidence bounds on either estimated dynamics models or value functions. There is an alternative Bayesian approach known as *posterior sampling reinforcement learning* (PSRL) (Osband *et al.*, 2013), which instead estimates a Bayesian posterior and generally uses Thompson sampling (Thompson, 1933). Algorithms in this category (Osband *et al.*, 2013; Gopalan and Mannor, 2015; Osband and Van Roy, 2017) do use a Bayesian formulation of regret, which is less strict than the worst-case (minimax) regret. However, some Bayesian approaches also come with frequentist worst-case regret bounds (Agrawal and Jia,

Table 8.1: Minimax regret bounds for different model-based reinforcement learning algorithms. $|\mathcal{S}|$ and $|\mathcal{A}|$ denote the size of the state and action space, respectively. H denotes the MDP horizon, and T is the total number of samples of the algorithm. EULER uses additional assumptions on the variance of the optimal value function.

Algorithm	**Regret**
UCRL2 (Jaksch *et al.*, 2010)	$\tilde{O}(H\|\mathcal{S}\|\sqrt{\|\mathcal{A}\|T})$
(Agrawal and Jia, 2017) (for large T)	$\tilde{O}(H\sqrt{\|\mathcal{S}\|\|\mathcal{A}\|T})$
UCBVI (Azar *et al.*, 2017)	$\tilde{O}(\sqrt{H\|\mathcal{S}\|\|\mathcal{A}\|T} + H\sqrt{T})$
EULER (Zanette and Brunskill, 2019)	$\tilde{O}(\sqrt{H\|\mathcal{S}\|\|\mathcal{A}\|T})$
Model-free (Q-learning) (Jin *et al.*, 2018)	$\tilde{O}(\sqrt{H^3\|\mathcal{S}\|\|\mathcal{A}\|T})$
Lower bound (Jaksch *et al.*, 2010)	$\Omega(\sqrt{H\|\mathcal{S}\|\|\mathcal{A}\|T})$

2017).

Note that there are also regret bounds for model-free RL algorithms (Jin *et al.*, 2018) (Table 8.1), but these fall outside the scope of this survey. The main overall story is that model-based RL allows for better PAC/regret bounds than model-free RL, but also suffers from worse (computational) time and space complexity (since we need to estimate and store the transition function). Moreover, all previously discussed methods assume full planning, which also adds to the computational burden. However, Efroni *et al.* (2019b) interestingly show that one-step greedy planning, as for example used in Dyna (Sutton, 1990), can actually match the regret bounds of UCRL2 and EULER, while reducing its time and space complexity.

9

Related Work

While model-based RL has been successful and received much attention (Silver *et al.*, 2017b; Levine and Koltun, 2013; Deisenroth and Rasmussen, 2011), 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.

Moerland *et al.* (2020b) present a framework for reinforcement learning and planning that disentangles their common underlying dimensions, but does not focus on their integration. In some sense, Moerland *et al.* (2020b) look '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 *et al.* (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. 6 as well).

Finally, several authors (Nguyen-Tuong and Peters, 2011; Polydoros and Nalpantidis, 2017; Sigaud *et al.*, 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, 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.

10

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 *et al.*, 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 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 *et al.*, 2007), which has received less attention than its brother knowledge-based intrinsic motivation (see Sec. 7.2). 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 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 (Wang *et al.*, 2019; Moerland *et al.*, 2020a). 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 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.

11

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 3.2).
- 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).
- 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. 5).

- 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. 6).
- Model-based RL can have various benefits, including aspects like data efficiency, targeted exploration, transfer, safety and explainability (Sec. 7). 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 (Silver *et al.*, 2017b; Levine and Koltun, 2013) 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.

References

Abbeel, P. and A. Y. Ng. (2005). "Learning first-order Markov models for control". In: *Advances in neural information processing systems*. 1–8.

Achiam, J., H. Edwards, D. Amodei, and P. Abbeel. (2018). "Variational option discovery algorithms". *arXiv preprint arXiv:1807.10299*.

Achiam, J. and S. Sastry. (2017). "Surprise-based intrinsic motivation for deep reinforcement learning". *arXiv preprint arXiv:1703.01732*.

Agostinelli, F., S. McAleer, A. Shmakov, and P. Baldi. (2019). "Solving the Rubik's cube with deep reinforcement learning and search". *Nature Machine Intelligence*. 1(8): 356–363.

Agrawal, P., A. V. Nair, P. Abbeel, J. Malik, and S. Levine. (2016). "Learning to poke by poking: Experiential learning of intuitive physics". In: *Advances in Neural Information Processing Systems*. 5074–5082.

Agrawal, S. and R. Jia. (2017). "Posterior sampling for reinforcement learning: worst-case regret bounds". In: *Advances in Neural Information Processing Systems*. 1184–1194.

Amodei, D., C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. (2016). "Concrete problems in AI safety". *arXiv preprint arXiv:1606.06565*.

Anand, A., E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm. (2019). "Unsupervised state representation learning in atari". In: *Advances in Neural Information Processing Systems*. 8766–8779.

Anthony, T., R. Nishihara, P. Moritz, T. Salimans, and J. Schulman. (2019). "Policy Gradient Search: Online Planning and Expert Iteration without Search Trees". *arXiv preprint arXiv:1904.03646*.

Anthony, T., Z. Tian, and D. Barber. (2017). "Thinking fast and slow with deep learning and tree search". In: *Advances in Neural Information Processing Systems*. 5360–5370.

Asadi, K., E. Cater, D. Misra, and M. L. Littman. (2018). "Towards a Simple Approach to Multi-step Model-based Reinforcement Learning". *arXiv preprint arXiv:1811.00128*.

Asmuth, J., L. Li, M. L. Littman, A. Nouri, and D. 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. 19–26.

Åström, K. J. and P. Eykhoff. (1971). "System identification—a survey". *Automatica*. 7(2): 123–162.

Aswani, A., H. Gonzalez, S. S. Sastry, and C. Tomlin. (2013). "Provably safe and robust learning-based model predictive control". *Automatica*. 49(5): 1216–1226.

Atkeson, C. G., A. W. Moore, and S. Schaal. (1997). "Locally weighted learning for control". In: *Lazy learning*. Springer. 75–113.

Atkeson, C. G. and J. C. Santamaria. (1997). "A comparison of direct and model-based reinforcement learning". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. IEEE. 3557–3564.

Auer, P. (2002). "Using confidence bounds for exploitation-exploration trade-offs". *Journal of Machine Learning Research*. 3(Nov): 397–422.

Avila Belbute-Peres, F. de, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. (2018). "End-to-end differentiable physics for learning and control". In: *Advances in Neural Information Processing Systems*. 7178–7189.

Azar, M. G., I. Osband, and R. Munos. (2017). "Minimax regret bounds for reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 263–272.

