Model-Based Deep Reinforcement Learning for High-Dimensional Problems, *a Survey*

Aske Plaat, Walter Kosters, Mike Preuss
Leiden Institute of Advanced Computer Science

Abstract—Deep reinforcement learning has shown remarkable success in the past few years. Highly complex sequential decision making problems have been solved in tasks such as game playing and robotics. Unfortunately, the sample complexity of most deep reinforcement learning methods is high, precluding their use in some important applications. Model-based reinforcement learning creates an explicit model of the environment dynamics to reduce the need for environment samples.

Current deep learning methods use high-capacity networks to solve high-dimensional problems. Unfortunately, high-capacity models typically require many samples, negating the potential benefit of lower sample complexity in model-based methods. A challenge for deep model-based methods is therefore to achieve high predictive power while maintaining low sample complexity.

In recent years, many model-based methods have been introduced to address this challenge. In this paper, we survey the contemporary model-based landscape. First we discuss definitions and relations to other fields. We propose a taxonomy based on three approaches: using explicit planning on given transitions, using explicit planning on learned transitions, and end-to-end learning of both planning and transitions. We use these approaches to organize a comprehensive overview of important recent developments such as latent models. We describe methods and benchmarks, and we suggest directions for future work for each of the approaches. Among promising research directions are curriculum learning, uncertainty modeling, and use of latent models for transfer learning.

Index Terms—Model-based reinforcement Learning, latent models, deep learning, machine learning, planning.

1 Introduction

Deep reinforcement learning has shown remarkable successes in the past few years. Applications in game playing and robotics have shown the power of this paradigm with applications such as learning to play Go from scratch or flying an acrobatic model helicopter [Abbeel et al., 2007, Mnih et al., 2015, Silver et al., 2016]. Reinforcement learning uses an environment from which training data is sampled; in contrast to supervised learning it does not need a large database of pre-labeled training data. This opens up many applications for machine learning for which no such database exists. Unfortunately, however, for most interesting applications many samples from the environment are necessary, and the computational cost of learning is prohibitive, a problem that is common in deep learning [LeCun et al., 2015]. Achieving faster learning is a major goal of much current research. Many promising approaches are tried, among them metalearning [Hospedales et al., 2020, Huisman et al., 2020], transfer learning [Pan et al., 2010], curriculum learning [Narvekar et al., 2020] and zero-shot learning [Xian et al., 2017]. The current paper focuses on model-based methods in deep reinforcement learning.

Model-based methods can reduce sample complexity. In contrast to model-free methods that sample at will from the environment, model-based methods build up a dynamics model of the environment as they sample. By using this dynamics model for policy updates, the number of necessary samples can be reduced substantially [Sutton, 1991]. Especially in robotics sample-efficiency is important (in games environment samples can often be generated more cheaply).

The success of the model-based approach hinges critically on the quality of the predictions of the dynamics model, and here the prevalence of deep learning presents a challenge [Talvitie, 2015]. Modeling the dynamics of high dimensional problems usually requires high capacity networks that, unfortunately, require many samples for training to achieve high generalization while preventing overfitting, potentially undoing the sample efficiency gains of model-based methods. Thus, the problem statement of the methods in this survey is *how to train a high-capacity dynamics model with high predictive power and low sample complexity*.

In addition to promising better sample efficiency than model-free methods, there is another reason for the interest in model-based methods for deep learning. Many problems in reinforcement learning are sequential decision problems, and learning the transition function is a natural way of capturing the core of long and complex decision sequences. This is what is called a forward model in game AI [Risi and Preuss, 2020, Torrado et al., 2018]. When a good transition function of the domain is present, then new, unseen, problems can be solved efficiently. Hence, model-based reinforcement learning may contribute to efficient transfer learning.

The contribution of this survey is to give an in-depth overview of recent methods for model-based deep reinforcement learning. We describe methods that use (1) explicit planning on given transitions, (2) explicit planning on a learned transition model, and (3) end-to-end learning of both planning and transitions. For each approach future directions are listed (specifically: latent models, uncertainty modeling, curriculum learning and multi-agent benchmarks).

Many research papers have been published recently, and the field of model-based deep reinforcement learning is advancing rapidly. The papers in this survey are selected on recency and impact on the field, for different applications, highlighting relationships between papers. Since our focus is on recent work, some of the references are to preprints in arXiv (of reputable groups). Excellent works with necessary background information

exist for reinforcement learning [Sutton and Barto, 2018], deep learning [Goodfellow et al., 2016], machine learning [Bishop, 2006], and artificial intelligence [Russell and Norvig, 2016]. As we mentioned, the main purpose of the current survey is to focus on deep learning methods, with high-capacity models. Previous surveys provide an overview of the uses of classic (non-deep) model-based methods [Deisenroth et al., 2013b, Kaelbling et al., 1996, Kober et al., 2013]. Other relevant surveys into model-based reinforcement learning are [Çalışır and Pehlivanoğlu, 2019, Hui, 2018, Justesen et al., 2019, Moerland et al., 2020b, Polydoros and Nalpantidis, 2017, Wang et al., 2019b].

The remainder of this survey is structured as follows. Section 2 provides necessary background and a familiar formalism of reinforcement learning. Section 3 then surveys recent papers in the field of model-based deep reinforcement learning. Section 4 introduces the main benchmarks of the field. Section 5 provides a discussion reflecting on the different approaches and provides open problems and future work. Section 6 concludes the survey.

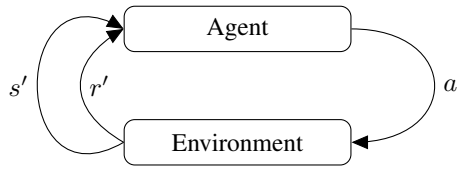

Fig. 1. Reinforcement Learning: Agent Acting on Environment, that provides new State and Reward to the Agent

2 Background

Reinforcement learning does not assume the presence of a database, as supervised learning does. Instead, it derives the ground truth from an internal model or from an external environment that can be queried by the learning agent, see Figure 1. The environment provides a new state s' and its reward r' (label) for every action a that the agent tries in a certain state s [Sutton and Barto, 2018]. In this way, as many action-reward pairs can be generated as needed, without a large hand-labeled database. Also, we can learn behavior beyond that what a supervisor prepared for us to learn.

As so much of artificial intelligence, reinforcement learning draws inspiration from principles of human and animal learning [Hamrick, 2019, Kahneman, 2011]. In psychology, learning is studied as behaviorial adaptation, as a result of reinforcing reward and punishment. Publications in artificial intelligence sometimes explicitly reference analogies in how learning in the two fields is described [Anthony et al., 2017, Duan et al., 2016, Weng, 2018].

Supervised learning frequently studies regression and classification problems. In reinforcement learning most problems are decision and control problems. Often problems are sequential decision problems, in which a goal is reached after a sequence of decisions are taken (behavior). In sequential decision making, the dynamics of the world are taken into consideration. Sequential decision making is a step-by-step approach in which earlier decisions influence later decisions. Before we continue, let us formalize key concepts in reinforcement learning.

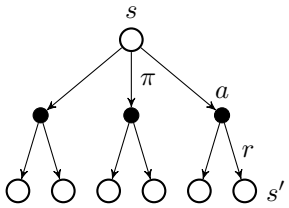

Fig. 2. Backup Diagram [Sutton and Barto, 2018]. Maximizing the reward for state s is done by following the *transition* function to find the next state s'. Note that the policy $\pi(s, a)$ tells the first half of this story, going from $s \rightarrow a$; the transition function $T_a(s, s')$ completes the story, going from $s \rightarrow s'$ (via a).

2.1 Formalizing Reinforcement Learning

Reinforcement learning problems are often modeled formally as a Markov Decision Process (MDP). First we introduce the basics: state, action, transition and reward. Then we introduce policy and value. Finally, we define model-based and model-free solution approaches.

A Markov Decision Process is a 4-tuple (S, A, T_a, R_a) where S is a finite set of states, A is a finite set of actions; $A_s \subseteq A$ is the set of actions available from state s. Furthermore, T_a is the transition function: $T_a(s, s')$ is the probability that action a in state s at time t will lead to state s' at time $t+1$. Finally, $R_a(s, s')$ is the immediate reward received after transitioning from state s to state s' due to action a. The goal of an MDP is to find the best decision, or action, in all states $s \in S$.

The goal of reinforcement learning is to find the optimal policy $a = \pi^\star(s)$, which is the function that gives the best action a in all states $s \in S$. The policy contains the actions of the answer to a sequential decision problem: a step-by-step prescription of which action must be taken in which state, in order to maximize reward for any given state. This policy can be found directly—model-free—or with the help of a transition model—model-based. Figure 2 shows a diagram of the transitions. More formally, the goal of an MDP is to find policy $\pi(s)$ that chooses an action in state s that will maximize the reward. This value V is the expected sum of future rewards $V^\pi(s) = E(\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}(s_t, s_{t+1}))$ that are discounted with parameter γ over t time periods, with $s = s_0$. The function $V^\pi(s)$ is called the value function of the state. In deep learning the policy π is determined by the parameters θ (or weights) of a neural network, and the parameterized policy is written as π_θ.

There are algorithms to compute the policy π directly, and there are algorithms that first compute this function $V^\pi(s)$. For stochastic problems often direct policy methods work best, for deterministic problems the value-methods are most often used [Kaelbling et al., 1996]. (A third, quite popular, approach combines the best of value and policy methods: actor-critic [Konda and Tsitsiklis, 2000, Mnih et al., 2016, Sutton and Barto, 2018].) In classical, table-based, reinforcement learning there is a close relation between policy and value, since the best action of a state leads to both the best policy and the best value, and finding the other can usually be done with a simple lookup. When the value and policy function are approximated, for example with a neural network, then this relation becomes weaker, and many advanced policy and value algorithms have been devised for deep reinforcement learning.

Value function algorithms calculate the state-action value

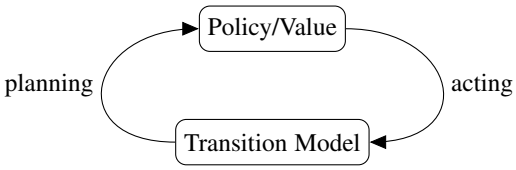

Fig. 3. Planning

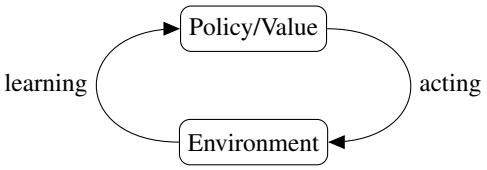

Fig. 4. Model-Free Learning

$Q^\pi(s,a)$. This Q-function gives the expected sum of discounted rewards when following action a in state s, and then afterwards policy π. The value $V(s)$ is the maximum of the $Q(s,a)$-values of that state. The optimal value-function is denoted as $V^\star(s)$. The optimal policy can be found by recursively choosing the argmax action with $Q(s,a) = V^\star(s)$ in each state.

To find the policy by planning, models for T and R must be known. When they are not known, an environment is assumed to be present for the agent to query in order to get the necessary reinforcing information, see Figure 1, after Sutton and Barto [2018]. The samples can be used to build the model of T and R (model-based reinforcement learning) or they can be used to find the policy without first building the model (direct or model-free reinforcement learning). When sampling, the environment is in a known state s, and the agent chooses an action a which it transmits to the environment, that responds with a new state s' and the corresponding reward value $r' = R_a(s,s')$.

The literature provides many solution algorithms. We now very briefly discuss classical planning and model-free approaches, before we continue to survey model-based algorithms in more depth in the next section.

2.2 Planning

Planning algorithms use the transition model to find the optimal policy, by selecting actions in states, looking ahead, and backing up reward values, see Figure 2 and Figure 3.

Algorithm 1 Value Iteration

```
Initialize V(s) to arbitrary values
repeat
    for all s do
        for all a do
            Q[s,a] = Σ_{s'} T_a(s,s')(R_a(s,s') + γV(s'))
        end for
        V[s] = max_a(Q[s,a])
    end for
until V converges
return V
```

In planning algorithms, the agent has access to an explicit transition and reward model. In the deterministic case the transition model provides the next state for each of the possible actions in the states, it is a function $s' = T_a(s)$. In the stochastic case, it provides the probability distribution $T_a(s,s')$. The reward model provides the immediate reward for transitioning from state s to state s' after taking action a. Figure 2 provides a backup diagram for the transition and reward function. The transition function moves downward in the diagram from state s to s', and the reward value goes upward in the diagram, backing up the value from the child state to the parent state. The transition function follows policy π with action a, after which state s' is chosen with probability p, yielding reward r'. The policy function $\pi(s,a)$ concerns the top layer of the diagram, from s to a. The transition function $T_a(s,s')$ covers both layers, from s to s'. In some domains, such as chess, there is a single deterministic state s' for each action a. Here each move leads to a single board position, simplifying the backup diagram.

Together, the transition and reward functions implicitly define a space of states that can be searched for the optimal policy $\pi^\star$ and value $V^\star$.

