Artificial Intelligence Review manuscript No.
(will be inserted by the editor)

High-Accuracy Model-Based Reinforcement Learning, *a Survey*

Aske Plaat · Walter Kosters · Mike Preuss

March 2, 2022

Abstract Deep reinforcement learning has shown remarkable success in the past few years. Highly complex sequential decision making problems from game playing and robotics have been solved with deep model-free methods. Unfortunately, the sample complexity of model-free methods is often high. Model-based reinforcement learning, in contrast, can reduce the number of environment samples, by learning an explicit internal model of the environment dynamics.

However, achieving good model accuracy in high dimensional problems is challenging. In recent years, a diverse landscape of model-based methods has been introduced to improve model accuracy, using methods such as probabilistic inference, model-predictive control, latent models, and end-to-end learning and planning. Some of these methods succeed in achieving high accuracy at low sample complexity in typical benchmark applications. In this paper, we survey these methods; we explain how they work and what their strengths and weaknesses are. We conclude with a research agenda for future work to make the methods more robust and applicable to a wider range of applications.

Keywords Model-based reinforcement learning · Latent models · Deep learning · Machine learning · Planning

1 Introduction

Recent breakthroughs in game playing and robotics have shown the power of deep reinforcement learning, for example, by learning to play Atari and Go from scratch or by learning to fly an acrobatic model helicopter (Mnih et al., 2015; Silver et al., 2016; Abbeel et al., 2007). Unfortunately, for most applications the training times for finding the optimal policy are large (Silver et al., 2016; LeCun et al., 2015), and achieving faster learning is a major topic in current research. Most reinforcement learning is model-free, and model-based methods can achieve faster learning by making an internal dynamics model of the environment. By then

Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands

E-mail: a.plaat@liacs.leidenuniv.nl

using the internal model for policy updates, the number of necessary environment samples can be reduced substantially (Sutton, 1991).

The success of the model-based approach hinges on the accuracy of this dynamics model—there is a trade-off between accuracy and sample complexity, and deep models tend to overfit (LeCun et al., 2015; Talvitie, 2015). Thus the promise of model-based methods, to reduce long training times, depends on the accuracy of the model. The challenge for the methods in this survey is *to train a high-accuracy dynamics model for high-dimensional problems with few samples*. Many approaches have been developed recently, some successful. The goal of the current paper is to survey these methods, and to explain the principles behind their success.

This survey contributes an overview of recent high-accuracy model-based methods for high-dimensional problems. We explain the challenge that model-based reinforcement learning must overcome, and we present a taxonomy based on learning method and planning method. While improving model accuracy is difficult, successful methods are presented for game playing and visuo-motor control. We describe how and why the methods work—we do note, however, that the computational cost is still high, and that the outcomes of experiments are often sensitive to the choice of hyperparameters. We close with a research agenda to improve reproducibility, to further improve accuracy, to make methods more widely applicable, and to make connections with other fields.

The field of deep model-based reinforcement learning is quite active. Previous surveys provide an overview of the uses of classic (tabular) model-based methods (Deisenroth et al., 2013; Kober et al., 2013; Kaelbling et al., 1996). The purpose of the current survey is to focus on *deep* learning methods. Other relevant surveys into model-based reinforcement learning are (Justesen et al., 2019; Polydoros and Nalpantidis, 2017; Hui, 2018; Wang et al., 2019; Çalışır and Pehlivanoğlu, 2019; Moerland et al., 2020b). Excellent works with background information exist for reinforcement learning (Sutton and Barto, 2018) and deep learning (Goodfellow et al., 2016).

Section 2 provides necessary background on Markov decision processes and planning and learning approaches in reinforcement learning. Section 3 then surveys the field. Section 4 provides a discussion reflecting on the different approaches and provides open problems and research directions for future work. Section 5 concludes the survey.

2 Background

This section starts with an intuitive description of the field of deep model-based reinforcement learning, followed by a more formal introduction using Markov decision processes.