Babaeizadeh, M., C. Finn, D. Erhan, R. H. Campbell, and S. Levine. (2017). "Stochastic variational video prediction". *arXiv preprint arXiv:1710.11252*.

Bacon, P.-L., J. Harb, and D. Precup. (2017). "The option-critic architecture". In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Bagnell, J. A. and J. 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. 1615–1620.

Baranes, A. and P.-Y. Oudeyer. (2009). "R-iac: Robust intrinsically motivated exploration and active learning". *IEEE Transactions on Autonomous Mental Development*. 1(3): 155–169.

Baranes, A. and P.-Y. Oudeyer. (2013). "Active learning of inverse models with intrinsically motivated goal exploration in robots". *Robotics and Autonomous Systems*. 61(1): 49–73.

Barreto, A., W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. (2017). "Successor features for transfer in reinforcement learning". In: *Advances in neural information processing systems*. 4055–4065.

Barto, A. G., S. J. Bradtke, and S. P. Singh. (1995). "Learning to act using real-time dynamic programming". *Artificial intelligence*. 72(1-2): 81–138.

Barto, A. G. and S. Mahadevan. (2003). "Recent advances in hierarchical reinforcement learning". *Discrete event dynamic systems*. 13(1-2): 41–77.

Battaglia, P., R. Pascanu, M. Lai, D. J. Rezende, *et al.* (2016). "Interaction networks for learning about objects, relations and physics". In: *Advances in neural information processing systems*. 4502–4510.

Battaglia, P. W., J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.* (2018). "Relational inductive biases, deep learning, and graph networks". *arXiv preprint arXiv:1806.01261*.

Baxter, J., A. Tridgell, and L. Weaver. (1999). "TDLeaf (lambda): Combining temporal difference learning with game-tree search". *arXiv preprint cs/9901001*.

Beck, J., K. Ciosek, S. Devlin, S. Tschiatschek, C. Zhang, and K. Hofmann. (2020). "Amrl: Aggregated memory for reinforcement learning". In:

Bellemare, M., S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. (2016). "Unifying count-based exploration and intrinsic motivation". In: *Advances in Neural Information Processing Systems*. 1471–1479.

Bellemare, M. G., Y. Naddaf, J. Veness, and M. Bowling. (2013). "The arcade learning environment: An evaluation platform for general agents". *Journal of Artificial Intelligence Research*. 47: 253–279.

Bellman, R. (1954). "The theory of dynamic programming". *Bulletin of the American Mathematical Society*. 60(6): 503–515.

Bellman, R. (1966). "Dynamic programming". *Science*. 153(3731): 34–37.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston. (2009). "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 41–48.

Berkenkamp, F., M. Turchetta, A. Schoellig, and A. Krause. (2017). "Safe model-based reinforcement learning with stability guarantees". In: *Advances in neural information processing systems*. 908–918.

Bertsekas, D. P., D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas. (1995). *Dynamic programming and optimal control*. Vol. 1. No. 2. Athena scientific Belmont, MA.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Blundell, C., B. Uria, A. Pritzel, Y. Li, A. Ruderman, J. Z. Leibo, J. Rae, D. Wierstra, and D. Hassabis. (2016). "Model-free episodic control". *arXiv preprint arXiv:1606.04460*.

Boone, G. (1997). "Efficient reinforcement learning: Model-based acrobot control". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 1. IEEE. 229–234.

Brafman, R. I. and M. Tennenholtz. (2002). "R-max-a general polynomial time algorithm for near-optimal reinforcement learning". *Journal of Machine Learning Research*. 3(Oct): 213–231.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. (2016). "OpenAI Gym". *arXiv preprint arXiv:1606.01540*.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. (2012). "A survey of monte carlo tree search methods". *IEEE Transactions on Computational Intelligence and AI in games*. 4(1): 1–43.

Brunskill, E. and L. Li. (2014). "Pac-inspired option discovery in lifelong reinforcement learning". In: *International conference on machine learning*. 316–324.

Buckman, J., D. Hafner, G. Tucker, E. Brevdo, and H. Lee. (2018). "Sample-efficient reinforcement learning with stochastic ensemble value expansion". In: *Advances in Neural Information Processing Systems*. 8224–8234.

Buesing, L., T. Weber, S. Racaniere, S. Eslami, D. Rezende, D. P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, *et al.* (2018). "Learning and querying fast generative models for reinforcement learning". *arXiv preprint arXiv:1802.03006*.

Caruana, R. (1997). "Multitask learning". *Machine learning*. 28(1): 41–75.

Castro, P. S. and D. Precup. (2007). "Using Linear Programming for Bayesian Exploration in Markov Decision Processes." In: *IJCAI*. Vol. 24372442.

Chang, M. B., T. Ullman, A. Torralba, and J. B. Tenenbaum. (2016). "A compositional object-based approach to learning physical dynamics". *arXiv preprint arXiv:1612.00341*.

Chentanez, N., A. G. Barto, and S. P. Singh. (2005). "Intrinsically motivated reinforcement learning". In: *Advances in neural information processing systems*. 1281–1288.

Chiappa, S., S. Racaniere, D. Wierstra, and S. Mohamed. (2017). "Recurrent environment simulators". *arXiv preprint arXiv:1704.02254*.

Choi, J., Y. Guo, M. Moczulski, J. Oh, N. Wu, M. Norouzi, and H. Lee. (2018). "Contingency-aware exploration in reinforcement learning". *arXiv preprint arXiv:1811.01483*.

Chrisman, L. (1992). "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach". In: *AAAI*. Vol. 1992. Citeseer. 183–188.

Christiano, P., Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. (2016). "Transfer from simulation to real world through learning deep inverse dynamics model". *arXiv preprint arXiv:1610.03518*.

Chua, K., R. Calandra, R. McAllister, and S. Levine. (2018). "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Advances in Neural Information Processing Systems*. 4754–4765.

Clavera, I., J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel. (2018). "Model-Based Reinforcement Learning via Meta-Policy Optimization". In: *Conference on Robot Learning*. 617–629.

Corneil, D., W. Gerstner, and J. Brea. (2018). "Efficient model-based deep reinforcement learning with variational state tabulation". *arXiv preprint arXiv:1802.04325*.

Coulom, R. (2006). "Efficient selectivity and backup operators in Monte-Carlo tree search". In: *International conference on computers and games*. Springer. 72–83.

Craik, K. J. W. (1943). "The Nature of Explanation".

Da Silva, B. C., E. W. Basso, A. L. Bazzan, and P. M. Engel. (2006). "Dealing with non-stationary environments using context detection". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 217–224.

Daniel, C., H. Van Hoof, J. Peters, and G. Neumann. (2016). "Probabilistic inference for determining options in reinforcement learning". *Machine Learning*. 104(2-3): 337–357.

Dann, C. and E. Brunskill. (2015). "Sample complexity of episodic fixed-horizon reinforcement learning". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*. 2818–2826.