The most basic form of planning is Bellman's dynamic programming [Bellman, 1957, 2013], a recursive traversal of the state and action space. Value iteration is a well-known, very basic, dynamic programming method. The pseudo-code for value iteration is shown in Algorithm 1 [Alpaydin, 2020]. It traverses all actions in all states, computing the value of the entire state space.

Many planning algorithms have been devised to efficiently generate and traverse state spaces, such as (depth-limited) A*, alpha-beta and Monte Carlo Tree Search (MCTS) [Browne et al., 2012, Hart et al., 1968, Korf, 1985, Moerland et al., 2018, 2020c, Pearl, 1984, Plaat et al., 1996].

Planning algorithms originated from exact, table-based, algorithms [Sutton and Barto, 2018] that fit in the symbolic AI tradition. For planning it is relevant to know how much of the state space must be traversed to find the optimal policy. When state spaces are too large to search fully, deep function approximation algorithms can be used to approximate the optimal policy and value [Plaat, 2020, Sutton and Barto, 2018].

Planning is sample-efficient in the sense that, when the agent has a model, a policy can be found without interaction with the environment. Sampling may be costly, and sample efficiency is an important concept in reinforcement learning.

A sampling action taken in an environment is irreversible, since state changes of the environment can not be undone by the agent. In contrast, a planning action taken in a transition model is reversible [Moerland et al., 2020a]. A planning agent can backtrack, a sampling agent cannot. Sampling finds local optima easily. For finding global optima the ability to backtrack out of a local optimum is useful, which is an advantage for model-based planning methods.

Note, however, that there are two ways of finding dynamics models. In some problems, the transition and reward models are given by the problem, such as in games, where the move rules are known, as in Go and chess. Here the dynamics models follow the problem perfectly, and many steps can be planned accurately into the future without problem, out-performing model-free sampling. In other problems the dynamics model must be learned from sampling the environment. Here the model will not be perfect, and will contain errors and biases. Planning far ahead will only work when the agent has a T and R model of sufficient quality. With learned models, it may be more difficult for model-based planning to achieve the performance of model-free sampling.

2.3 Model-Free

When the transition or reward model are not available to the agent, then the policy and value function have to be learned through querying the environment. Learning the policy or value function without a model, through sampling the environment, is called model-free learning, see Figure 4.

Recall that the policy is a mapping of states to best actions. Each time when a new reward is returned by the environment the policy can be improved: the best action for the state is updated to reflect the new information. Algorithm 2 shows the simple high-level steps of model-free reinforcement learning (later on the algorithms become more elaborate).

Algorithm 2 Model-Free Learning

repeat
  Sample env E to generate data $D = (s, a, r', s')$
  Use D to update policy $\pi(s, a)$
until π converges

Model-free reinforcement learning is the most basic form of reinforcement learning. It has been successfully applied to a range of challenging problems [Deisenroth et al., 2013b, Kober et al., 2013]. In model-free reinforcement learning a policy is learned from the ground up through interactions (samples) with the environment.

The main goal of model-free learning is to achieve good generalization: to achieve high accuracy on test problems not seen during training. A secondary goal is to do so with good sample efficiency: to need as few environment samples as possible for good generalization.

Model-free learning is essentially blind, and learning the policy and value takes many samples. A well-known model-free reinforcement learning algorithm is Q-learning [Watkins, 1989]. Algorithms such as Q-learning can be used in a classical table-based setting. Deep neural networks have also been used with success in model-free learning, in domains in which samples can be generated cheaply and quickly, such as in Atari video games [Mnih et al., 2015]. Deep model-free algorithms such as Deep Q-Network (DQN) [Mnih et al., 2013] and Proximal Policy Optimzation (PPO) [Schulman et al., 2017] have become quite popular. PPO is an algorithm that computes the policy directly, DQN finds the value function first (Section 2.1).

Although risky, an advantage of flying blind is the absence of bias. Model-free reinforcement learning can find global optima without being distracted by a biased model (it has no model). Learned models in model-based reinforcement learning may introduce bias, and model-based methods may not always be able to find as good results as model-free can (although it does find the biased results with fewer samples).

Let us look at the cost of our methods. Interaction with the environment can be costly. Especially when the environment involves the real world, such as in real robot-interaction, then sampling should be minimized, for reasons of cost, and to prevent wear of the robot arm. In virtual environments on the other hand, model-free approaches have been quite successful, as we have noted in Atari and other game play [Mnih et al., 2015].

A good overview of model-free reinforcement learning can be found in [Çalışır and Pehlivanoğlu, 2019, Kaelbling et al., 1996, Sutton and Barto, 2018].

2.4 Model-Based

It is now time to look at model-based reinforcement learning, a method that learns the policy and value in a different way than by sampling the environment directly. Recall that the environment samples return (s', r') pairs, when the agent selects action a in state s. Therefore all information is present to learn the transition model $T_a(s, s')$ and the reward model $R_a(s, s')$, for example by supervised learning. When no transition model is given by the problem, then the model can be learned by sampling the environment, and be used with planning to update the policy and value as often as we like. This alternative approach of finding the policy and the value is called model-based learning.

If the model is given, then no environment samples are needed and model-based methods are more sample efficient. But if the model is not given, why would we want to go this convoluted model-and-planning route, if the samples can teach us the optimal policy and value directly? The reason is that the convoluted route may be more sample efficient.

When the complexity of learning the transition/reward model is smaller than the complexity of learning the policy model directly, and planning is fast, then the model-based route can be more efficient. In model-free learning a sample is used once to optimize the policy, and then thrown away, in model-based learning the sample is used to learn a transition model, which can then be used many times in planning to optimize the policy. The sample is used more efficiently.

The recent successes in deep learning caused much interest and progress in deep model-free learning. Many of the deep function approximation methods that have been so successful in supervised learning [Goodfellow et al., 2016, LeCun et al., 2015] can also be used in model-free reinforcement learning for approximating the policy and value function.

However, there are reasons for interest in model-based methods as well. Many real world problems are long and complex sequential decision making problems, and we are now seeing efforts to make progress in model-based methods. Furthermore, the interest in lifelong learning stimulates interest in model-based learning [Silver et al., 2013]. Model-based reinforcement learning is close to human and animal learning, in that all new knowledge is interpreted in the context of existing knowledge. The dynamics model is used to process and interpret new samples, in contrast to model-free learning, where all samples, old and new, are treated alike, and are not interpreted using the knowledge that has been accumulated so far in the model.

After these introductory words, we are now ready to take a deeper look into recent concrete deep model-based reinforcement learning methods.

3 Survey of Model-Based Deep Reinforcement Learning

The success of model-based reinforcement learning depends on the quality of the dynamics model. The model is typically used by planning algorithms for multiple sequential predictions, and errors in predictions accumulate quickly. We group the methods in three main approaches. First the transitions are given and used by explicit planning, second the transitions are learned and used by explicit planning, and third both transitions and planning are learned end-to-end:

1) **Explicit Planning on Given Transitions**
   First, we discuss methods for problems that give us

clear transition rules. In this case transition models are perfect, and classical, explicit, planning methods are used to optimize the value and policy functions of large state spaces. Recently, large and complex problems have been solved in two-agent games using self-learning methods that give rise to curriculum learning. Curriculum learning has also been used in single agent problems.

2) **Explicit Planning on Learned Transitions**
Second, we discuss methods for problems where no clear rules exist, and the transition model must be learned from sampling the environment. (The transitions are again used with conventional planning methods.) The environment samples allow learning by backpropagation of high-capacity models. It is important that the model has as few errors as possible. Uncertainty modeling and limited lookahead can reduce the impact of prediction errors.

3) **End-to-end Learning of Planning and Transitions**
Third, we discuss the situation where both the transition model and the planning algorithm are learned from the samples, end-to-end. A neural network can be used in a way that it performs the actual steps of certain planners, in addition to learning the transitions from the samples, as before. The model-based algorithm is learned fully end-to-end. A drawback of this approach is the tight connection between network architecture and problem type, limiting its applicabily. This drawback can be resolved with the use of latent models, see below.

In addition to the three main approaches, we now discuss two orthogonal approaches. These can be used to improve performance of the three main approaches. They are the hybrid imagination idea from Sutton's Dyna [Sutton, 1991], and abstract, or latent, models.

- **Hybrid Model-Free/Model-Based Imagination**
We first mention a sub-approach where environment samples are not only used to train the transition model, but also to train the policy function directly, just as in model-free learning. This hybrid approach thus combines model-based and model-free learning. It is also called *imagination* because the looking ahead with the dynamics model resembles simulating or imagining environment samples outside the real environment. In this approach the imagined, or planned, "samples" augment the real (environment) samples. This augmentation reduces sample complexity of model-free methods.
- **Latent Models**
Next, we discuss a sub-approach where the learned dynamics model is split into several lower-capacity, specialized, latent models. These latent models are then used with planning or imagination to find the policy. Latent models have been used with and without end-to-end model training and with and without imagination. Latent models thus build on and improve the previous approaches.

The different approaches can and have been used alone and in combination, as we will see shortly. Table 1 provides an overview of all approaches and methods that we will discuss in this survey. The methods are grouped into the three main categories that were introduced above. The use of the two orthogonal approaches by the methods (imagination and latent models) is indicated in Table 1 in two separate columns. The final column provides an indication of the application that the method is used on (such as Swimmer, Chess, and Cheetah). In the next section, Sect. 4, these applications will be explained in more depth.

All methods in the table will be explained in detail in the remainder of this section (for ease of reference, we will repeat the methods of each subsection in their own table). The sections will again mention some of the applications on which they were tested. Please refer to the section on Benchmarks.

Model-based methods work well for low-dimensional tasks where the transition and reward dynamics are relatively simple [Sutton and Barto, 2018]. While efficient methods such as Gaussian processes can learn these models quickly—with few samples—they struggle to represent complex and discontinuous systems [Wang et al., 2019b]. Most current model-free methods use deep neural networks to deal with problems that have such complex, high-dimensional, and discontinuous characteristics, leading to a high sample complexity.

The main challenge that the model-based reinforcement learning algorithms in this survey thus address is as follows. For high-dimensional tasks the curse of dimensionality causes data to be sparse and variance to be high. Deep methods tend to overfit on small datasets, and model-free methods use large data sets and have bad sample efficiency. Model-based methods that use poor models make poor planning predictions far into the future [Talvitie, 2015]. The challenge is to learn deep, high-dimensional transition functions from limited data, that can account for model uncertainty, and plan over these models to achieve policy and value functions that perform as well or better than model-free methods.

We will now discuss the algorithms. We will discuss (1) methods that use explicit planning on given transitions, (2) use explicit planning on a learned transition model, and (3) use end-to-end learning of planning and transitions. We will encounter the first occurrence of hybrid imagination and latent models approaches in the second section, on explicit planning/learned transitions.

3.1 Explicit Planning on Given Transitions

The first approach in model-based learning is when the transition and reward model is provided clearly in the rules of the problem. This is the case, for example, in games such as Go and chess. Table 2 summarizes the approaches of this subsection. Note the addition of the reinforcement learning method in an extra column.

With this approach high performing results have recently been achieved on large and complex domains. These results have been achieved by combining classical, explicit, heuristic search planning algorithms such as Alpha-beta and MCTS [Browne et al., 2012, Knuth and Moore, 1975, Plaat, 2020], and deep learning with self-play, achieving tabula rasa curriculum learning. Curriculum learning is based on the observation that a difficult problem is learned more quickly by first learning a sequence of easy, but related problems—just as we teach school children easy concepts (such as addition) first before we teach them harder concepts (such as multiplication, or logarithms).