Reinforcement learning finds solutions for sequential decision problems; these are problems in which not a single step, but a sequence of decisions must be made in order to reach the solution for start state s_0. A sequence of states s_t, actions a_t, and rewards r_t for time $t = 0, T$ is:

$$s_0, a_1, s_1, r_1, a_2, s_2, r_2, a_3, s_3, r_3, \ldots, r_T$$

Examples of sequential decision problems are grid-world mazes, where a sequence of moves must be made before the end goal is reached (Figure 1 shows a picture of the Taxi problem), or robotic manipulation tasks, where a correct sequence of joint movements must be made in order to achieve a goal, such as pouring a drink in a glass. In reinforcement learning a policy function is trained on reward values that may only be available after many action decisions have been taken—intermediate rewards may be zero. A flat reward landscape can

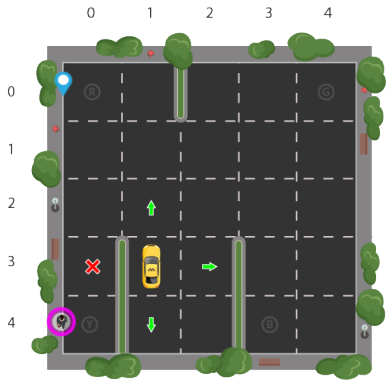

Fig. 1 Taxi in a Grid world (Dietterich, 1998)

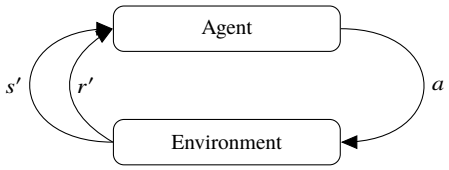

Fig. 2 Reinforcement learning: agent performs action a on environment, which provides new state s' and reward r'

be enhanced using *reward shaping*, for example by adding heuristics such as the euclidean distance to the goal to the reward function at intermediate states (Ng et al., 1999).

Reinforcement learning is a form of interactive learning (Kaelbling et al., 1996). Where in supervised learning a static dataset provides labeled training pairs $\{(x, y)\}$, in reinforcement learning the data is generated in an action/reward interaction between an *agent* and an *environment* that provides the feedback to learn from, see Figure 2. In state s the agent policy chooses action a, for which the environment then returns a new state s' and a corresponding reward value r', that signals the desirability of the new state (Sutton and Barto, 2018). In this way, as many (state, action) pairs can be generated as needed.[1] The goal of reinforcement learning is to learn optimal behavior for a certain environment, maximizing expected cumulative future reward. This goal is reached after a sequence of decisions is taken; the best sequence of actions solves the sequential decision making problem.

Reinforcement learning agents learn by trial and error. The most basic approach, the model-free approach, samples an action sequence of the decision problem (episode), finds the reward, which it then uses to update the view of the best action sequence at the intermediate states.

Where model-free agents learn to take the best action in each state, model-based methods go a step further: a full model of the transitions of the environment is learned by the agent. Model-based methods capture the core of complex decision sequences by learning a local transition model, and models may also be applicable to related environments (Risi and Preuss, 2020; Torrado et al., 2018), for transfer learning, or for explainable AI (Heuillet et al., 2021).

[1] A dataset is static. In reinforcement learning the choice of actions may depend on the rewards that are returned during the learning process, giving rise to a dynamic, potentially unstable, learning process.

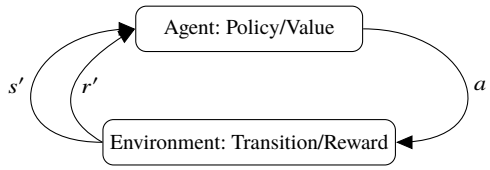

Fig. 3 Model-Free Learning: Agent learns policy from rewards and states that the Environment's transition function calculates

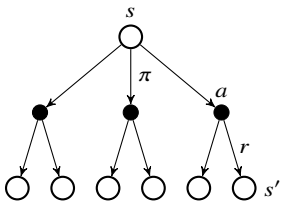

Fig. 4 Backup Diagram (Sutton and Barto, 2018). Maximizing the reward for state s is performed by following the *transition* function to find the next state s'. Note that the policy $\pi(s, a)$ tells the first half of this transition, going from $s \rightarrow a$; the transition function $T_a(s, s')$ completes the transition, going from $s \rightarrow s'$ (via a).

The main challenge of model-based methods is to find accurate transition models. We now arrive at the core challenge facing the research of this survey. In high-dimensional sequential decision problems the transition model is often a deep neural network, that may be prone to overfitting. One way to reduce overfitting is to take many environment samples, but increasing sample complexity negates the main advantage of model-based methods. The challenge for model-based reinforcement learning of high-dimensional sequential decision problems is to learn an accurate model with low sample complexity, and this survey will provide an overview of the state of the art of the methods that were designed to do so.

Reinforcement learning draws inspiration from human and animal learning (Hamrick, 2019; Kahneman, 2011; Anthony et al., 2017; Duan et al., 2016), where behavioral adaptation by reward and punishment is studied.