Dayan, P. (1993). "Improving generalization for temporal difference learning: The successor representation". *Neural Computation*. 5(4): 613–624.

Dayan, P. and G. E. Hinton. (1993). "Feudal reinforcement learning". In: *Advances in neural information processing systems*. 271–278.

Dearden, R., N. Friedman, and D. Andre. (1999). "Model based Bayesian exploration". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 150–159.

Dearden, R., N. Friedman, and S. Russell. (1998). "Bayesian Q-learning". In: *AAAI/IAAI*. 761–768.

Degrave, J., M. Hermans, J. Dambre, *et al.* (2019). "A differentiable physics engine for deep learning in robotics". *Frontiers in neurorobotics*. 13.

Deisenroth, M. and C. 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)*. 465–472.

Depeweg, S., J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. (2016). "Learning and policy search in stochastic dynamical systems with bayesian neural networks". *arXiv preprint arXiv:1605.07127*.

Der Kiureghian, A. and O. Ditlevsen. (2009). "Aleatory or epistemic? Does it matter?" *Structural Safety*. 31(2): 105–112.

Dilokthanakul, N., C. Kaplanis, N. Pawlowski, and M. Shanahan. (2019). "Feature control as intrinsic motivation for hierarchical reinforcement learning". *IEEE transactions on neural networks and learning systems*.

Diuk, C., A. Cohen, and M. L. Littman. (2008). "An object-oriented representation for efficient reinforcement learning". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 240–247.

Doll, B. B., D. A. Simon, and N. D. Daw. (2012). "The ubiquity of model-based reinforcement learning". *Current opinion in neurobiology*. 22(6): 1075–1081.

Doya, K., K. Samejima, K.-i. Katagiri, and M. Kawato. (2002). "Multiple model-based reinforcement learning". *Neural computation*. 14(6): 1347–1369.

Duff, M. O. and A. Barto. (2002). "Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes". *PhD thesis*. University of Massachusetts at Amherst.

Ecoffet, A., J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. (2019). "Go-explore: a new approach for hard-exploration problems". *arXiv preprint arXiv:1901.10995*.

Edwards, A. D., L. Downs, and J. C. Davidson. (2018). "Forward-backward reinforcement learning". *arXiv preprint arXiv:1803.10227*.

Efroni, Y., G. Dalal, B. Scherrer, and S. Mannor. (2018). "Beyond the One-Step Greedy Approach in Reinforcement Learning". In: *International Conference on Machine Learning*. 1386–1395.

Efroni, Y., M. Ghavamzadeh, and S. Mannor. (2019a). "Multi-Step Greedy and Approximate Real Time Dynamic Programming". *arXiv preprint arXiv:1909.04236*.

Efroni, Y., N. Merlis, M. Ghavamzadeh, and S. Mannor. (2019b). "Tight Regret Bounds for Model-Based Reinforcement Learning with Greedy Policies". *arXiv preprint arXiv:1905.11527*.

El Hihi, S. and Y. Bengio. (1996). "Hierarchical recurrent neural networks for long-term dependencies". In: *Advances in neural information processing systems*. 493–499.

Evans, J. S. B. (1984). "Heuristic and analytic processes in reasoning". *British Journal of Psychology*. 75(4): 451–468.

Eysenbach, B., R. R. Salakhutdinov, and S. Levine. (2019a). "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". *Advances in Neural Information Processing Systems*. 32.

Eysenbach, B., A. Gupta, J. Ibarz, and S. Levine. (2019b). "Diversity is All You Need: Learning Skills without a Reward Function". In: *International Conference on Learning Representations*.

Fairbank, M. and E. Alonso. (2012). "Value-gradient learning". In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 1–8.

Farquhar, G., T. Rocktäschel, M. Igl, and S. Whiteson. (2018). "Treeqn and atreec: Differentiable tree planning for deep reinforcement learning". In: International Conference on Learning Representations.

Finn, C., I. Goodfellow, and S. Levine. (2016). "Unsupervised learning for physical interaction through video prediction". In: *Advances in neural information processing systems*. 64–72.

Florensa, C., Y. Duan, and P. Abbeel. (2017). "Stochastic neural networks for hierarchical reinforcement learning". *arXiv preprint arXiv:1704.03012*.

Florensa, C., D. Held, X. Geng, and P. Abbeel. (2018). "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference on Machine Learning*. 1514–1523.

Fortunato, M., M. Tan, R. Faulkner, S. Hansen, A. P. Badia, G. Buttimore, C. Deck, J. Z. Leibo, and C. Blundell. (2019). "Generalization of reinforcement learners with working and episodic memory". *arXiv preprint arXiv:1910.13406*.

Fox, R., S. Krishnan, I. Stoica, and K. Goldberg. (2017). "Multi-level discovery of deep options". *arXiv preprint arXiv:1703.08294*.

Fox, R., M. Moshkovitz, and N. Tishby. (2016). "Principled option learning in Markov decision processes". *arXiv preprint arXiv:1609.05524*.

Fraccaro, M., S. Kamronn, U. Paquet, and O. Winther. (2017). "A disentangled recognition and nonlinear dynamics model for unsupervised learning". In: *Advances in Neural Information Processing Systems*. 3601–3610.

Fragkiadaki, K., P. Agrawal, S. Levine, and J. Malik. (2015). "Learning visual predictive models of physics for playing billiards". *arXiv preprint arXiv:1511.07404*.

François-Lavet, V., Y. Bengio, D. Precup, and J. Pineau. (2019). "Combined reinforcement learning via abstract representations". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 3582–3589.

Frans, K., J. Ho, X. Chen, P. Abbeel, and J. Schulman. (2018). "Meta learning shared hierarchies". In: *International Conference on Learning Representations*.

Fröhlich, F., F. J. Theis, and J. Hasenauer. (2014). "Uncertainty analysis for non-identifiable dynamical systems: Profile likelihoods, bootstrapping and more". In: *International Conference on Computational Methods in Systems Biology*. Springer. 61–72.

Gal, Y., R. McAllister, and C. E. Rasmussen. (2016). "Improving PILCO with Bayesian neural network dynamics models". In: *Data-Efficient Machine Learning workshop, ICML*. Vol. 4.

Gemici, M., C.-C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos, and T. Lillicrap. (2017). "Generative temporal models with memory". *arXiv preprint arXiv:1702.04649*.

Gershman, S. J. and N. D. Daw. (2017). "Reinforcement learning and episodic memory in humans and animals: an integrative framework". *Annual review of psychology*. 68: 101–128.

Ghahramani, Z. and G. 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, Z. and S. T. Roweis. (1999). "Learning nonlinear dynamical systems using an EM algorithm". In: *Advances in neural information processing systems*. 431–437.

Ghavamzadeh, M., S. Mannor, J. Pineau, A. Tamar, *et al.* (2015). "Bayesian reinforcement learning: A survey". *Foundations and Trends® in Machine Learning*. 8(5-6): 359–483.