In self-play the agent plays against the environment, which is also the same agent with the same network, see Figure 5. The states and actions in the games are then used by a deep learning system to improve the policy and value functions. These functions are used as the selection and evaluation functions in MCTS, and thus improving them improves the quality of play of MCTS. This has the effect that as the agent is getting smarter, so is the environment. A virtuous circle of a mutually increasing level of play is the result, a natural form of curriculum learning [Bengio

Approach	*Name*	*Learning*	*Planning*	*Hybrid Imagination*	*Latent Models*	*Application*
Explicit Planning	TD-Gammon [Tesauro, 1995a]	Fully connected net	Alpha-beta	-	-	Backgammon
Given Transitions	Expert Iteration [Anthony et al., 2017]	Policy/Value CNN	MCTS	-	-	Hex
(Sect. 3.1)	Alpha(Go) Zero [Silver et al., 2017a]	Policy/Value ResNet	MCTS	-	-	Go/chess/shogi
	Single Agent [Feng et al., 2020]	Resnet	MCTS	-	-	Sokoban
Explicit Planning	PILCO [Deisenroth and Rasmussen, 2011]	Gaussian Processes	Gradient based	-	-	Pendulum
Learned Transitions	iLQG [Tassa et al., 2012]	Quadratic Non-linear	MPC	-	-	Humanoid
(Sect. 3.2)	GPS [Levine and Abbeel, 2014]	iLQG	Trajectory	-	-	Swimmer
	SVG [Heess et al., 2015]	Value Gradients	Trajectory	-	-	Swimmer
	PETS [Chua et al., 2018]	Uncertainty Ensemble	MPC	-	-	Cheetah
	Visual Foresight [Finn and Levine, 2017]	Video Prediction	MPC	-	-	Manipulation
	Local Model [Gu et al., 2016]	Quadratic Non-linear	Short rollouts	+	-	Cheetah
	MVE [Feinberg et al., 2018]	Samples	Short rollouts	+	-	Cheetah
	Meta Policy [Clavera et al., 2018]	Meta-ensembles	Short rollouts	+	-	Cheetah
	GATS [Azizzadenesheli et al., 2018]	Pix2pix	MCTS	+	-	Cheetah
	Policy Optim [Janner et al., 2019]	Ensemble	Short rollouts	+	-	Cheetah
	Video-prediction [Oh et al., 2015]	CNN/LSTM	Action	+	+	Atari
	VPN [Oh et al., 2017]	CNN encoder	d-step	+	+	Atari
	SimPLe [Kaiser et al., 2019]	VAE, LSTM	MPC	+	+	Atari
	PlaNet [Hafner et al., 2018]	RSSM (VAE/RNN)	CEM	-	+	Cheetah
	Dreamer [Hafner et al., 2019]	RSSM+CNN	Imagine	-	+	Hopper
	Plan2Explore [Sekar et al., 2020]	RSSM	Planning	-	+	Hopper
End-to-End Learning	VIN [Tamar et al., 2016]	CNN	Rollout in network	+	-	Mazes
Planning & Transitions	VProp [Nardelli et al., 2018]	CNN	Hierarch Rollouts	+	-	Maze, nav
(Sect. 3.3)	TreeQN [Farquhar et al., 2018]	Tree-shape Net	Plan-functions	+	+	Box-push
	Planning [Guez et al., 2019]	CNN+LSTM	Rollouts in network	+	-	Sokoban
	I2A [Weber et al., 2017]	CNN/LSTM encoder	Meta-controller	+	+	Sokoban
	Predictron [Silver et al., 2017b]	k, γ, λ-CNN-predictr	k-rollout	+	+	Mazes
	World Model [Ha and Schmidhuber, 2018b]	VAE	CMA-ES	+	+	Car Racing
	MuZero [Schrittwieser et al., 2019]	Latent	MCTS	-	+	Atari/Go

TABLE 1
Overview of Deep Model-Based Reinforcement Learning Methods

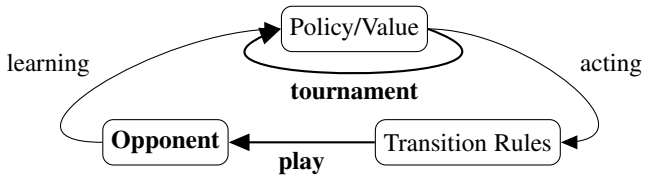

Fig. 5. Explicit Planning/Given Transitions

et al., 2009]. A sequence of ever-improving tournaments is played, in which a game can be learned to play from scratch, from zero-knowledge to world champion level [Silver et al., 2017a].

The concept of self-play was invented in multiple places and has a long history in two-agent game playing AI. Three well-known examples are Samuel's checkers player [Samuel, 1959], Tesauro's Backgammon player [Tesauro, 1995a, 2002] and DeepMind's Alpha(Go) Zero [Silver et al., 2017a, 2018].

Let us discuss some of the self-play approaches.

TD-Gammon [Tesauro, 1995a] is a Backgammon playing program that uses a small neural network with a single fully connected hidden layer with just 80 hidden units and a small (two-level deep) Alpha-beta search [Knuth and Moore, 1975]. It teaches itself to play Backgammon from scratch using temporal-difference learning. A small neural network learns the value function. TD-Gammon was the first Backgammon program to play at World-Champion level, and the first program to successfully use a self-learning curriculum approach in game playing since Samuel's checkers program [Samuel, 1959].

An approach similar to the AlphaGo and AlphaZero programs was presented as **Expert Iteration** [Anthony et al., 2017]. The problem was again how to learn to play a complex game from scratch. Expert Iteration (ExIt) combines search-based planning (the expert) with deep learning (by iteration). The expert finds improvements to the current policy. ExIt uses a single multi-task neural network, for the policy and the value function. The planner uses the neural network policy and value estimates to improve the quality of its plans, resulting in a cycle of mutual improvement. The planner in ExIt is MCTS. ExIt uses a version with rollouts. ExIt was used with the boardgame Hex [Hayward and Toft, 2019], and compared favorably against a strong MCTS-only program MoHex [Arneson et al., 2010]. A further development of ExIt is Policy Gradient Search, which uses planning without an explicit search tree [Anthony et al., 2019].

AlphaZero, and its predecessor AlphaGo Zero, are self-play curriculum learning programs that were developed by a team of researchers [Silver et al., 2017a, 2018]. The programs are desiged to play complex board games full of tactics and strategy well, specifically Go, chess, and shogi, a Japanese game similar to chess, but more complex [Iida et al., 2002]. AlphaZero and AlphaGo Zero are self-play model-based reinforcement learning programs. The environment against which they play is the same program as the agent that is learning to play. The transition function and the reward function are defined by the rules of the game. The goal is to learn optimal policy and value functions. AlphaZero uses a single neural network, a 19-block residual network with a value head and a policy head. For each different game—Go, chess, shogi—it uses different input and output layers, but the hidden layers are identical, and so is the rest of the architecture and the hyperparameters that govern the learning process. The loss-function is the sum of the policy-loss and the value-loss [Wang

Approach	*Learning*	*Planning*	*Reinforcement Learning*	*Application*
TD-Gammon [Tesauro, 1995a]	Fully connected net	Alpha-beta	Temporal Difference	Backgammon
Expert Iteration [Anthony et al., 2017]	Pol/Val CNN	MCTS	Curriculum	Hex
Alpha(Go) Zero [Silver et al., 2017a]	Pol/Val ResNet	MCTS	Curriculum	Go/chess/shogi
Single Agent [Feng et al., 2020]	ResNet	MCTS	Curriculum	Sokoban

TABLE 2
Overview of Explicit Planning/Given Transitionds Methods

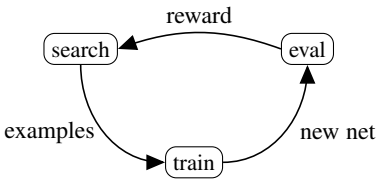

Fig. 6. Self-Play/Curriculum Learning Loop

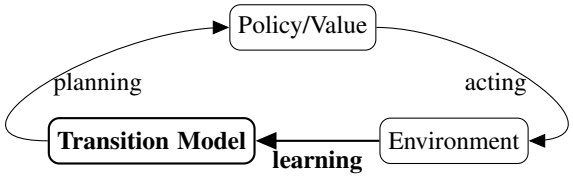

Fig. 7. Explicit Planning/Learned Transitions

et al., 2019a]. The planning algorithm is based on Monte Carlo Tree Search [Browne et al., 2012, Coulom, 2006] although it does not perform random rollouts. Instead, it uses the value head of the resnet for evaluation and the policy head of the ResNet to augment the UCT selection function [Kocsis and Szepesvári, 2006], as in P-UCT [Rosin, 2011]. The residual network is used in the evaluation and selection of MCTS. The self-play mechanism starts from a randomly initialized resnet. MCTS is used to play a tournament of games, to generate training positions for the resnet to be trained on, using a DQN-style replay buffer [Mnih et al., 2015]. This trained resnet is then again used by MCTS in the next training tournament to generate training positions, etc., see Figure 6. Self-play feeds on itself in multiple ways, and achieving stable learning is a challenging task, requiring judicious tuning, exploration, and much training. AlphaZero is currently the worldwide strongest player in Go, chess, and shogi [Silver et al., 2018].

The success of curriculum learning in two-player self-play has inspired work on **single-agent curriculum learning**. These single-agent approaches do not do self-play, but do use curriculum learning. Laterre et al. introduce the Ranked Reward method for solving bin packing problems [Laterre et al., 2018] and Wang et al. presented a method for Morpion Solitaire [Wang et al., 2020]. Feng et al. use an AlphaZero based approach to solve hard Sokoban instances [Feng et al., 2020]. Their model is an 8 block standard residual network, with MCTS as planner. Solving Sokoban instances is a hard problem in single-agent combinatorial search. The curriculum approach, where the agent learns to solve easy instances before it tries to solve harder instances, is a natural fit. In two-player games, a curriculum is generated in self-play. Feng et al. create a curriculum in a different way, by constructing simpler subproblems from hard instances, using the fact that Sokoban problems have a natural hierarchical structure. As in AlphaZero, the problem learns from scratch, no Sokoban heuristics are provided to the solver. This approach was able to solve harder Sokoban instances than had been solved before.

Conclusion

In self-play curriculum learning the opponent has the same model as the agent. The opponent is the environment of the agent. As the agent learns, so does its opponent, providing tougher counterplay, teaching the agent more. The agent is exposed to curriculum learning, a sequence of increasingly harder learning tasks. In this way, learning strong play has been achieved in Backgammon, Go, chess and shogi [Silver et al., 2018, Tesauro, 1995b].

In two-agent search a natural idea is to duplicate the agent as the environment, creating a self-play system. Self-play has been used in planning (as minimax), with policy learning, and in combination with latent models. Self-generated curriculum learning is a powerful paradigm. Work is under way to see if it can be applied to single-agent problems as well [Doan et al., 2019, Feng et al., 2020, Laterre et al., 2018, Narvekar et al., 2020], and in multi-agent (real-time strategy) games, addressing problems with specialization of two-agent games (Sect. 4.4 [Vinyals et al., 2019]).

Algorithm 3 Explicit Planning/Learned Transitions

repeat
  Sample env E to generate data $D = (s, a, r', s')$
  Use D to learn $T_a(s, s')$
  Use T to update policy $\pi(s, a)$ by planning
until π converges

3.2 Explicit Planning on Learned Transitions

In the previous section, transition rules could be derived from the problem directly (by inspection). In many problems, this is not the case, and we have to resort to sampling the environment to learn a model of the transitions. The second category of algorithms of this survey is to learn the tranition model by backpropagation from environment samples. This learned model is then still used by classical, explicit, planning algorithms, as before. We will discuss various approaches where the transition model is learned with supervised learning methods such as backpropagation through time [Werbos, 1988], see Figure 7.

Algorithm 3 shows the steps of using explicit planning and transition learning by backpropagation. Table 3 summarizes the approaches of this subsection, showing both the *learning* and the *planning* approach. Two variants of this approach are also discussed in this subsection: hybrid imagination and latent models, see Table 4 and Table 5.

We will first see how simple Gaussian Processes and quadratic methods can create predictive transition models. Next, precision is improved with trajectory methods, and we make the step to video prediction methods. Finally, methods that focus on uncertainty and

Approach	*Learning*	*Planning*	*Application*
PILCO [Deisenroth and Rasmussen, 2011]	Gaussian Processes	Gradient based	Pendulum
iLQG [Tassa et al., 2012]	Quadratic Non-linear	MPC	Humanoid
GPS [Levine and Abbeel, 2014]	iLQG	Trajectory	Swimmer
SVG [Heess et al., 2015]	Value Gradients	Trajectory	Swimmer
PETS [Chua et al., 2018]	Uncertainty Ensemble	MPC	Cheetah
Visual Foresight [Finn and Levine, 2017]	Video Prediction	MPC	Manipulation

TABLE 3
Overview of Explicit Planning/Learned Transitions Methods

ensemble methods will be introduced. We know that deep neural nets need much data and learn slowly, or will overfit. Uncertainty modeling is based on the insight that early in the training the model has seen little data, and tends to overfit, and later on, as it has seen more data, it may underfit. This issue can be mitigated by incorporating uncertainty into the dynamics models, as we shall see in the later methods [Chua et al., 2018].

For smaller models, environment samples can be used to approximate a transition model as a **Gaussian Process** of random variables. This approach is followed in PILCO, which stands for Probabilistic Inference for Learning Control, see [Deisenroth and Rasmussen, 2011, Deisenroth et al., 2013a, Kamthe and Deisenroth, 2017]. Gaussian Processes can accurately learn simple processes with good sample efficiency [Bishop, 2006], although for high dimensional problems they need more samples. PILCO treats the transition model $T_a(s, s')$ as a probabilistic function of the environment samples. The planner improves the policy based on the analytic gradients relative to the policy parameters θ. PILCO has been used to optimize small problems such as Mountain car and Cartpole pendulum swings, for which it works well. Although they achieve model learning using higher order model information, Gaussian Processes do not scale to high dimensional environments, and the method is limited to smaller applications.

A related method uses a trajectory optimization approach with nonlinear least-squares optimization. In control theory, the linear–quadratic–Gaussian (LQG) control problem is one of the most fundamental optimal control problems. **Iterative LQG** [Tassa et al., 2012] is the control analog of the Gauss-Newton method for nonlinear least-squares optimization. In contrast to PILCO, the model learner uses quadratic approximation on the reward function and linear approximation of the transition function. The planning part of this method uses a form of online trajectory optimization, model-predictive control (MPC), in which step-by-step real-time local optimization is used, as opposed to full-problem optimization [Richards, 2005]. By using many further improvements throughout the MPC pipeline, including the trajectory optimization algorithm, the physics engine, and cost function design, Tassa et al. were able to achieve near-real-time performance in humanoid simulated robot manipulation tasks, such as grasping.