2.1 Formalizing Model-Based Reinforcement Learning

Sequential decision problems are modeled as Markov decision processes, MDP. In this section we introduce the main notation and MDP formalism as it is used in the field of reinforcement learning. The field has settled on a standard notation; see also (Littman, 1994; Bishop, 2006; Sutton and Barto, 2018; Plaat, 2022). First we introduce state, action, transition and reward. Then we introduce trajectory, policy and value. Finally, we discuss model-based and model-free solution approaches.

We define a Markov decision process as a 5-tuple (S, A, T_a, R_a, γ) 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, and γ is a discount factor to reduce the impact of future reward values for the present. Figure 4 shows a visual diagram of states, actions, and transitions.

A policy π is a stochastic $\pi(a|s)$ or deterministic function $\pi(s) \rightarrow a$ mapping states to action(-distributions). The goal of the agent is to learn a policy that maximizes the value function, the expected cumulative discounted sum of rewards $V(s) = \mathbb{E}_\tau\left[\sum_{t=0}^{T} \gamma^t r_t\right]$ in a trajectory τ, with γ a discount parameter in an episode with T steps. An optimal policy $\pi^\star$ contains a solution to a sequential decision problem: a prescription of which action must be taken in each state to reach the optimal outcome.

The transition model $T_a(\cdot)$ computes the distribution for the next state s' in model-free reinforcement learning; the policy is learned directly from environment feedback r' (Figure 3). In contrast, in model-based reinforcement learning, the agent constructs its own version of the transition model, and the policy can be learned from the environment feedback *and* with the help of the local transition and reward model.

The policy is computed with two elements: the learning method and the selection method. First we will discuss the learning method. The function $V(s)$ is called the state-value function of the state. $V(s)$ is more fully written as $V^\pi(s)$, where the superscript π indicates that the value of state s has been computed by following policy π from that state.

Closely related to the state-value function is the state-action-value function $Q^\pi(s, a)$. This Q-function gives the expected sum of discounted rewards for action a in state s, and then afterwards following policy π. The optimal policy can be found by recursively choosing the argmax action with $Q(s, a) = V(s)$ in each state. This relationship is given by $\pi^\star = \max_\pi V^\pi(s) = \max_{\pi,a} Q^\pi(s, a)$. Algorithms that compute the policy by first finding the value function are called value-based methods. Algorithms that compute a parameterized policy directly are called policy-based methods. Actor-critic algorithms combine both methods (Sutton and Barto, 2018; Konda and Tsitsiklis, 2000; Mnih et al., 2016).

In deep learning, functions such as the policy function π are approximated by the parameters (or weights) θ of a deep neural network, and are written as π_θ, to distinguish them from classical tabular policies.

In addition to the choice for learning function, reinforcement learning algorithms consist of a selection method, that greatly influences its efficiency. There are many algorithms to find optimal policies (Kaelbling et al., 1996; Bertsekas and Tsitsiklis, 1996; Kober et al., 2013; Sutton and Barto, 2018; Hessel et al., 2018): algorithms that use an agent's transition function directly to find the next state are called planning algorithms, algorithms that use the environment to find the next state are called learning algorithms (Moerland et al., 2020b). We now briefly discuss classical model-free learning approaches (Section 2.2), and planning approaches (Section 2.3), before we continue to survey model-based algorithms in more depth in the next section.

2.2 Model-Free Learning

When the agent does not have a local transition or reward model, then the policy can be learned by querying the environment, in order to find the reward for the action in a certain state. Learning the policy or value function in this way is called model-free learning, see Figure 3.

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 1 shows high-level steps of model-free reinforcement learning (later on the algorithms become more elaborate).

Model-free reinforcement learning is the most basic form of reinforcement learning (Kaelbling et al., 1996; Deisenroth et al., 2013; Kober et al., 2013). It has been successfully applied

Algorithm 1 Model-Free Learning

repeat
  Sample env E to generate data $D = (s, a, r', s')$
  Use D to update policy $\pi(s, a)$ ▷ distribute the reward value over the decision points
until π converges

to a range of challenging problems, such as described in (Mnih et al., 2015; Abbeel et al., 2007). In model-free reinforcement learning a policy is learned from the ground up through interactions with the environment.

The goal of classic model-free learning is to find the optimal policy for the states of the environment; the goal of deep model-free learning is to find a policy function that generalizes well to states from the environment that have not been seen during training. A secondary goal is to do so with good sample efficiency: to use as few environment samples as possible.