Ghosh, D., A. Gupta, and S. Levine. (2018). "Learning Actionable Representations with Goal-Conditioned Policies". *arXiv preprint arXiv:1811.07819*.

Goel, S. and M. Huber. (2003). "Subgoal discovery for hierarchical reinforcement learning using learned policies". In: *FLAIRS conference*. 346–350.

Goodfellow, I., Y. Bengio, and A. Courville. (2016). *Deep learning*. MIT press.

Gopalan, A. and S. Mannor. (2015). "Thompson sampling for learning parameterized markov decision processes". In: *Conference on Learning Theory*. PMLR. 861–898.

Graves, A., G. Wayne, and I. Danihelka. (2014). "Neural turing machines". *arXiv preprint arXiv:1410.5401*.

Gregor, K., D. J. Rezende, and D. Wierstra. (2016). "Variational intrinsic control". *arXiv preprint arXiv:1611.07507*.

Grimm, C., A. Barreto, S. Singh, and D. Silver. (2020). "The Value Equivalence Principle for Model-Based Reinforcement Learning". *Advances in Neural Information Processing Systems*.

Gu, S., T. Lillicrap, I. Sutskever, and S. Levine. (2016). "Continuous deep q-learning with model-based acceleration". In: *International Conference on Machine Learning*. 2829–2838.

Guestrin, C., D. Koller, C. Gearhart, and N. 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. 1003–1010.

Guez, A., M. Mirza, K. Gregor, R. Kabra, S. Racaniere, T. Weber, D. Raposo, A. Santoro, L. Orseau, T. Eccles, *et al.* (2019). "An Investigation of Model-Free Planning". In: *International Conference on Machine Learning*. 2464–2473.

Guez, A., D. Silver, and P. Dayan. (2012). "Efficient Bayes-adaptive reinforcement learning using sample-based search". In: *Advances in neural information processing systems*. 1025–1033.

Guez, A., T. Weber, I. Antonoglou, K. Simonyan, O. Vinyals, D. Wierstra, R. Munos, and D. Silver. (2018). "Learning to search with MCTSnets". *arXiv preprint arXiv:1802.04697*.

Guo, X., S. Singh, H. Lee, R. L. Lewis, and X. Wang. (2014). "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning". In: *Advances in neural information processing systems*. 3338–3346.

Ha, D. and J. Schmidhuber. (2018). "Recurrent world models facilitate policy evolution". In: *Advances in Neural Information Processing Systems*. 2450–2462.

Hafner, D., T. Lillicrap, J. Ba, and M. Norouzi. (2019a). "Dream to Control: Learning Behaviors by Latent Imagination". In: *International Conference on Learning Representations*.

Hafner, D., T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. (2019b). "Learning latent dynamics for planning from pixels". In: *International Conference on Machine Learning*. PMLR. 2555–2565.

Hamidi, M., P. Tadepalli, R. Goetschalckx, and A. Fern. (2015). "Active imitation learning of hierarchical policies". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Hamrick, J. B. (2019). "Analogues of mental simulation and imagination in deep learning". *Current Opinion in Behavioral Sciences*. 29: 8–16.

Hamrick, J. B., A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, and P. W. Battaglia. (2017). "Metacontrol for adaptive imagination-based optimization". *arXiv preprint arXiv:1705.02670*.

Hamrick, J. B., V. Bapst, A. Sanchez-Gonzalez, T. Pfaff, T. Weber, L. Buesing, and P. W. Battaglia. (2020). "Combining q-learning and search with amortized value estimates". *International Conference on Learning Representations (ICLR)*.

Hasselt, H. P. van, M. Hessel, and J. Aslanides. (2019). "When to use parametric models in reinforcement learning?" In: *Advances in Neural Information Processing Systems*. 14322–14333.

Hausman, K., J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. (2018). "Learning an Embedding Space for Transferable Robot Skills". In: *International Conference on Learning Representations*.

Heess, N., G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. (2016). "Learning and transfer of modulated locomotor controllers". *arXiv preprint arXiv:1610.05182*.

Heess, N., G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. (2015). "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2944–2952.

Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. (2018). "Deep reinforcement learning that matters". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hengst, B. (2017). "Hierarchical reinforcement learning". *Encyclopedia of Machine Learning and Data Mining*: 611–619.

Hester, T. and P. 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. 1–6.

Hester, T. and P. Stone. (2012b). "Learning and using models". In: *Reinforcement learning*. Springer. 111–141.

Hester, T. and P. Stone. (2013). "TEXPLORE: real-time sample-efficient reinforcement learning for robots". *Machine learning*. 90(3): 385–429.

Hochreiter, S. and J. Schmidhuber. (1997). "Long short-term memory". *Neural computation*. 9(8): 1735–1780.

Holland, G. Z., E. J. Talvitie, and M. Bowling. (2018). "The effect of planning shape on dyna-style planning in high-dimensional state spaces". *arXiv preprint arXiv:1806.01825*.

Houthooft, R., X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. (2016). "Vime: Variational information maximizing exploration". In: *Advances in Neural Information Processing Systems*. 1109–1117.

Hu, H., J. Ye, G. Zhu, Z. Ren, and C. Zhang. (2021). "Generalizable episodic memory for deep reinforcement learning". *arXiv preprint arXiv:2103.06469*.

Jaderberg, M., V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. (2016). "Reinforcement learning with unsupervised auxiliary tasks". *arXiv preprint arXiv:1611.05397*.

Jaksch, T., R. Ortner, and P. Auer. (2010). "Near-optimal Regret Bounds for Reinforcement Learning". *Journal of Machine Learning Research*. 11(4).

Janner, M., J. Fu, M. Zhang, and S. Levine. (2019). "When to trust your model: Model-based policy optimization". In: *Advances in Neural Information Processing Systems*. 12519–12530.

Jaulmes, R., J. Pineau, and D. Precup. (2005). "Learning in non-stationary partially observable Markov decision processes". In: *ECML Workshop on Reinforcement Learning in non-stationary environments*. Vol. 25. 26–32.

Jayaraman, D., F. Ebert, A. A. Efros, and S. Levine. (2018). "Time-agnostic prediction: Predicting predictable video frames". *arXiv preprint arXiv:1808.07784*.

Jiang, D., E. Ekwedike, and H. Liu. (2018). "Feedback-Based Tree Search for Reinforcement Learning". In: *International Conference on Machine Learning*. 2289–2298.

Jin, C., Z. Allen-Zhu, S. Bubeck, and M. I. Jordan. (2018). "Is Q-learning provably efficient?" In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 4868–4878.

Jong, N. K. and P. Stone. (2007). "Model-based function approximation in reinforcement learning". In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM. 95.

Jonschkowski, R. and O. Brock. (2015). "Learning state representations with robotic priors". *Autonomous Robots*. 39(3): 407–428.