Another trajectory optimization method takes its inspiration from model-free learning. Levine and Koltun [Levine and Koltun, 2013] introduce **Guided Policy Search** (GPS) in which the search uses trajectory optimization to avoid poor local optima. In GPS, the parameterized policy is trained in a supervised way with samples from a trajectory distribution. The GPS model optimizes the trajectory distribution for cost and the current policy, to create a good training set for the policy. Guiding samples are generated by differential dynamic programming and are incorporated into the policy with regularized importance sampling. In contrast to the previous methods, GPS algorithms can train complex policies with thousands of parameters. In a sense, Guided Policy Search transforms the iLQG controller into a neural network policy π_θ with a trust region in which the new controller does not deviate too much from the samples [Finn et al., 2016, Levine and Abbeel, 2014, Montgomery and Levine, 2016]. GPS has been evaluated on planar swimming, hopping, and walking, as well as simulated 3D humanoid running.

Another attempt at increasing the accuracy of learned parameterized transition models in continuous control problems is **Stochastic Value Gradients** (SVG) [Heess et al., 2015]. It mitigates learned model inaccuracy by computing value gradients along the real environment trajectories instead of planned ones. The mismatch between predicted and real transitions is addressed with re-parametrization and backpropagation through the stochastic samples. In comparison, PILCO uses Gaussian process models to compute analytic policy gradients that are sensitive to model-uncertainty and GPS optimizes policies with the aid of a stochastic trajectory optimizer and locally-linear models. SVG in contrast focuses on global neural network value function approximators. SVG results are reported on simulated robotics applications in Swimmer, Reacher, Gripper, Monoped, Half-Cheetah, and Walker.

Other methods also focus on uncertainty in high dimensional modeling, but use ensembles. Chua et al. propose **probabilistic ensembles with trajectory sampling** (PETS) [Chua et al., 2018]. The learned transition model of PETS has an uncertainty-aware deep network, which is combined with sampling-based uncertainty propagation. PETS uses a combination of probabilistic ensembles [Lakshminarayanan et al., 2017]. The dynamics are modelled by an ensemble of probabilistic neural network models in a model-predictive control setting (the agent only applies the first action from the optimal sequence and re-plans at every time-step) [Nagabandi et al., 2018]. Chua et al. report experiments on simulated robot tasks such as Half-Cheetah, Pusher, Reacher. Performance on these tasks is reported to approach asymptotic model-free baselines, stressing the importance of uncertainty estimation in model-baed reinforcement learning.

An important problem in robotics is to learn arm manipulation directly from video camera input by seeing which movements work and which fail. The video input provides a high dimensional and difficult input and increases problem size and complexity substantially. Both Finn et al. and Ebert et al. report on learning complex robotic manipulation skills from high-dimenstional raw sensory pixel inputs in a method called **Visual Foresight** [Ebert et al., 2018, Finn and Levine, 2017]. The aim of Visual Foresight is to generalize deep learning methods to never-before-seen tasks and objects. It uses a training procedure where data is sampled according to a probability distribution. Concurrently, a video prediction model is trained with the samples. This model generates

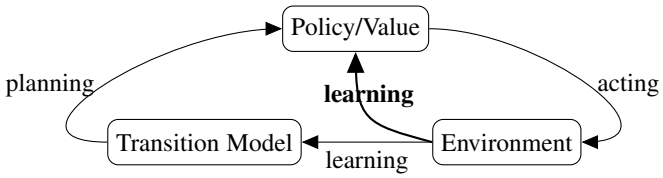

Fig. 8. Hybrid Model-Free/Model-Based Imagination

Algorithm 4 Hybrid Model-Free/Model-Based Imagination

repeat
  Sample env E to generate data $D = (s, a, r', s')$
  Use D to update policy $\pi(s, a)$
  Use D to learn $T_a(s, s')$
  Use T to update policy $\pi(s, a)$ by planning
until π converges

the corresponding sequence of future frames based on an image and a sequence of actions, as in GPS. At test time, the least-cost sequence of actions is selected in a model-predictive control planning framework. Visual Foresight is able to perform multi-object manipulation, pushing, picking and placing, and cloth-folding tasks.

Conclusion

Model learning with a single network works well for low-dimensional problems. We have seen that Gaussian Process modeling achieves sample efficiency and generalization to good policies. For high-dimensional problems, generalization and sample efficiency deteriorate, more samples are needed and policies do not perform as well. We have discussed methods for improvement by guiding policies with real samples (GPS), limiting the scope of predictions with model-predictive control, and using ensembles and uncertainty aware neural networks to model uncertainty (PETS).

3.2.1 Hybrid Model-Free/Model-Based Imagination

In the preceding subsection, we have looked at how to use environment samples to build a transition model. Many methods were covered to learn transition models with as few samples as possible. These methods are related to supervised learning methods. The transition model was then used by a planning method to optimize the policy or value function.

We will now review methods that use a complementary approach, a hybrid model-based/model-free approach of using the environment samples for two purposes. Here the emphasis is no longer on learning the model but on using it effectively. This approach was introduced by Sutton [Sutton, 1990, 1991] in the Dyna system, long before deep learning was used widely. Dyna uses the samples to update the policy function directly (model-free learning) and also uses the samples to learn a transition model, which is then used by planning to augment the model-free environment-samples with the model-based imagined "samples." In this way the sample-efficiency of model-free learning is improved quite directly. Figure 8 illustrates the working of the Dyna approach. (Note that now two arrows learn from the environment samples. Model-free learning is drawn bold.)

Dyna was introduced for a table-based approach before deep learning became popular. Originally, in Dyna, the transition model is updated directly with samples, without learning through back-propagation, however, here we discuss only deep imagination methods. Algorithm 4 shows the steps of the algorithm (compared to Algorithm 3, the line in italics is new, from Algorithm 2). Note how the policy is updated twice in each iteration, by environment sampling, and by transition planning.

We will describe five deep learning approaches that use imagination to augment the sample-free data. Table 4 summarizes these approaches. Note that in the next subsection more methods are described that also use hybrid model-free/model-based updating of the policy function. These are also listed in Table 4. We will first see how quadratic methods are used with imagination rollouts. Next, short rollouts are introduced, and ensembles, to improve the precision of the predictive model. Imagination is a hybrid model-free/model-based approach, we will see methods that build on successful model-free deep learning approaches such as meta-learning and generative-adversarial networks.

Let us have a look at how Dyna-style imagination works in deep model-based algorithms. Earlier, we saw that linear–quadratic–Gaussian methods were used to improve model learning. Gu et al. merge the backpropagation iLQG aproaches with Dyna-style synthetic policy rollouts [Gu et al., 2016]. To accelerate model-free continuous Q-learning they combine **locally linear models** with local on-policy imagination rollouts. The paper introduces a version of continuous Q-learning called normalized advantage functions, accelerating the learning with imagination rollouts. Data efficiency is improved with model-guided exploration using off-policy iLQG rollouts. As application the approach has been tested on simulated robotics tasks such as Gripper, Half-Cheetah and Reacher.

Feinberg et al. present **model-based value expansion** (MVE) which, like the previous algorithm [Gu et al., 2016], controls for uncertainty in the deep model by only allowing imagination to fixed depth [Feinberg et al., 2018]. Value estimates are split into a near-future model-based component and a distant future model-free component. In contrast to stochastic value gradients (SVG), MVE works without differentiable dynamics, which is important since transitions can include non-differentiable contact interactions [Heess et al., 2015]. The planning part of MVE uses short rollouts. The overall reinforcement learning algorithm that is used is a combined value-policy actor-critic setting [Sutton and Barto, 2018] and deep deterministic policy gradients (DDPG) [Lillicrap et al., 2015]. As application re-implementations of simulated robotics were used such as for Cheetah, Swimmer and Walker.

An ensemble approach has been used in combination with gradient-based meta-learning by Clavera et al. who introduced Model-based Reinforcement Learning via **Meta-Policy Optimization** (MP-MPO) [Clavera et al., 2018]. This method learns an ensemble of dynamics models and then it learns a policy that can be adapted quickly to any of the fitted dynamics models with one gradient step (the MAML-like meta-learning step [Finn et al., 2017]). MB-MPO frames model-based reinforcement learning as meta-learning a policy on a distribution of dynamic models, in the form of an ensemble of the real environment dynamics. The approach builds on the gradient-based meta-learning framework MAML [Finn et al., 2017]. The planning part of the algorithm samples imagined trajectories. MB-MPO is evaluated on continuous control benchmark tasks in a robotics simulator: Ant, Half-Cheetah, Hopper, Swimmer, Walker. The results reported indicate that meta-learning a policy over an ensemble of learned models approaches the level of performance of model-free methods with

Approach	*Learning*	*Planning*	*Reinforcement Learning*	*Application*
Local Model [Gu et al., 2016]	Quadratic Non-linear	Short rollouts	Q-learning	Cheetah
MVE [Feinberg et al., 2018]	Samples	Short rollouts	Actor-critic	Cheetah
Meta Policy [Clavera et al., 2018]	Meta-ensembles	Short rollouts	Policy optimization	Cheetah
GATS [Azizzadenesheli et al., 2018]	Pix2pix	MCTS	Deep Q Network	Cheetah
Policy Optim [Janner et al., 2019]	Ensemble	Short rollouts	Soft-Actor-Critic	Cheetah
Video predict [Oh et al., 2015]	CNN/LSTM	Action	Curriculum	Atari
VPN [Oh et al., 2017]	CNN encoder	d-step	k-step	Mazes, Atari
SimPLe [Kaiser et al., 2019]	VAE, LSTM	MPC	PPO	Atari

TABLE 4
Overview of Hybrid Model-Free/Model-based Imagination Methods

substantially better sample complexity.

Another attempt to improve the accuracy and efficiency of dynamics models has been through generative adversarial networks [Goodfellow et al., 2014]. Azizzadenesheli et al. aim to combine successes of **generative adversarial networks** with planning robot motion in model-based reinforcement learning [Azizzadenesheli et al., 2018]. Manipulating robot arms based on video input is an important application in AI (see also Visual Foresight in Section 3.2, and the SimPLe approach, in Section 3.2.2). A generative dynamics model is introduced to model the transition dynamics based on the pix2pix architecture [Isola et al., 2017]. For planning Monte Carlo Tree Search [Browne et al., 2012, Coulom, 2006] is used. GATS is evaluated on Atari games such as Pong, and does not perform better than model-free DQN [Mnih et al., 2015].

Achieving a good performing high-dimensional predictive model remains a challenge. Janner et al. propose in Model-based Policy Optimization (MBPO) a new approach to **short rollouts with ensembles** [Janner et al., 2019]. In this approach the model horizon is much shorter than the task horizon. These model rollouts are combined with real samples, and matched with plausible environment observations [Kalweit and Boedecker, 2017]. MBPO uses an ensemble of probabilistic networks, as in PETS [Chua et al., 2018]. Soft-actor-critic [Haarnoja et al., 2018] is used as reinforcement learning method. Experiments show that the policy optimization algorithm learns substantially faster with short rollouts than other algorithms, while retaining asymptotic performance relative to model-free algorithms. The applications used are simulated robotics tasks: Hopper, Walker, Half-Cheetah, Ant. The method surpasses the sample efficiency of prior model-based algorithms and matches the performance of model-free algorithms.

Conclusion

The hybrid imagination methods aim to combine the advantage of model-free methods with model-based methods in a hybrid approach augmenting "real" with "imagined" samples, to improve sample effciency of deep model-free learning. A problem is that inaccuracies in the model may be enlarged in the planned rollouts. Most methods limited lookahead to local lookahead. We have discussed interesting approaches combining meta-learning and generative-adversarial networks, and ensemble methods learning robotic movement directly from images.

3.2.2 Latent Models

The next group of methods that we describe are the latent or abstract model algorithms. Latent models are born out of the need for more accurate predictive deep models. Latent models replace the single transition model with separate, smaller, specialized, representation models, for the different functions in a reinforcement learning algorithm. All of the elements of the MDP-tuple may now get their own model. Planning occurs in latent space.