Model-free learning follows the current behavior policy π in selecting the action to try, deciding between exploration of new actions and exploitation of known good actions with a selection rule such as ϵ-greedy (Sutton and Barto, 2018). Exploration is essentially blind, and learning the policy and value often takes many samples, millions in current experiments (Mnih et al., 2015; Wang et al., 2019).

A well-known model-free reinforcement learning algorithm is Q-learning (Watkins, 1989). Algorithms such as Q-learning were developed in a classical tabular setting. Deep neural networks have 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 Optimization, 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).

Model-free methods select actions in a straightforward manner, without using a separately learned local transition model. An advantage of the straightforward action selection is that they can find global optima without suffering from selection bias from model imperfections. Model-based methods may not always be able to find as good policies as model-free can.

A disadvantage of model-free methods is that interaction with the environment can be costly. Especially when the environment involves the real world, such as in real-world robot-interaction, then sampling should be minimized, for reasons of cost, and to prevent wear of the robot. In virtual environments on the other hand, model-free approaches have been quite successful (Mnih et al., 2015). An overview of model-free reinforcement learning can be found in (Sutton and Barto, 2018; Kaelbling et al., 1996).

2.3 Planning

When an agent has an internal transition and reward model, then planning algorithms can use it to find the optimal policy. They select actions in states, look ahead, and back up reward values, see Figure 5. Planning algorithms require access to an explicit transition 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, backing up the value from the child state to the parent state (see the backup diagram in Figure 4). The policy function $\pi(s, a)$ concerns the top layer

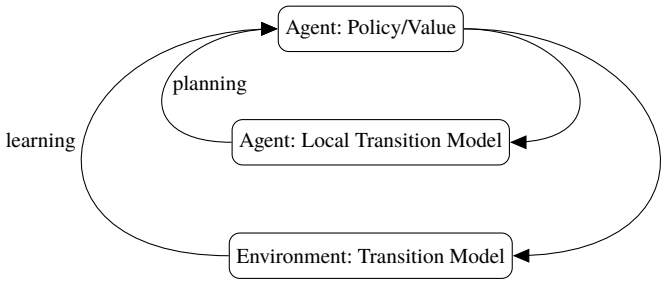

Fig. 5 Model-Based: Planning and Learning

Algorithm 2 Value Iteration (Alpaydin, 2020)

```
Initialize V(s) to arbitrary values
repeat
    for all s do
        for all a do
            Q[s, a] = ∑_{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
```

of the diagram, from s to a. The transition function $T_a(s, s')$ covers both layers, from s to s'. The transition function defines a space of states in which the planning algorithm can search for the optimal policy function $\pi^\star$ and value function $V^\star$.

Bellman's dynamic programming is a basic planning approach to recursively find the value of a state s (Bellman, 1957, 2013). Bellman's equation defines a recursive traversal of the state space, when the transition function and policy are given.

$$V^\pi(s) = R_{\pi(s)}(s, s') + \sum_{s' \in S} T_{\pi(s)}(s, s')\, \gamma\, V^\pi(s') \tag{1}$$

Based on this equation, value iteration straightforwardly finds the value of a state. The pseudo-code for value iteration is shown in Algorithm 2 (Alpaydin, 2020). It traverses all actions in all states, computing the value function for the entire state space, until the value function converges.

When the agent has an accurate local transition model, planning algorithms can be used to find the best policy. This approach is sample efficient since a policy is found without interaction with the environment.

Note that a sampling action performed 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 the agent's local transition model is reversible (Moerland et al., 2020a, 2018). A planning agent can backtrack, a sampling agent cannot. The ability to backtrack is especially useful to try alternatives to further improve on local optima—local optima can be found easily by sampling; global optima may require efficiently the ability to backtrack out of a local optimum.

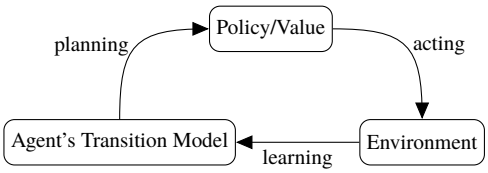

Fig. 6 Model-Based Reinforcement Learning

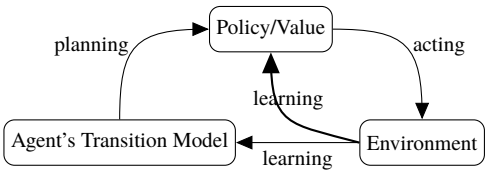

Fig. 7 Dyna's Model-Based Imagination

Algorithm 3 Model-Based Reinforcement Learning

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

2.4 Model-Based Learning

It is now time to look at model-based reinforcement learning. In this approach the policy and value function are learned by both sampling and planning.