Jordan, M. I. and D. E. Rumelhart. (1992). "Forward models: Supervised learning with a distal teacher". *Cognitive science*. 16(3): 307–354.

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

Kakade, S. and J. Langford. (2002). "Approximately optimal approximate reinforcement learning". In: *In Proc. 19th International Conference on Machine Learning*. Citeseer.

Kakade, S., M. Wang, and L. F. Yang. (2018). "Variance reduction methods for sublinear reinforcement learning".

Kakade, S. M. *et al.* (2003). "On the sample complexity of reinforcement learning". *PhD thesis*. University of London London, England.

Kalchbrenner, N., A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. (2017). "Video pixel networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 1771–1779.

Kalweit, G. and J. Boedecker. (2017). "Uncertainty-driven imagination for continuous deep reinforcement learning". In: *Conference on Robot Learning*. 195–206.

Kamthe, S. and M. P. Deisenroth. (2017). "Data-efficient reinforcement learning with probabilistic model predictive control". *arXiv preprint arXiv:1706.06491*.

Kansky, K., T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. 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. 1809–1818.

Karl, M., M. Soelch, J. Bayer, and P. van der Smagt. (2016). "Deep variational bayes filters: Unsupervised learning of state space models from raw data". *arXiv preprint arXiv:1605.06432*.

Ke, N. R., A. Singh, A. Touati, A. Goyal, Y. Bengio, D. Parikh, and D. Batra. (2019). "Learning Dynamics Model in Reinforcement Learning by Incorporating the Long Term Future". *arXiv preprint arXiv:1903.01599*.

Keramati, M., A. Dezfouli, and P. Piray. (2011). "Speed/accuracy trade-off between the habitual and the goal-directed processes". *PLoS computational biology*. 7(5).

Khansari-Zadeh, S. M. and A. Billard. (2011). "Learning stable non-linear dynamical systems with gaussian mixture models". *IEEE Transactions on Robotics*. 27(5): 943–957.

Kipf, T., E. van der Pol, and M. Welling. (2020). "Contrastive Learning of Structured World Models". In: *International Conference on Learning Representations*.

Kober, J., J. A. Bagnell, and J. Peters. (2013). "Reinforcement learning in robotics: A survey". *The International Journal of Robotics Research*. 32(11): 1238–1274.

Kocsis, L. and C. Szepesvári. (2006). "Bandit based monte-carlo planning". In: *ECML*. Vol. 6. Springer. 282–293.

Kolter, J. Z. and A. Y. Ng. (2009). "Near-Bayesian exploration in polynomial time". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 513–520.

Konidaris, G. and A. G. Barto. (2007). "Building Portable Options: Skill Transfer in Reinforcement Learning." In: *IJCAI*. Vol. 7. 895–900.

Konidaris, G., S. Kuindersma, R. Grupen, and A. Barto. (2012). "Robot learning from demonstration by constructing skill trees". *The International Journal of Robotics Research*. 31(3): 360–375.

Konidaris, G. D. (2006). "A framework for transfer in reinforcement learning". In: *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*.

Korf, R. E. (1990). "Real-time heuristic search". *Artificial intelligence*. 42(2-3): 189–211.

Krishnan, R. G., U. Shalit, and D. A. Sontag. (2015). "Deep Kalman Filters". *ArXiv*. abs/1511.05121.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 1097–1105.

Kulkarni, T. D., K. Narasimhan, A. Saeedi, and J. Tenenbaum. (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*. 3675–3683.

Kurniawati, H., D. Hsu, and W. S. Lee. (2008). "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces." In: *Robotics: Science and systems*. Vol. 2008. Zurich, Switzerland.

Kurutach, T., A. Tamar, G. Yang, S. J. Russell, and P. Abbeel. (2018). "Learning plannable representations with causal infogan". In: *Advances in Neural Information Processing Systems*. 8733–8744.

Lai, M. (2015). "Giraffe: Using deep reinforcement learning to play chess". *arXiv preprint arXiv:1509.01549*.

Lai, T. L. and H. Robbins. (1985). "Asymptotically efficient adaptive allocation rules". *Advances in applied mathematics*. 6(1): 4–22.

Lakshminarayanan, A. S., R. Krishnamurthy, P. Kumar, and B. Ravindran. (2016). "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering". *arXiv preprint arXiv:1605.05359*.

Lange, S., T. Gabel, and M. Riedmiller. (2012). "Batch reinforcement learning". In: *Reinforcement learning*. Springer. 45–73.

LaValle, S. M. (1998). "Rapidly-exploring random trees: A new tool for path planning".

Laversanne-Finot, A., A. Pere, and P.-Y. Oudeyer. (2018). "Curiosity Driven Exploration of Learned Disentangled Goal Spaces". In: *Conference on Robot Learning*. 487–504.

Lazaric, A. (2012). "Transfer in reinforcement learning: a framework and a survey". In: *Reinforcement Learning*. Springer. 143–173.

Lesort, T., N. Dıaz-Rodrıguez, J.-F. Goudou, and D. Filliat. (2018). "State representation learning for control: An overview". *Neural Networks*. 108: 379–392.

Levine, S. and P. Abbeel. (2014). "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 1071–1079.

Levine, S. and V. Koltun. (2013). "Guided policy search". In: *International Conference on Machine Learning*. 1–9.

Levy, A., R. Platt, and K. Saenko. (2019). "Hierarchical Reinforcement Learning with Hindsight". In: *International Conference on Learning Representations*.

Lin, L.-J. (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". *Machine learning*. 8(3-4): 293–321.

Lin, L.-J. (1993). "Reinforcement learning for robots using neural networks". *Tech. rep.* Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lin, L.-J. and T. M. Mitchell. (1992). *Memory approaches to reinforcement learning in non-Markovian domains.* Citeseer.

Lin, Z., T. Zhao, G. Yang, and L. Zhang. (2018). "Episodic memory deep Q-networks". *arXiv preprint arXiv:1805.07603.*

Ljung, L. (2001). "System identification". *Wiley Encyclopedia of Electrical and Electronics Engineering.*

Lopes, M., T. Lang, M. Toussaint, and P.-Y. Oudeyer. (2012). "Exploration in model-based reinforcement learning by empirically estimating learning progress". In: *Advances in neural information processing systems.* 206–214.

Lowrey, K., A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch. (2018). "Plan online, learn offline: Efficient learning and exploration via model-based control". *arXiv preprint arXiv:1811.01848.*

Loynd, R., M. Hausknecht, L. Li, and L. Deng. (2018). "Now I Remember! Episodic Memory For Reinforcement Learning".

Lu, K., I. Mordatch, and P. Abbeel. (2019). "Adaptive Online Planning for Continual Lifelong Learning". *arXiv preprint arXiv:1912.01188.*

Machado, M. C., M. G. Bellemare, and M. 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. 2295–2304.