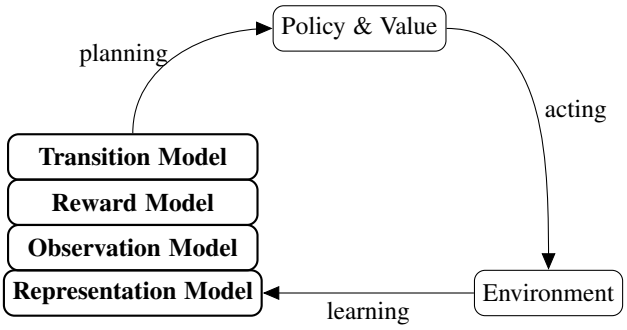

Fig. 9. Latent Models

Traditional deep learning models represent input states directly in a single model: the layers of neurons and filters are all related in some way to the input and output of the domain, be it an image, a sound, a text or a joystick action or arm movement. All of the MDP functions, state, value, action, reward, policy, and discount, act on this single model. Latent models, on the other hand, are not connected directly to the input and output, but are connected to other models and signals. They do not work on direct representations, but on latent, more compact, representations. The interactions are captured in three to four different models, such as observation, representation, transition, and reward models. These may be smaller, lower capacity, models. They may be trained with unsupervised or self-supervised deep learning such as variational autoencoders [Kingma and Welling, 2013, 2019] or generative adversarial networks [Goodfellow et al., 2014], or recurrent networks. Latent models use multiple specialized networks, one for each function to be approximated. The intuition behind the use of latent models is dimension reduction: they can better specialize and thus have more precise predictions, or can better capture the essence of higher level reasoning in the input domains, and need fewer samples (without overfitting) due to their lower capacity.

Figure 9 illustrates the abstract (latent) learning process (using the modules of Dreamer [Hafner et al., 2019] as an example). Table 5 summarizes the methods of this subsection (three are also mentioned in Table 4). Quite a few different latent (abstract) model approaches have been published. Latent models work well, both for games and robotics. Different rollout methods are proposed, such as local rollouts, and differentiable imagination, and end-to-end model learning and planning. Finally, latent models are

Approach	*Learning*	*Planning*	*Reinforcement Learning*	*Application*
Video predict [Oh et al., 2015]	CNN/LSTM	Action	Curriculum	Atari
VPN [Oh et al., 2017]	CNN encoder	d-step	k-step	Mazes, Atari
SimPLe [Kaiser et al., 2019]	VAE, LSTM	MPC	PPO	Atari
PlaNet [Hafner et al., 2018]	RSSM (VAE/RNN)	CEM	MPC	Cheetah
Dreamer [Hafner et al., 2019]	RSSM+CNN	Imagine	Actor-Critic	Control
Plan2Explore [Sekar et al., 2020]	RSSM	Planning	Few-shot	Control

TABLE 5
Overview of Latent Modeling Methods

applied to transfer learning in few-shot learning. In the next subsection more methods are described that also use latent models (see Table 6 and the overview Table 1).

Let us now have a look at the latent model approaches.

An important application in games and robotics is the long range prediction of video images. Building a **generative model for video data** is a challenging problem involving high-dimensional natural-scene data with temporal dynamics, introduced by [Schmidhuber and Huber, 1991]. In many applications next-frame prediction also depends on control or action variables, especially in games. A first paper by [Oh et al., 2015] builds a model to predict Atari games using a high-dimensional video encoding model and action-conditional transformation. The authors describe three-step experiments with a convolutional and with a recurrent (LSTM) encoder. The next step performs action-conditional encoding, after which convolutional decoding takes place. To reduce the effect of small prediction errors compounding through time, a multi-step prediction target is used. Short-term future frames are predicted and fine-tuned to predict longer-term future frames after the previous phase converges, using a curriculum that stabilizes training [Bengio et al., 2009]. Oh et al. perform planning on an abstract, encoded, representation; showing the benefit of acting in latent space. Experimental results on Atari games showed generation of visually-realistic frames useful for control over up to 100-step action-conditional predictions in some games. This architecture was developed further into the VPN approach, which we will describe next.

The **Value Prediction Network** (VPN) approach [Oh et al., 2017] integrates model-free and model-based reinforcement learning into a single abstract neural network that consists of four modules. For training, VPN combines temporal-difference search [Silver et al., 2012] and n-step Q-learning [Mnih et al., 2016]. VPN performs lookahead planning to choose actions. Classical model-based reinforcement learning predicts future observations $T_a(s, s')$. VPN plans future values without having to predict future observations, using abstract representations instead. The VPN network architecture consists of the modules: encoding, transition, outcome, and value. The encoding module is applied to the environment observation to produce a latent state s. The value, outcome, and transition modules work in latent space, and are recursively applied to expand the tree.[1] It does not use MCTS, but a simpler rollout algorithm that performs planning up to a planning horizon. VPN uses imagination to update the policy. It outperforms model-free DQN on Mazes and Atari games such as Seaquest, QBert, Krull, and Crazy Climber. Value Prediction Networks are related to Value Iteration Networks and to the Predictron, which we will describe next.

For robotics and games, video prediction methods are important. Simulated policy learning, or SimPLe, uses **stochastic video prediction** techniques [Kaiser et al., 2019]. SimPLe uses video frame prediction as a basis for model-based reinforcement learning. In contrast to Visual Foresight, SimPLe builds on model-free work on video prediction using variational autoencoders, recurrent world models and generative models [Chiappa et al., 2017, Leibfried et al., 2016, Oh et al., 2015] and model-based work [Azizzadenesheli et al., 2018, Ha and Schmidhuber, 2018b, Oh et al., 2017]. The latent model is formed with a variational autoencoder that is used to deal with the limited horizon of past observation frames [Babaeizadeh et al., 2017, Bengio et al., 2015]. The model-free PPO algorithm [Schulman et al., 2017] is used for policy optimization. In an experimental evaluation, SimPLe is more sample efficient than the Rainbow algorithm [Hessel et al., 2017] on 26 ALE games to learn Atari games with 100,000 sample steps (400k frames).

Learning dynamics models that are accurate enough for planning is a long standing challenge, especially in image-based domains. PlaNet trains a model-based agent to learn the environment dynamics from images and choose actions through planning in latent space with both deterministic and stochastic transition elements. PlaNet is introduced in **Planning from Pixels** [Hafner et al., 2018]. PlaNet uses a Recurrent State Space Model (RSSM) that consists of a transition model, an observation model, a variational encoder and a reward model. Based on these models a Model-Predictive Control agent is used to adapt its plan, replanning each step. For planning, the RSSM is used by the Cross-Entropy-Method (CEM) to search for the best action sequence [Buesing et al., 2018, Doerr et al., 2018, Karl et al., 2016]. In contrast to many model-free reinforcement learning approaches, no explicit policy or value network is used. PlaNet is tested on tasks from MuJoCo and the DeepMind control suite: Swing-up, Reacher, Cheetah, Cup Catch. It reaches performance that is close to strong model-free algorithms.

A year after the PlaNet paper was pubished [Hafner et al., 2019] published Dream to Control: Learning Behaviors by **Latent Imagination**. World models enable interpolating between past experience, and latent models predict both actions and values. The latent models in Dreamer consist of a representation model, an observation model, a transition model, and a reward model. It allows the agent to plan (imagine) the outcomes of potential action sequences without executing them in the environment. It uses an actor-critic approach to learn behaviors that consider rewards beyond the horizon. Dreamer backpropagates through the value model, similar to DDPG [Lillicrap et al., 2015] and Soft-actor-

1. VPN uses a convolutional neural network as the encoding module. The transition module consists of an option-conditional convolution layer (see [Oh et al., 2015]). A residual connection from the previous abstract-state to the next asbtract-state is used [He et al., 2016]. The outcome module is similar to the transition module. The value module consists of two fully-connected layers. The number of layers and hidden units varies depending on the application domain.

critic [Haarnoja et al., 2018]. Dreamer is tested with applications from the DeepMind control suite: 20 visual control tasks such as Cup, Acrobot, Hopper, Walker, Quadruped, on which it achieves good performance.

Finally, Plan2Explore [Sekar et al., 2020] studies how reinforcement learning with latent models can be used for transfer learning, in partiular, few-shot and **zero-shot learning** [Xian et al., 2017]. Plan2Explore is a self-supervised reinforcement learning method that learns a world model of its environment through unsupervised exploration, which it then uses to solve zero-shot and few-shot tasks. Plan2Explore was built on PlaNet [Hafner et al., 2018] and Dreamer [Hafner et al., 2019] learning dynamics models from images, using the same latent models: image encoder (convolutional neural network), dynamics (recurrent state space model), reward predictor, image decoder. With this world model, behaviors must be derived for the learning tasks. The agent first uses planning to explore to learn a world model in a self-supervised manner. After exploration, it receives reward functions to adapt to multiple tasks such as standing, walking, running and flipping. Plan2Explore achieved good zero-shot performance on the DeepMind Control Suite (Swingup, Hopper, Pendulum, Reacher, Cup Catch, Walker) in the sense that the agent's self-supervised zero-shot performance was competitive to Dreamer's supervised reinforcement learning performance.

Conclusion

In the preceding methods we have seen how a single network model can be specialized in three or four separate models. Different rollout methods were proposed, such as local rollouts, and differentiable imagination. Latent, or abstract, models are a direct descendent of model learning networks, with different models for different aspects of the reinforcement learning algorithms. The latent representations have lower capacity, allowing for greater accuracy, better generalization and reduced sample complexity. The smaller latent representation models are often learned unsupervised or self-supervised, using variational autoencoders or recurrent LSTMs. Latent models were applied to transfer learning in few-shot learning.

3.3 End-to-end Learning of Planning and Transitions

In the previous subsection the approach is (1) to learn a transition model through backpropagation and then (2) to do conventional lookahead rollouts using a planning algorithm such as value iteration, depth-limited search, or MCTS. A larger trend in machine learning is to replace conventional algorithms by differentiable or gradient style approaches, that are self-learning and self-adapting. Would it be possible to make the conventional rollouts differentiable as well? If updates can be made differentiable, why not planning?

The final approach of this survey is indeed to learn both the transition model and planning steps end-to-end. This means that the neural network represents both the transition model and executes the planning steps with it. This is a challenge that has to do with a single neural network, but we will see that abstract models, with latent representations, can more easily be used to achieve the execution of planning steps.

When we look at the action that a neural network normally performs as a transformation and filter activity (selection, or classification) then it is easy to see than planning, which consists of state unrolling and selection, is not so far from what a neural network is normally used for. Note that especially recurrent neural networks and LSTM contain implicit state, making their use as a planner even easier.

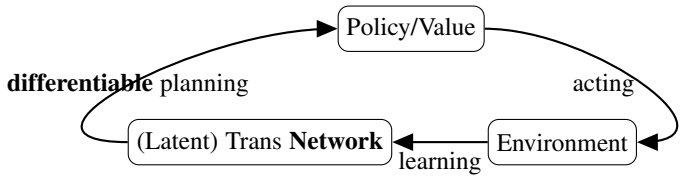

Fig. 10. End-to-End Planning/Transitions

Some progress has been made with this idea. One approach is to map the planning iterations onto the layers of a deep neural network, with each layer representing a lookahead step. The transition *model* becomes embedded in a transition *network*, see Figure 10.

In this way, the planner becomes part of one large trained end-to-end agent. (In the figure the full circle is made bold to signal end-to-end learning.) Table 6 summarizes the approaches of this subsection. We will see how the iterations of value iteration can be implemented in the layers of a convolutional neural network (CNN). Next, two variations of this method are presented, and a way to implement planning with convolutional LSTM modules. All these approaches implement differentiable, trainable, planning algorithms, that can generalize to different inputs. The later methods use elaborate schemes with latent models so that the learning can be applied to different application domains.

Value Iteration Networks (VIN) are introduced by Tamar et al. in [Tamar et al., 2016], see also [Niu et al., 2018]. A VIN is a differentiable multi-layer network that is used to perform the steps of a simple planning algorithm. The core idea it that value iteration (VI, see Algorithm 1) or step-by-step planning can be implemented by a multi-layer convolutional network: each layer does a step of lookahead. The VI iterations are rolled-out in the network layers Q with A channels. Through backpropagation the model learns the value iteration parameters. The aim is to learn a general model, that can navigate in unseen environments. VIN learns a fully differentiable planning algorithm. The idea of planning by gradient descent exists for some time, several authors explored learning approximations of dynamics in neural networks [Ilin et al., 2007, Kelley, 1960, Schmidhuber, 1990a]. VIN can be used for discrete and continuous path planning, and has been tried in grid world problems and natural language tasks. VIN has achieved generalization of finding shortest paths in unseen mazes.

However, a limitation of VIN is that the number of layers of the CNN restricts the number of planning steps, restricting VINs to small and low-dimensional domains. Schleich et al. [Schleich et al., 2019] extend VINs by adding abstraction, and Srinivas et al. [Srinivas et al., 2018] introduce universal planning networks, UPN, which generalize to modified robot morphologies. VProp, or **Value Propagation** [Nardelli et al., 2018] is another attempt at creating generalizable planners inspired by VIN. By using a hierarchical structure VProp has the ability to generalize to larger map sizes and dynamic environments. VProp not only learns to plan and navigate in dynamic environments, but their hierarchical structure provides a way to generalize to navigation tasks where the required planning horizon and the size of the map are much larger than the ones seen at training time. VProp is evaluated

Approach	*Learning*	*Planning*	*Reinforcement Learning*	*Application*
VIN [Tamar et al., 2016]	CNN	Rollout in network	Value Iteration	Mazes
VProp [Nardelli et al., 2018]	CNN	Hierarch Rollouts	Value Iteration	Navigation
TreeQN [Farquhar et al., 2018]	Tree-shape Net	Plan-functions	DQN/Actor-Critic	Box pushing
ConvLSTM [Guez et al., 2019]	CNN+LSTM	Rollouts in network	A3C	Sokoban
I2A [Weber et al., 2017]	CNN/LSTM encoder	Meta-controller	A3C	Sokoban
Predictron [Silver et al., 2017b]	k, γ, λ-CNN-predictr	k-rollout	λ-accum	Mazes
World Model [Ha and Schmidhuber, 2018b]	VAE	CMA-ES	MDN-RNN	Car Racing
MuZero [Schrittwieser et al., 2019]	Latent	MCTS	Curriculum	Go/chess/shogi+Atari

TABLE 6
Overview of End-to-End Planning/Transition Methods

on grid-worlds and also on dynamic environments and on a navigation scenario from StarCraft.

A different approach is taken in TreeQN/ATreeC. Again, the aim is to create **differentiable tree planning** for deep reinforcement learning [Farquhar et al., 2018]. As VIN, TreeQN is focused on combining planning and deep reinforcement learning. Unlike VIN, however, TreeQN does so by incorporating a recursive tree structure in the network. It models an MDP by incorporating an explicit encoder function, a transition function, a reward function, a value function, and a backup function (see also latent models in the next subsection). In this way, it aims to achieve the same goal as VIN, that is, to create a differentiable neural network architecture that is suitable for planning. TreeQN is based on DQN-value-functions, an actor-critic variant is proposed as ATreeC. TreeQN is a prelude to latent models methods in the next subsection. In addition to being related to VIN, this approach is also related to VPN [Oh et al., 2017] and the Predictron [Silver et al., 2017b]. TreeQN is tried on box pushing applications, like Sokoban, and nine Atari games.

Another approach to differentiable planning is to teach a sequence of convolutional neural networks to exhibit planning behavior. A paper by [Guez et al., 2019] takes this approach. The paper demonstrates that a neural network architecture consisting of modules of a **convolutional network and LSTM** can learn to exhibit the behavior of a planner. In this approach the planning occurs implicitly, by the network, which the authors call model-free planning, in contrast to the previous approaches in which the network structure more explicitly resembles a planner [Farquhar et al., 2018, Guez et al., 2018, Tamar et al., 2016]. In this method model-based behavior is learned with a general recurrent architecture consisting of LSTMs and a convolutional network [Schmidhuber, 1990b] in the form of a stack of ConvLSTM modules [Xingjian et al., 2015]. For the learning of the ConvLSTM modules the A3C actor-critic approach is used [Mnih et al., 2016]. The method is tried on Sokoban and Boxworld [Zambaldi et al., 2018]. A stack of depth D, repreated N times (time-ticks) allows the network to plan. In harder Sokoban instances, larger capacity networks with larger depth performed better. The experiments used a large number of environment steps, future work should investigate how to achieve sample-efficiency with this architecture.

Conclusion

Planning networks combine planning and transition learning. They fold the planning into the network, making the planning process itself differentiable. The network then learns which planning decisions to make. Value Iteration Networks have shown how learning can transfer to mazes that have not been seen before. A drawback is that due to the marriage of problem size and network topology the approach has been limited to smaller sizes, something that subsequent methods have tried to reduce. One of these approaches is TreeQN, which uses multiple smaller models and a tree-structured network. The related Predictron architecture [Silver et al., 2017b] also learns planning end-to-end, and is applicable to different kinds and sizes of problems.

The Predictron uses abstract models, and will be discussed in the next subsection.

3.3.1 End-to-End Planning/Transitions with Latent Models

We will now discuss latent model approaches in end-to-end learning of planning and transitions.

The first abstract imagination-based approach that we discuss is **Imagination-Augmented Agent**, or I2A, by [Buesing et al., 2018, Pascanu et al., 2017, Weber et al., 2017]. A problem of model-based algorithms is the sensitivity of the planning to model imperfections. I2A deals with these imperfections by introducing a latent model, that learns to interpret internal simulations and adapt a strategy to the current state. I2A uses latent models of the environment, based on [Buesing et al., 2018, Chiappa et al., 2017]. The core architectural feature of I2A is an environment model, a recurrent architecture trained unsupervised from agent trajectories. I2A has four elements that together constitute the abstract model: (1) It has a manager that constructs a plan, which can be implemented with a CNN. (2) It has a controller that creates an action policy. (3) It has an environment model to do imagination. (4) Finally, it has a memory, which can be implemented with an LSTM [Pascanu et al., 2017]. I2A uses a manager or meta-controller to choose between rolling out actions in the environment or by imagination (see [Hamrick et al., 2017]). This allows the use of models which only coarsely capture the environmental dynamics, even when those dynamics are not perfect. The I2A network uses a recurrent architecture in which a CNN is trained from agent trajectories with A3C [Mnih et al., 2016]. I2A achieves success with little data and imperfect models, optimizing point-estimates of the expected Q-values of the actions in a discrete action space. I2A is applied to Sokoban and Mini-Pacman by [Buesing et al., 2018, Weber et al., 2017]. Performance is compared favorably to model-free and planning algorithms (MCTS). Pascanu et al. apply the approach on a maze and a spaceship task [Pascanu et al., 2017].

Planning networks (VIN) combine planning and learning end-to-end. A limitation of VIN is that the tight connection between problem domain, iteration algorithm, and network architecture limited the applicability to small grid world problem. The **Predictron** introduces an abstract model to remove this limitation. The Predictron was introduced by Silver et al. and combines end-

to-end planning and model learning [Silver et al., 2017b]. As with [Oh et al., 2017], the model is an abstract model that consists of four components: a representation model, a next-state model, a reward model, and a discount model. All models are differentiable. The goal of the abstract model in Predictron is to facilitate value prediction (not state prediction) or prediction of pseudo-reward functions that can encode special events, such as "staying alive" or "reaching the next room." The planning part rolls forward its internal model k steps. As in the Dyna architecture, imagined forward steps can be combined with samples from the actual environment, combining model-free and model-based updates. The Predictron has been applied to procedurally generated mazes and a simulated pool domain. In both cases it out-performed model-free algorithms.

Latent models of the dynamics of the environment can also be viewed as **World Models**, a term used by [Ha and Schmidhuber, 2018a,b]. World Models are inspired by the manner in which humans are thought to contruct a mental model of the world in which we live. World Models are generative recurrent neural networks that are trained unsupervised to generate states for simulation using a variational autoencoder and a recurrent network. They learn a compressed spatial and temporal representation of the environment. By using features extracted from the World Model as inputs to the agent, a compact and simple policy can be trained to solve a task, and planning occurs in the compressed or simplified world. For a visual environment, World Models consist of a vision model, a memory model, and a controller. The vision model is often trained unsupervised with a variational autoencoder. The memory model is approximated with a mixture density network of a Gaussian distribution (MDN-RNN) [Bishop, 1994, Graves, 2013]. The controller model is a linear model that maps directly to actions. It uses the CMA-ES Evolutionary Strategy for optimizing Controller models. Rollouts in World Models are also called *dreams*, to contrast them with samples from the real environment. With World Models a policy can in principle even be trained completely inside the dream, using imagination only inside the World Model, to test it out later in the actual environment. World Models have been applied experimentally to VizDoom tasks such as Car Racing [Kempka et al., 2016].

Taking the development of AlphaZero further is the work on **MuZero** [Schrittwieser et al., 2019]. Board games are well suited for model-based methods because the transition function is given by the rules of the game. However, would it be possible to do well if the rules of the game were *not* given? In MuZero a new architecture is used to learn transition functions for a range of different games, from Atari to board games. MuZero learns the transition model for all games from interaction with the environment, with one architecture, that is able to learn different transition models. As with the Predictron [Silver et al., 2017b] and Value Prediction Networks [Oh et al., 2017], MuZero has an abstract model with different modules: representation, dynamics, and prediction function. The dynamics function is a recurrent process that computes transition latent state) and reward. The prediction function computes policy and value functions. For planning, MuZero uses a version of MCTS, without the rollouts, and with P-UCT as selection rule, using information from the abstract model as input for node selection. MuZero can be regarded as joining the Predictron with self-play. It performs well on Atari games and on board games, learning to play the games from scratch, after having learned the rules of the games from scratch from the environment.

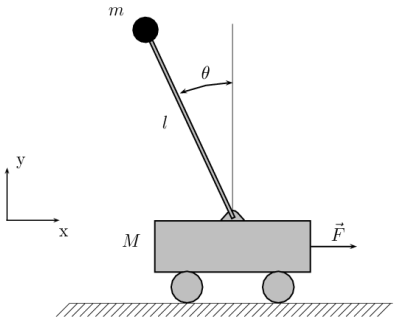

Fig. 11. Cartpole Pendulum [Sutton and Barto, 2018]

Conclusion

Latent models represent states with a number of smaller, latent models, allowing planning to happen in a smaller latent space. They are useful for explicit planning and with end-to-end learnable planning, as in the Predictron [Silver et al., 2017b]. Latent models allow end-to-end planning to be applied to a broader range of applications, beyond small mazes. The Predictron creates an abstract planning network, in the spirit of, but without the limitations of, Value Iteration Networks [Tamar et al., 2016]. The World Models interpretation links latent models to the way in which humans create mental models of the world that we live in.

After this detailed survey of model-based methods—the agents—it is time to discuss our findings and draw conclusions. Before we do so, we will first look at one of the most important elements for reproducible reinforcement learning research, the benchmark.

4 BENCHMARKS

Benchmarks—the environments—play a key role in artificial intelligence. Without them, progress cannot be measured, and results cannot be compared in a meaningful way. The benchmarks define the kind of intelligence that our artificial methods should approach. For reinforcement learning, Mountain car and Cartpole are well-known small problems that characterize the kind of problem to solve (see Figure 11). Chess has been called the Drosophila of AI [Landis and Yaglom, 2001]. In addition to Mountain car and chess a series of benchmark applications have been used to measure progress of artificially intelligent methods. Some of the benchmarks are well-known and have been driving progress. In image recognition, the ImageNet sequence of competitions has stimulated great progress [Fei-Fei et al., 2009, Guo et al., 2016, Krizhevsky et al., 2012]. The current focus on reproducibility in reinforcement learning emphasizes the importance of benchmarks [Henderson et al., 2017, Islam et al., 2017, Khetarpal et al., 2018].

Most papers that introduce new model-based reinforcement learning algorithms perform some form of experimental evaluation of the algorithm. Still, since papers use different versions and hyper-parameter settings, comparing algorithm performance remains difficult in practice. A recent benchmarking study compared the performance of 14 algorithms, and some baseline algorithms on a number of MuJoCo [Todorov et al., 2012] robotics benchmarks [Wang et al., 2019b]. There was no clear winner. Performance of methods varied widely from application to application.

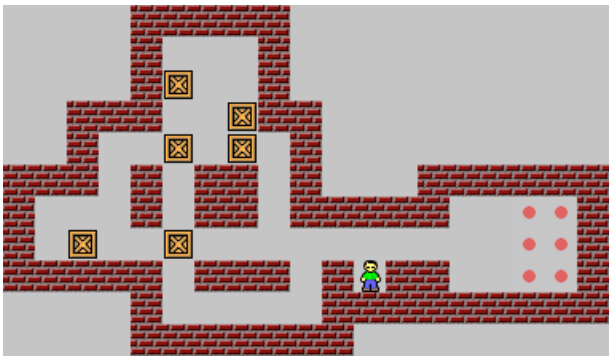

Fig. 12. Sokoban Puzzle [Chao, 2013]

There is much room for further improvement on many applications of model-based reinforcement learning algorithms, and for making methods more robust. The use of benchmarks should become more standardized to ease meaningful performance comparisons.

We will now describe benchmarks commonly used in deep model-based reinforcement learning. We will discuss five sets of benchmarks: (1) puzzles and mazes, (2) Atari arcade games such as Pac-Man, (3) board games such as Go and chess, (4) real-time strategy games such as StarCraft, and (5) simulated robotics tasks such as Half-Cheetah. As an aside, some of these benchmarks resemble challenges that children and adults use to play and learn new skills.

We will have a closer look at these sets of benchmarks, some with discrete, some with continuous action spaces.

4.1 Mazes

Trajectory planning algorithms are crucial in robotics [Gasparetto et al., 2015, Latombe, 2012]. There is a long tradition of using 2D and 3D path-finding problems in reinforcement learning and AI. The Taxi domain was introduced by [Dietterich, 2000] in the context of hierarchical problem solving, and box-pushing problems such as Sokoban have been used frequently [Dor and Zwick, 1999, Junghanns and Schaeffer, 2001, Murase et al., 1996, Zhou and Dovier, 2013], see Figure 12. The action space of these puzzles and mazes is discrete. The related problems are typically NP-hard or PSPACE-hard [Culberson, 1997, Hearn and Demaine, 2009] and solving them requires basic path and motion planning skills.