Machado, M. C., M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. (2018). "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents". *Journal of Artificial Intelligence Research.* 61: 523–562.

Mahadevan, S. (2009). "Learning Representation and Control in Markov Decision Processes: New Frontiers". *Foundations and Trends® in Machine Learning.* 1(4): 403–565. ISSN: 1935-8237. DOI: 10.1561/2200000003.

Mann, T. and S. Mannor. (2014). "Scaling up approximate value iteration with options: Better policies with fewer iterations". In: *International conference on machine learning.* 127–135.

Mannor, S., I. Menache, A. Hoze, and U. Klein. (2004). "Dynamic abstraction in reinforcement learning via clustering". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 71.

Matiisen, T., A. Oliver, T. Cohen, and J. Schulman. (2017). "Teacher-student curriculum learning". *arXiv preprint arXiv:1707.00183*.

McCallum, R. (1997). "Reinforcement learning with selective perception and hidden state".

McGovern, A. and A. G. Barto. (2001). "Automatic discovery of subgoals in reinforcement learning using diverse density".

Menache, I., S. Mannor, and N. Shimkin. (2002). "Q-cut—dynamic discovery of sub-goals in reinforcement learning". In: *European Conference on Machine Learning*. Springer. 295–306.

Mishra, N., P. Abbeel, and I. Mordatch. (2017). "Prediction and control with temporal segment models". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2459–2468.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.* (2015). "Human-level control through deep reinforcement learning". *Nature*. 518(7540): 529.

Moerland, T. M., J. Broekens, and C. M. Jonker. (2017a). "Efficient exploration with double uncertain value networks". *Deep Reinforcement Learning Symposium, 31st Conference on Neural Information Processing Systems (NIPS)*.

Moerland, T. M., J. Broekens, and C. M. Jonker. (2017b). "Learning Multimodal Transition Dynamics for Model-Based Reinforcement Learning". *Scaling Up Reinforcement Learning (SURL) Workshop, European Conference on Machine Learning (ECML)*.

Moerland, T. M., J. Broekens, and C. M. Jonker. (2018a). "Emotion in reinforcement learning agents and robots: a survey". *Machine Learning*. 107(2): 443–480.

Moerland, T. M., J. Broekens, A. Plaat, and C. M. Jonker. (2018b). "A0C: Alpha zero in continuous action space". *Planning and Learning Workshop, 35th International Conference on Machine Learning (ICML)*.

Moerland, T. M., A. Deichler, S. Baldi, J. Broekens, and C. M. Jonker. (2020a). "Think Too Fast Nor Too Slow: The Computational Trade-off Between Planning And Reinforcement Learning". *arXiv preprint arXiv:2005.07404*.

Moerland, T. M., J. Broekens, and C. M. Jonker. (2020b). "A Framework for Reinforcement Learning and Planning". *arXiv preprint arXiv:2006.15009*.

Momennejad, I., E. M. Russek, J. H. Cheong, M. M. Botvinick, N. D. Daw, and S. J. Gershman. (2017). "The successor representation in human reinforcement learning". *Nature Human Behaviour*. 1(9): 680.

Moore, A. W. and C. G. Atkeson. (1993). "Prioritized sweeping: Reinforcement learning with less data and less time". *Machine learning*. 13(1): 103–130.

Müller, K.-R., A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. (1997). "Predicting time series with support vector machines". In: *International Conference on Artificial Neural Networks*. Springer. 999–1004.

Munos, R. (2003). "Error bounds for approximate policy iteration". In: *ICML*. Vol. 3. 560–567.

Nachum, O., S. S. Gu, H. Lee, and S. Levine. (2018). "Data-efficient hierarchical reinforcement learning". In: *Advances in Neural Information Processing Systems*. 3303–3313.

Nagabandi, A., I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. (2018a). "Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning". In: *International Conference on Learning Representations*.

Nagabandi, A., C. Finn, and S. Levine. (2018b). "Deep online learning via meta-learning: Continual adaptation for model-based RL". *arXiv preprint arXiv:1812.07671*.

Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine. (2018c). "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. 7559–7566.

Narendra, K. S. and K. Parthasarathy. (1990). "Identification and control of dynamical systems using neural networks". *IEEE Transactions on neural networks*. 1(1): 4–27.

Neitz, A., G. Parascandolo, S. Bauer, and B. Schölkopf. (2018). "Adaptive skip intervals: Temporal abstraction for recurrent dynamical models". In: *Advances in Neural Information Processing Systems*. 9816–9826.

Nguyen-Tuong, D. and J. Peters. (2011). "Model learning for robot control: a survey". *Cognitive processing*. 12(4): 319–340.

Nouri, A. and M. L. Littman. (2010). "Dimension reduction and its application to model-based exploration in continuous spaces". *Machine Learning*. 81(1): 85–98.

Oh, J., X. Guo, H. Lee, R. L. Lewis, and S. Singh. (2015). "Action-conditional video prediction using deep networks in atari games". In: *Advances in neural information processing systems*. 2863–2871.

Oh, J., S. Singh, and H. Lee. (2017). "Value prediction network". In: *Advances in Neural Information Processing Systems*. 6118–6128.

Osband, I., C. Blundell, A. Pritzel, and B. Van Roy. (2016). "Deep exploration via bootstrapped DQN". In: *Advances in Neural Information Processing Systems*. 4026–4034.

Osband, I., Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepezvari, S. Singh, B. V. Roy, R. Sutton, D. Silver, and H. V. Hasselt. (2019). "Behaviour Suite for Reinforcement Learning". arXiv: 1908.03568 [cs.LG].

Osband, I., D. Russo, and B. Van Roy. (2013). "(More) efficient reinforcement learning via posterior sampling". *arXiv preprint arXiv:1306.0940*.

Osband, I. and B. Van Roy. (2016). "On lower bounds for regret in reinforcement learning". *arXiv preprint arXiv:1608.02732*.

Osband, I. and B. Van Roy. (2017). "Why is posterior sampling better than optimism for reinforcement learning?" In: *International conference on machine learning*. PMLR. 2701–2710.

Ostafew, C. J., A. P. Schoellig, and T. D. Barfoot. (2016). "Robust constrained learning-based NMPC enabling reliable mobile robot path tracking". *The International Journal of Robotics Research*. 35(13): 1547–1563.

Ostrovski, G., M. G. Bellemare, A. van den Oord, and R. Munos. (2017). "Count-based exploration with neural density models". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2721–2730.

Oudeyer, P.-Y., F. Kaplan, and V. V. Hafner. (2007). "Intrinsic motivation systems for autonomous mental development". *IEEE transactions on evolutionary computation*. 11(2): 265–286.

Oudeyer, P.-Y., F. Kaplan, *et al.* (2008). "How can we define intrinsic motivation". In: *Proc. of the 8th Conf. on Epigenetic Robotics*. Vol. 5. 29–31.