Small versions of the mazes can be solved exactly by planning, larger instances are only suitable for approximate planning or learning methods.

Mazes can be used to test algorithms for "flat" reinforcement learning path finding problems [Nardelli et al., 2018, Silver et al., 2017b, Tamar et al., 2016]. Grids and box-pushing games such as Sokoban can also feature rooms or subgoals, that may then be used to test algorithms for hierarchically structured problems [Farquhar et al., 2018, Feng et al., 2020, Guez et al., 2019, Weber et al., 2017].

The problems can be made more difficult by enlarging the grid and by inserting more obstacles. Mazes and Sokoban grids are sometimes procedurally generated [Hendrikx et al., 2013, Shaker et al., 2016, Togelius et al., 2013]. The goal for the algorithms is typically to find a solution for a grid of a certain difficulty class, to find a shortest solution, or to learn to solve a class of grids by training on a different class of grids, to test transfer learning.

4.2 Board Games

Another classic group of benchmarks for planning and learning algorithms is board games.

Two-person zero-sum perfect information board games such as tic tac toe, chess, checkers, Go, and shogi have been used since the 1950s as benchmarks in AI. The action space of these games is discrete. Notable achievements were in checkers, chess, and Go, where human world champions were defeated in 1994, 1997, and 2016, respectively [Campbell et al., 2002, Schaeffer et al., 1996, Silver et al., 2016]. Other games are used as well as benchmarks, such as Poker [Brown and Sandholm, 2018] and Diplomacy [Anthony et al., 2020].

The board games are typically used "as is" and are not changed for different experiments (as with mazes). They are fixed benchmarks, challenging and inspirational games where the goal is often beating human world champions. In model-based deep reinforcement learning they are used for self-play methods [Anthony et al., 2017, Schrittwieser et al., 2019, Tesauro, 1995a].

Board games have been traditional mainstays of artificial intelligence, mostly associated with the symbolic reasoning approach to AI. In contrast, the next benchmark is associated with connectionist AI.

4.3 Atari

Shortly after 2010 the Atari Learning Environment (ALE) [Bellemare et al., 2013] was introduced for the sole purpose of evaluating reinforcement learning algorithms on high-dimensional input, to see if end-to-end pixel-to-joystick learning would be possible. ALE has been used widely in the field of reinforcement learning, after impressive results such as [Mnih et al., 2015]. ALE runs on top of an emulator for the classic 1980s Atari gaming console, and features more than 50 arcade games such as Pong, Breakout, Space Invaders, Pac-Man, and Montezuma's Revenge. ALE is well suited for benchmarking perception and eye-hand-coordination type skills, less so for planning. ALE is mostly used for deep reinforcement learning algorithms in sensing and recognition tasks.

ALE is a popular benchmark that has been used in many model-based papers [Ha and Schmidhuber, 2018b, Kaiser et al., 2019, Oh et al., 2015, 2017, Schrittwieser et al., 2019], and many model-free reinforcement learning methods. As with mazes, the action space is discrete and low-dimensional—9 joystick directions and a push-button—although the input space is high-dimensional.

The ALE games are quite varied in nature. There are "easy" eye-hand-coordination tasks such as Pong and Breakout, and there are more strategic level games where long periods of movement exist without changes in score, such as Pitfall and Montezuma's Revenge. The goal of ALE experiments is typically to achieve a score level comparable to humans in as many of the 57 games as possible (which has recently been achieved [Badia et al., 2020]). After this achievement, some researchers believe that the field is ready for more challenging benchmarks [Machado et al., 2018].

4.4 Real-Time Strategy and Video Games

The Atari benchmarks are based on simple arcade games of 35–40 years ago, most of which are mostly challenging for eye-hand-coordination skills. Real-time strategy (RTS) games and games such as StarCraft, DOTA, and Capture the Flag provide more challenging tasks. The strategy space is large; the state space of

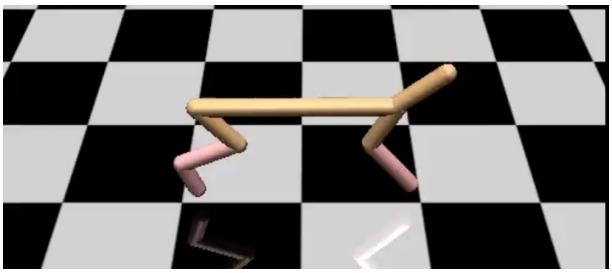

Fig. 13. Half-Cheetah

StarCraft has been estimated at 10^{1685}, much larger than chess (10^{47}) or go (10^{147}). Most RTS games are multi-player, non-zero-sum, imperfect information games that also feature high-dimensional pixel input, reasoning, team collaboration, as well as eye-hand-coordination. The action space is stochastic and is a mix of discrete and continuous actions. A very high degree of diversity is necessary to prevent specialization.

Despite the challenging nature, impressive achievements have been reported recently in all three mentioned games where human performance was matched or even exceeded [Berner et al., 2019, Jaderberg et al., 2019, Vinyals et al., 2019]. In these efforts deep model-based reinforcement learning is combined with multi-agent and population based methods. These mixed approaches achieve impressive results on RTS games; added diversity diminishes the specialization trap of two-agent approaches [Vinyals et al., 2019], their approaches may combine aspects of self-play and latent models, although often the well-tuned combination of a number of methods is credited with the high achievements in these games. The mixed approaches are not listed separately in the taxonomy of the next section.

4.5 Robotics

Reinforcement learning is a paradigm that is well-suited for modeling planning and control problems in robotics. Instead of minutely programming high-dimensional robot-movements step-by-step, reinforcement learning is used to train behavior more flexibly, and possibly end-to-end from camera input to arm manipulation.

Training with real-world robots is expensive and complicated. In robotics sample efficiency is of great importance because of the cost of interaction with real-world environments and the wear of physical robot arms. For this reason virtual environments have been devised. Todorov et al. [Todorov et al., 2012] introduced MuJoCo, a software suite for simulated robot behavior. It is used extensively in reinforcement learning research. Well-known benchmark tasks are Reacher, Swimmer, Half-Cheetah, and Ant, in which the agent's task is to teach itself the appropriate movement actions, see Figure 13 for an example. Many model-based deep reinforcement learning methods are tested on MuJoCo [Azizzadenesheli et al., 2018, Chua et al., 2018, Clavera et al., 2018, Feinberg et al., 2018, Gu et al., 2016, Hafner et al., 2018, Heess et al., 2015, Janner et al., 2019] and other robotics tasks [Finn and Levine, 2017, Hafner et al., 2019, Levine and Abbeel, 2014, Sekar et al., 2020, Tassa et al., 2012]. The action space of these tasks is continuous, and the emphasis in experiments is on sample efficiency.

Conclusion

No discussion on empirical deep reinforcement learning is complete without the mention of OpenAI Gym [Brockman et al., 2016]. Gym is a toolkit for developing and comparing reinforcement learning algorithms and provides a training environment for reinforcement learning agents. Gym includes interfaces to benchmark sets such as ALE and MuJoCo. Other software suites are [Bellemare et al., 2013, Tassa et al., 2018, Todorov et al., 2012, Vinyals et al., 2017, Yu et al., 2020].

Baseline implementations of many deep reinforcement learning agent algorithms can also be found at the Gym website https://gym.openai.com.

Research into suitable benchmarks is active. Further interesting approaches are Procedural Content Generation [Togelius et al., 2013], MuJoCo Soccer [Liu et al., 2019], and the Obstacle Tower Challenge [Juliani et al., 2019].

Now that we have seen the benchmarks on which our model-based methods are tested, it is time for an in-depth discussion and outlook for future work.

5 Discussion and Outlook

Model-based reinforcement learning promises lower sample complexity. Sutton's work on imagination, where a model is created with environment samples that are then used to create extra imagined samples for free, clearly suggests this aspect of model-based reinforcement learning. The transition model acts as a multiplier on the amount of information that is used from each environment sample.

Another, and perhaps more important aspect, is generalization performance. Model-based reinforcement learning builds a dynamics model of the domain. This model can be used multiple times, for new problem instances, but also for new problem classes. By learning the transition and reward model, model-based reinforcement learning may be better at capturing the essence of a domain than model-free methods. Model-based reinforcement learning may thus be better suited for solving transfer learning problems, and for solving long sequential decision making problems, a class of problems that is important in real world decision making.

Classical table based approaches and Gaussian Process approaches have been quite succesful in achieving low sample complexity for problems of moderate complexity [Deisenroth et al., 2013b, Kober et al., 2013, Sutton and Barto, 2018]. However, the topic of the current survey is *deep* models, for high dimensional problems with complex, non-linear, and discontinuous functions. These application domains pose a problem for classical model-based approaches. Since high-capacity deep neural networks require many samples to achieve high generalization (without overfitting), a challenge in deep model-based reinforcement learning is to maintain low sample complexity.

We have seen a wide range of approaches that attempt this goal. Let us now discuss and summarize the benchmarks, the approaches for deep models, and possible future work.

5.1 Benchmarking

Benchmarks are the lifeblood of AI. We must test our algorithms to know if they exhibit intelligent behavior. Many of the benchmarks allow difficult decision making situations to be modeled. Two-person games allow modeling of competition. In

real world decision making, collaboration and negotiation are also important. Real-time strategy games allow collaboration, competition and negotiation to be modelled, and multi-agent and hierarchical algorithms are being developed for these decision making situations [Jaderberg et al., 2019, Kulkarni et al., 2016, Makar et al., 2001].

Unfortunately, the wealth of choice in benchmarks makes it difficult to compare results that are reported in the literature. Not all authors publish their code. We typically need to rerun experiments with identical benchmarks to compare algorithms conclusively. Outcomes often differ from the original works, also because not all hyperparameter settings are always clear and implementation details may differ [Henderson et al., 2017, Islam et al., 2017, Wang et al., 2019b]. Authors should publish their code if our field wants to make progress. The recent attention for reproducibility in deep reinforcement learning is helping the field move in the right direction. Many papers now publish their code and the hyperparameter settings that were used in the reported experiments.

5.2 Curriculum Learning and Latent Models

Model-based reinforcement learning works well when the transition and reward models are **given** by the rules of the problem. We have seen how perfect models in games such as chess and Go allow deep and accurate planning. Systems were constructed [Anthony et al., 2017, Silver et al., 2018, Tesauro, 1995a] where curriculum learning facilitated tabula rasa self-learning of highly complex games of strategy; see also [Bengio et al., 2009, Narvekar et al., 2020, Plaat, 2020]. The success of self-play has led to interest in curriculum learning in single-agent problems [Doan et al., 2019, Duan et al., 2016, Feng et al., 2020, Laterre et al., 2018].

When the rules are not given, they might be **learned**, to create a transition model. Unfortunately, the planning accuracy of learned models is less than perfect. We have seen efforts with Gaussian Processes and ensembles to improve model quality, and efforts with local planning and Model-Predictive Control, to limit the damage of imperfections in the transition model. We have also discussed, at length, latent, abstract, models, where for each of the functions of the Markov Decision Process a separate sub-module is learned. Latent models achieve better accuracy with explicit planning, as planning occurs in latent space [Hafner et al., 2019, Kaiser et al., 2019, Oh et al., 2017].

The work on Value Iteration Networks [Tamar et al., 2016] inspired **end-to-end** learning, where both the transition model and the planning algorithm are learned, end-to-end. Combined with latent models (or World Models [Ha and Schmidhuber, 2018b]) impressive results where achieved [Silver et al., 2017b], and model/planning accuracy was improved to the extent that tabula rasa curriculum self-learning was achieved, in Muzero [Schrittwieser et al., 2019] for both chess, Go, and Atari games. End-to-end learning and latent models together allowed the circle to be closed, achieving curriculum learning self-play also for problems where the rules are not given. See Figure 14 for relations between the different approaches of this survey. The main categories are color-coded, latent methods are dashed.

Optimizing directly in a latent space has been successful with a generative adversarial network [Volz et al., 2018]. World Models are linked to neuroevolution by [Risi and Stanley, 2019]. For future work, the combination of curriculum learning, ensembles,

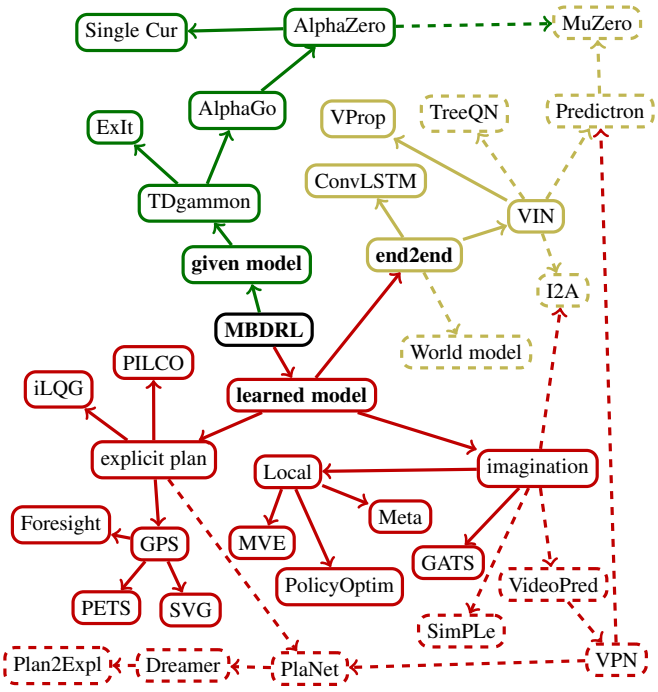

Fig. 14. Influence of Model-Based Deep Reinforcement Learning Approaches. Green: given transitions/explicit planning; Red: learned transitions/explicit planning; Yellow: end-to-end transitions and planning. Dashed: latent models.

and latent models appears quite fruitful. Self-play has been used to achieve further success in other challenging games, such as StarCraft [Vinyals et al., 2019], DOTA [Berner et al., 2019], Capture the Flag [Jaderberg et al., 2019], and Poker [Brown and Sandholm, 2019]. In multi-agent real-time strategy games the aspect of collaboration and teams is important, and self-play model-based reinforcement learning has been combined with multi-agent, hierarchical, and population based methods.

In future work, more research is needed to explore the potential of (end-to-end) planning with latent representational models more fully for larger problems, for transfer learning, and for cooperative problems. More research is needed in uncertainty-aware neural networks. For such challenging problems as real-time strategy and other video games, more combinations of deep model-based reinforcement learning with multi-agent and population based methods can be expected [Bäck, 1996, Risi and Preuss, 2020, Vinyals et al., 2019].

In end-to-end planning and learning, latent models introduce differentiable versions of more and more classical algorithms. For example, World Models [Ha and Schmidhuber, 2018b] have a trio of Vision, Memory, Control models, reminding us of the Model, View, Controller design pattern [Gamma et al., 2009], or even of classical computer architecture elements such as ALU, RAM, and Control Unit [Hennessy and Patterson, 2017]. Future work will show if differentiable algorithms will be found for even more classical algorithms.

6 Conclusion

Deep learning has revolutionized reinforcement learning. The new methods allow us to approach more complicated problems than before. Control and decision making tasks involving high dimensional visual input come within reach.

In principle, model-based methods offer the advantage of lower sample complexity over model-free methods, because of their transition model. However, traditional methods, such as Gaussian Processes, that work well on moderately complex problems with few samples, do not perform well on high-dimensional problems. High-capacity models have high sample complexity, and finding methods that generalize well with low sample complexity has been difficult.

In the last five years many new methods have been devised, and great success has been achieved in model-free and in model-based deep reinforcement learning. This survey has summarized the main ideas of recent papers in three approaches. For more and more aspects of model-based reinforcement learning algorithms differentiable methods appear. Latent models condense complex problems into compact latent representations that are easier to learn. End-to-end curriculum learning of latent planning and learning models has been achieved.

In the discussion we mentioned open problems for each of the approaches, where we expect worthwhile future work will occur, such as curriculum learning, uncertainty modeling and transfer learning by latent models. Benchmarks are important to test progress, and more benchmarks of latent models can be expected. Curriculum learning has shown how complex problems can be learned relatively quickly from scratch, and latent models allow planning in efficient latent spaces. Impressive results have been reported; future work can be expected in transfer learning with latent models, and the interplay of curriculum learning with (generative) latent models, in combination with end-to-end learning of larger problems.

Benchmarks in the field have also had to keep up. Benchmarks have progressed from single-agent grid worlds to multi-agent Real-time strategy games and complicated camera-arm robotics manipulation tasks. Reproducibility and benchmarking studies are of great importance for real progress. In real-time strategy games model-based methods are being combined with multi-agent, hierarchical and evolutionary approaches, allowing the study of collaboration, competation and negotiation.

Model-based deep reinforcement learning is a vibrant field of AI with a long history before deep learning. The field is blessed with a high degree of activity, an open culture, clear benchmarks, shared code-bases [Brockman et al., 2016, Tassa et al., 2018, Vinyals et al., 2017] and a quick turnaround of ideas. We hope that this survey will lower the barrier of entry even further.

Acknowledgments

We thank the members of the Leiden Reinforcement Learning Group, and especially Thomas Moerland and Mike Huisman, for many discussions and insights.

References

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*, pages 1–8, 2007.

Ethem Alpaydin. *Introduction to machine learning, Third edition*. MIT Press, 2020.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5360–5370, 2017.

Thomas Anthony, Robert Nishihara, Philipp Moritz, Tim Salimans, and John Schulman. Policy gradient search: Online planning and expert iteration without search trees. *arXiv preprint arXiv:1904.03646*, 2019.

Thomas Anthony, Tom Eccles, Andrea Tacchetti, János Kramár, Ian Gemp, Thomas C Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, Richard Everett, et al. Learning to play no-press diplomacy with best response policy iteration. *arXiv preprint arXiv:2006.04635*, 2020.

Broderick Arneson, Ryan B Hayward, and Philip Henderson. Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010.

Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary C Lipton, and Animashree Anandkumar. Surprising negative results for generative adversarial tree search. *arXiv preprint arXiv:1806.05780*, 2018.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

Thomas Bäck. *Evolutionary algorithms in theory and practice: evolutionary strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. *arXiv preprint arXiv:2003.13350*, 2020.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Richard Bellman. *Dynamic programming*. Courier Corporation, 1957, 2013.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48, 2009.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Christopher M Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994.

Christopher M Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer Verlag, Heidelberg, 2006.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Lars Buesing, Theophane Weber, Sébastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

Sinan Çalışır and Meltem Kurt Pehlivanoğlu. Model-free reinforcement learning algorithms: A survey. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2019.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

Yang Chao. Sokoban.org, 2013.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*, 2018.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.

Joseph Culberson. Sokoban is PSPACE-complete. Technical report, University of Alberta, 1997.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 465–472, 2011.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2013a.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. In *rFoundations and Trends in Robotics 2*, pages 1–142. Now publishers, 2013b.

Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Thang Doan, Joao Monteiro, Isabela Albuquerque, Bogdan Mazoure, Audrey Durand, Joelle Pineau, and R Devon Hjelm. On-line adaptative curriculum learning for gans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3470–3477, 2019.

Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. Probabilistic recurrent state-space models. *arXiv preprint arXiv:1801.10395*, 2018.

Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and SA Whiteson. TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning. International Conference on Learning Representations, 2018.

Li Fei-Fei, Jia Deng, and Kai Li. Imagenet: Constructing a large-scale image database. *Journal of Vision*, 9(8):1037–1037, 2009.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Dieqiao Feng, Carla P Gomes, and Bart Selman. Solving hard AI planning instances using curriculum-driven deep reinforcement learning. *arXiv preprint arXiv:2006.02689*, 2020.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, 2017.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of reusable object-oriented sofware*. Addison Wesley, 2009.

Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and trajectory planning algorithms: A general overview. In *Motion and Operation Planning of Robotic Systems*, pages 3–27. Springer, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, Cambridge, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

Arthur Guez, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. Learning to search with MCTSnets. *arXiv preprint arXiv:1802.04697*, 2018.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. *arXiv preprint arXiv:1901.03559*, 2019.

Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018a.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018b.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

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

Jessica B Hamrick, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia. Metacontrol for adaptive imagination-based optimization. *arXiv preprint arXiv:1705.02670*, 2017.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Ryan B Hayward and Bjarne Toft. *Hex: The Full Story*. CRC Press, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. CRC Press, 2009.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9 (1):1–22, 2013.

John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

Jonathan Hui. Model-based reinforcement learning https://medium.com/@jonathan_hui/rl-model-based-reinforcement-learning-3c2b6f0aa323. Medium post, 2018.

Mike Huisman, Jan van Rijn, and Aske Plaat. Metalearning for deep neural networks. In Pavel Brazdil, editor, *Metalearning: Applications to data mining*. Springer, 2020.

Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason. Computer shogi. *Artificial Intelligence*, 134(1-2):121–144, 2002.

Roman Ilin, Robert Kozma, and Paul J Werbos. Efficient learning in cellular simultaneous recurrent neural networks—the case of maze navigation problem. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 324–329, 2007.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1125–1134, 2017.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019.

Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1-2):219–251, 2001.

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 12(1): 1–20, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4: 237–285, 1996.

Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, 2011.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374*, 2019.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206, 2017.

Sanket Kamthe and Marc Peter Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. *arXiv preprint arXiv:1706.06491*, 2017.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick Van der Smagt. Deep variational Bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Henry J Kelley. Gradient theory of optimal flight paths. *American Rocket Society Journal*, 30(10):947–954, 1960.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. VizDoom: A Doom-based AI research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games*, pages 1–8, 2016.

Khimya Khetarpal, Zafarali Ahmed, Andre Cianflone, Riashat Islam, and Joelle Pineau. Re-evaluate: Reproducibility in evaluating reinforcement learning algorithms. In *Reproducibility in Machine Learning Workshop, ICML*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

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

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer, 2006.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014, 2000.

Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

Evgenii Mikhailovich Landis and Isaak Moiseevich Yaglom. About Aleksandr Semenovich Kronrod. *Russian Mathematical Surveys*, 56(5):993–1007, 2001.

Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Torbjorn S Dahl, Amine Kerkeni, and Karim Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672*, 2018.

Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Felix Leibfried, Nate Kushman, and Katja Hofmann. A deep learning approach for joint video frame and reward prediction in Atari games. *arXiv preprint arXiv:1611.07078*, 2016.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. Emergent coordination through competition. *arXiv preprint arXiv:1902.07151*, 2019.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253. ACM, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. A0c: Alpha zero in continuous action space. *arXiv preprint arXiv:1805.09613*, 2018.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. A framework for reinforcement learning and planning. *arXiv preprint arXiv:2006.15009*, 2020a.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020b.

Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. The second type of uncertainty in Monte Carlo Tree Search. *arXiv preprint arXiv:2005.09645*, 2020c.

William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. Automatic making of Sokoban problems. In *Pacific Rim International Conference on Artificial Intelligence*, pages 592–600. Springer, 1996.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018.

Nantas Nardelli, Gabriel Synnaeve, Zeming Lin, Pushmeet Kohli, Philip HS Torr, and Nicolas Usunier. Value propagation networks. *arXiv preprint arXiv:1805.11199*, 2018.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C Smith, and Jelena Kovačević. Generalized value iteration networks: Life beyond lattices. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128, 2017.

Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

Aske Plaat. *Learning to Play: Reinforcement Learning and Games*. Springer Verlag, Heidelberg, 2020.

Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie De Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255–293, 1996.

Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

Sebastian Risi and Mike Preuss. From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI. *KI-Künstliche Intelligenz*, pages 1–11, 2020.

Sebastian Risi and Kenneth O Stanley. Deep neuroevolution of recurrent and discrete world models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 456–462, 2019.

Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, Malaysia, 2016.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. Chinook, the world man-machine checkers champion. *AI Magazine*, 17(1):21, 1996.

Daniel Schleich, Tobias Klamt, and Sven Behnke. Value iteration networks on multiple levels of abstraction. *arXiv preprint arXiv:1905.11068*, 2019.

Juergen Schmidhuber and Rudolf Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2 (01–02):125–134, 1991.

Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 253–258. IEEE, 1990a.

Jürgen Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. 1990b.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. *arXiv preprint arXiv:2005.05960*, 2020.

Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016.

Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI Spring Symposium Series*, 2013.

David Silver, Richard S Sutton, and Martin Müller. Temporal-difference search in computer Go. *Machine Learning*, 87(2):183–219, 2012.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 (7587):484, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 3191–3199, 2017b.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. 1990.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning, An Introduction, Second Edition*. MIT Press, 2018.

Erik Talvitie. Agnostic system identification for monte carlo planning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Adv. in Neural Information Processing Systems*, pages 2154–2162, 2016.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Gerald Tesauro. TD-gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer, 1995a.

Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995b.

Gerald Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199, 2002.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

Julian Togelius, Alex J Champandard, Pier Luca Lanzi, Michael Mateas, Ana Paiva, Mike Preuss, and Kenneth O Stanley. Procedural content generation: Goals, challenges and actionable steps. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. Deep reinforcement learning for general video game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 221–228, 2018.

Hui Wang, Michael Emmerich, Mike Preuss, and Aske Plaat. Alternative loss functions in AlphaZero-like self-play. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 155–162, 2019a.

Hui Wang, Mike Preuss, Michael Emmerich, and Aske Plaat. Tackling Morpion Solitaire with AlphaZero-like Ranked Reward reinforcement learning. *arXiv preprint arXiv:2006.07970*, 2020.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *preprint arXiv:1907.02057*, 2019b.

Christopher JCH Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Théophane Weber, Sébastien Racanière, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5690–5701, 2017.

Lilian Weng. Meta-learning: Learning to learn fast. Lil'Log https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html, November 2018.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning—the good, the bad and the ugly. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4582–4591, 2017.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100, 2020.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Neng-Fa Zhou and Agostino Dovier. A tabled Prolog program for solving Sokoban. *Fundamenta Informaticae*, 124(4):561–575, 2013.