Parlos, A. G., K. T. Chong, and A. F. Atiya. (1994). "Application of the recurrent multilayer perceptron in modeling complex process dynamics". *IEEE Transactions on Neural Networks*. 5(2): 255–266.

Parr, R., L. Li, G. Taylor, C. Painter-Wakefield, and M. 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. 752–759.

Pascanu, R., Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia. (2017). "Learning model-based planning from scratch". *arXiv preprint arXiv:1707.06170*.

Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell. (2017). "Curiosity-driven exploration by self-supervised prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 16–17.

Péré, A., S. Forestier, O. Sigaud, and P.-Y. Oudeyer. (2018). "Unsupervised learning of goal spaces for intrinsically motivated goal exploration". *arXiv preprint arXiv:1803.00781*.

Peshkin, L., N. Meuleau, and L. P. Kaelbling. (1999). "Learning Policies with External Memory". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc. 307–314.

Plaat, A., W. Kosters, and M. Preuss. (2020). "Model-Based Deep Reinforcement Learning for High-Dimensional Problems, a Survey". *arXiv preprint arXiv:2008.05598*.

Plappert, M., R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. (2017). "Parameter space noise for exploration". *arXiv preprint arXiv:1706.01905*.

Polydoros, A. S. and L. Nalpantidis. (2017). "Survey of model-based reinforcement learning: Applications on robotics". *Journal of Intelligent & Robotic Systems*. 86(2): 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.

Pritzel, A., B. Uria, S. Srinivasan, A. P. Badia, O. Vinyals, D. Hassabis, D. Wierstra, and C. Blundell. (2017). "Neural Episodic Control". In: *International Conference on Machine Learning*. 2827–2836.

Puterman, M. L. (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Racanière, S., T. Weber, D. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, *et al.* (2017). "Imagination-augmented agents for deep reinforcement learning". In: *Advances in neural information processing systems*. 5690–5701.

Ramani, D. (2019). "A short survey on memory based reinforcement learning". *arXiv preprint arXiv:1904.06736*.

Riemer, M., M. Liu, and G. Tesauro. (2018). "Learning abstract options". In: *Advances in Neural Information Processing Systems*. 10424–10434.

Roijers, D. M., P. Vamplew, S. Whiteson, and R. Dazeley. (2013). "A survey of multi-objective sequential decision-making". *Journal of Artificial Intelligence Research*. 48: 67–113.

Roijers, D. M. and S. Whiteson. (2017). "Multi-objective decision making". *Synthesis Lectures on Artificial Intelligence and Machine Learning*. 11(1): 1–129.

Rummery, G. A. and M. Niranjan. (1994). *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England.

Russell, S. J. and P. Norvig. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,

Samuel, A. L. (1967). "Some studies in machine learning using the game of checkers. II - Recent progress." *IBM Journal of research and development*. 11(6): 601–617.

Sawada, Y. (2018). "Disentangling Controllable and Uncontrollable Factors of Variation by Interacting with the World". *arXiv preprint arXiv:1804.06955*.

Schaul, T., D. Horgan, K. Gregor, and D. Silver. (2015). "Universal value function approximators". In: *International Conference on Machine Learning*. 1312–1320.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver. (2016). "Prioritized Experience Replay". In: *International Conference on Learning Representations (ICLR)*.

Schmidhuber, J. (1991). "Curious model-building control systems". In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. IEEE. 1458–1463.

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.* (2019). "Mastering atari, go, chess and shogi by planning with a learned model". *arXiv preprint arXiv:1911.08265*.

Sekar, R., O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. (2020). "Planning to Explore via Self-Supervised World Models". *arXiv preprint arXiv:2005.05960*.

Sequeira, P., F. S. Melo, and A. Paiva. (2014). "Learning by appraising: an emotion-based approach to intrinsic reward design". *Adaptive Behavior*. 22(5): 330–349.

Sermanet, P., C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain. (2018). "Time-contrastive networks: Self-supervised learning from video". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 1134–1141.

Sharma, A., S. Gu, S. Levine, V. Kumar, and K. Hausman. (2019). "Dynamics-Aware Unsupervised Discovery of Skills". In: *International Conference on Learning Representations*.

Shu, T., C. Xiong, and R. Socher. (2017). "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning". *arXiv preprint arXiv:1712.07294*.

Shyam, P., W. Jaśkowski, and F. Gomez. (2019). "Model-Based Active Exploration". In: *International Conference on Machine Learning*. 5779–5788.

Sigaud, O., C. Salaün, and V. Padois. (2011). "On-line regression algorithms for learning mechanical models of robots: a survey". *Robotics and Autonomous Systems*. 59(12): 1115–1129.

Silver, D., H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. 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. 3191–3199.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.* (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". *Science*. 362(6419): 1140–1144.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.* (2017b). "Mastering the game of go without human knowledge". *Nature*. 550(7676): 354.

Silver, D., R. S. Sutton, and M. Müller. (2008). "Sample-based learning and search with permanent and transient memories". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 968–975.

Silver, D. and J. Veness. (2010). "Monte-Carlo planning in large POMDPs". In: *Advances in neural information processing systems*. 2164–2172.

Şimşek, Ö., A. P. Wolfe, and A. G. Barto. (2005). "Identifying useful subgoals in reinforcement learning by local graph partitioning". In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 816–823.

Singh, S. P., T. Jaakkola, and M. I. Jordan. (1995). "Reinforcement learning with soft state aggregation". In: *Advances in neural information processing systems*. 361–368.

Spaan, M. T. and N. Spaan. (2004). "A point-based POMDP algorithm for robot planning". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 3. IEEE. 2399–2404.

Spelke, E. S. and K. D. Kinzler. (2007). "Core knowledge". *Developmental science*. 10(1): 89–96.

Srinivas, A., A. Jabri, P. Abbeel, S. Levine, and C. Finn. (2018). "Universal planning networks". *arXiv preprint arXiv:1804.00645*.

Stadie, B. C., S. Levine, and P. Abbeel. (2015). "Incentivizing exploration in reinforcement learning with deep predictive models". *arXiv preprint arXiv:1507.00814*.

Strehl, A. L., L. Li, and M. L. Littman. (2006). "Pac reinforcement learning bounds for rtdp and rand-rtdp". In: *Proceedings of AAAI workshop on learning for search*.

Sutton, R. S. (1990). "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine Learning Proceedings 1990*. Elsevier. 216–224.

Sutton, R. S. (1991). "Dyna, an integrated architecture for learning, planning, and reacting". *ACM Sigart Bulletin*. 2(4): 160–163.

Sutton, R. S. and A. G. Barto. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., D. Precup, and S. Singh. (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". *Artificial intelligence*. 112(1-2): 181–211.

Sutton, R. S., C. Szepesvári, A. Geramifard, and M. P. Bowling. (2012). "Dyna-style planning with linear function approximation and prioritized sweeping". *arXiv preprint arXiv:1206.3285*.

Sutton, R. S., C. Szepesvári, A. Geramifard, and M. 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*. Helsinki, Finland: AUAI Press. 528–536. ISBN: 0-9749039-4-9.

Szita, I. and C. Szepesvári. (2010). "Model-based reinforcement learning with nearly tight exploration complexity bounds". In: *ICML*.

Talvitie, E. (2014). "Model Regularization for Stable Sample Rollouts." In: *UAI*. 780–789.

Talvitie, E. (2017). "Self-correcting models for model-based reinforcement learning". In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Tamar, A., Y. Wu, G. Thomas, S. Levine, and P. Abbeel. (2016). "Value iteration networks". In: *Advances in Neural Information Processing Systems*. 2154–2162.

Taylor, M. E. and P. Stone. (2009). "Transfer learning for reinforcement learning domains: A survey". *Journal of Machine Learning Research*. 10(Jul): 1633–1685.

Tesauro, G. and G. 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. 1068–1074.

Tessler, C., S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. (2017). "A deep hierarchical approach to lifelong learning in minecraft". In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Thomas, V., E. Bengio, W. Fedus, J. Pondard, P. Beaudoin, H. Larochelle, J. Pineau, D. Precup, and Y. Bengio. (2018). "Disentangling the independently controllable factors of variation by interacting with the world". *arXiv preprint arXiv:1802.09484*.

Thompson, W. R. (1933). "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". *Biometrika*. 25(3/4): 285–294.

Thrun, S. and A. Schwartz. (1995). "Finding structure in reinforcement learning". In: *Advances in neural information processing systems*. 385–392.

Thrun, S. B. (1992). "Efficient exploration in reinforcement learning".

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. 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. 23–30.

Todorov, E. and W. 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. 300–306.

Tolman, E. C. (1948). "Cognitive maps in rats and men". *Psychological review*. 55(4): 189.

Van Hoof, H., N. Chen, M. Karl, P. van der Smagt, and J. 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. 3928–3934.

Van Seijen, H., H. Nekoei, E. Racah, and S. Chandar. (2020). "The LoCA Regret: A Consistent Metric to Evaluate Model-Based Behavior in Reinforcement Learning". *Advances in Neural Information Processing Systems*. 33.

Van Steenkiste, S., M. Chang, K. Greff, and J. Schmidhuber. (2018). "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions". *arXiv preprint arXiv:1802.10353*.

Vanseijen, H. and R. Sutton. (2015). "A deeper look at planning as learning from replay". In: *International conference on machine learning*. 2314–2322.

Veness, J., D. Silver, A. Blair, and W. Uther. (2009). "Bootstrapping from game tree search". In: *Advances in neural information processing systems*. 1937–1945.

Venkatraman, A., M. Hebert, and J. A. Bagnell. (2015). "Improving multi-step prediction of learned time series models". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Vezhnevets, A. S., S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. (2017). "Feudal networks for hierarchical reinforcement learning". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 3540–3549.

Waa, J. van der, J. van Diggelen, K. v. d. Bosch, and M. Neerincx. (2018). "Contrastive explanations for reinforcement learning in terms of expected consequences". *arXiv preprint arXiv:1807.08706*.

Wahlström, N., T. B. Schön, and M. P. Deisenroth. (2015). "From pixels to torques: Policy learning with deep dynamical models". *arXiv preprint arXiv:1502.02251*.

Wang, J., A. Hertzmann, and D. J. Fleet. (2006). "Gaussian process dynamical models". In: *Advances in neural information processing systems*. 1441–1448.

Wang, T., X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. (2019). "Benchmarking Model-Based Reinforcement Learning". *CoRR*. abs/1907.02057. arXiv: 1907.02057.

Watkins, C. J. and P. Dayan. (1992). "Q-learning". *Machine learning*. 8(3-4): 279–292.

Watson, J. S. (1966). "The development and generalization of" contingency awareness" in early infancy: Some hypotheses". *Merrill-Palmer Quarterly of Behavior and Development*. 12(2): 123–135.

Watter, M., J. Springenberg, J. Boedecker, and M. Riedmiller. (2015). "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in neural information processing systems*. 2746–2754.

Watters, N., L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner. (2019). "Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration". *arXiv preprint arXiv:1905.09275*.

Werbos, P. J. (1989). "Neural networks for control and system identification". In: *Proceedings of the 28th IEEE Conference on Decision and Control*. IEEE. 260–265.

Wiering, M. and J. Schmidhuber. (1998). "Efficient model-based exploration". In: *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*. Vol. 6. 223–228.

Wiering, M. A., M. Withagen, and M. M. Drugan. (2014). "Model-based multi-objective reinforcement learning". In: *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 1–6.

Willemsen, D., H. Baier, and M. Kaisers. (2020). "Value targets in off-policy AlphaZero: a new greedy backup". In: *Adaptive and Learning Agents (ALA) Workshop*.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine learning*. 8(3-4): 229–256.

Wilson, A., A. Fern, S. Ray, and P. Tadepalli. (2007). "Multi-task reinforcement learning: a hierarchical Bayesian approach". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 1015–1022.

Wolpert, D. M., Z. Ghahramani, and M. I. Jordan. (1995). "An internal model for sensorimotor integration". *Science*. 269(5232): 1880–1882.

Wu, J., I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. (2015). "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning". *Advances in neural information processing systems*. 28: 127–135.

Xu, Z., Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu. (2019). "Unsupervised discovery of parts, structure, and dynamics". *arXiv preprint arXiv:1903.05136*.

Yamaguchi, T., S. Nagahama, Y. Ichikawa, and K. Takadama. (2019). "Model-Based Multi-objective Reinforcement Learning with Unknown Weights". In: *International Conference on Human-Computer Interaction*. Springer. 311–321.

Yu, L., W. Zhang, J. Wang, and Y. Yu. (2017). "Seqgan: Sequence generative adversarial nets with policy gradient". In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Zanette, A. and E. Brunskill. (2019). "Tighter problem-dependent regret bounds in reinforcement learning without domain knowledge using value function bounds". In: *International Conference on Machine Learning*. PMLR. 7304–7312.

Zhang, L., G. Yang, and B. C. Stadie. (2021). "World model as a graph: Learning latent landmarks for planning". In: *International Conference on Machine Learning*. PMLR. 12611–12620.

Zhang, M., S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. (2019). "SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning". In: *International Conference on Machine Learning*. 7444–7453.

Zhu, Z., K. Lin, and J. Zhou. (2020). "Transfer Learning in Deep Reinforcement Learning: A Survey". *arXiv preprint arXiv:2009.07888*.

Ziegler, Z. M. and A. M. Rush. (2019). "Latent normalizing flows for discrete sequences". *arXiv preprint arXiv:1901.10548*.