Recall that the environment samples return (s', r') pairs when the agent selects action a in state s. This means that we can learn the transition model $T_a(s, s')$ and the reward model $R_a(s, s')$, for example by supervised learning, since all necessary information is present. When the transition and reward model are present in the agent, they can then be used with planning to update the policy and value functions as often as we like *without any further sampling of the environment* (although we might want to continue sampling to further improve our models). This alternative approach of finding the policy and the value is called model-based learning, see Algorithm 3 and Figure 6.

Why would we want to go this convoluted learning-and-planning route, if the environment samples can teach us the optimal policy and value directly? The reason is that the convoluted route may be more sample efficient. In model-free learning a sample is used once to optimize the policy, and then thrown away, while 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 by the agent.

A well-known classic model-based approach is *imagination*, which was introduced by Sutton (1990, 1991) in the Dyna system, long before deep learning was used widely. Dyna uses the environment samples to update the policy function directly (model-free learning) and also uses the samples to learn a transition model, to augment the model-free environment-samples with the model-based imagined "samples." Imagination is a hybrid algorithm that uses both model-based planning and model-free learning to improve the behavior policy. Figure 7 illustrates the working of the Dyna approach. Algorithm 4 shows the

Algorithm 4 Dyna's 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 $T_a(s, s')$ and $R_a(s, s')$
  Use T, R to update policy $\pi(s, a)$ by planning
until π converges

Algorithm 5 Dyna-Q: Classic learning and planning with a Q-function-based dynamics model (Sutton, 1990)

Initialize $Q(s, a) \to \mathbb{R}$ randomly
Initialize $M(s, a) \to \mathbb{R} \times S$ randomly ▷ Model
repeat
  Select $s \in S$ randomly
  $a \leftarrow \pi(s)$ ▷ $\pi(s)$ can be ϵ-greedy(s) based on Q
  $(s', r) \leftarrow E(s, a)$ ▷ Learn new state and reward from environment
  $Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)]$
  $M(s, a) \leftarrow (s', r)$
  for $n = 1, \ldots, N$ **do**
    Select $\hat{s}$ and $\hat{a}$ randomly
    $(s', r) \leftarrow M(\hat{s}, \hat{a})$ ▷ Plan imagined state and reward from model
    $Q(\hat{s}, \hat{a}) \leftarrow Q(\hat{s}, \hat{a}) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(\hat{s}, \hat{a})]$
  end for
until Q converges

steps of the algorithm (compared to Algorithm 3, the line in italics is new, from Algorithm 1). Note how the policy is updated twice in each iteration, by environment sampling, and by transition planning. More details are shown in Algorithm 5 (Sutton, 1990).

2.5 Sample Complexity of Policy and Transition Function

Model-free reinforcement learning methods have achieved impressive success in Atari games and simulated robotics (Mnih et al., 2015; Hessel et al., 2018). However, convergence of the policy function often requires many millions of environment samples, which may be unacceptably high for some real world applications, such as robotics (Wang et al., 2019).

Model-based methods can reduce the number of environment samples significantly for the policy function. However, the deep neural networks that are typically used to solve in high-dimensional problems, may require many samples to train, to reduce the impact of overfitting. Thus, finding an accurate transition function, in order to reduce the sample complexity for the policy function, may itself suffer from high sample complexity. The main challenge for deep model-based reinforcement learning is to find methods that find accurate transition models that do not require a large number of environment samples.

After these introductory words, we are now ready to see what concrete deep model-based reinforcement learning methods have been developed recently.

3 Survey of Model-Based Deep Reinforcement Learning

The success of model-based reinforcement learning in high-dimensional problems depends on the accuracy of the transition and reward model. The model is typically used by planning

Learning	Planning
Probabilistic inference	Trajectory rollouts
Ensemble methods	Model-predictive control
Latent models	End-to-end learning and planning

Table 1 Taxonomy: *Learning* and *Planning*

algorithms for multiple sequential predictions, and errors in predictions accumulate quickly with each step. Model-based reinforcement learning is an active field, and many papers have been published that document progress towards improving model-accuracy. We will look at how these methods were developed.

We will now present our taxonomy. The taxonomy distinghuishes two aspects: (1) the *learning* method and (2) the *planning* method. Each with three approaches. Table 1 summarizes the taxonomy, which is the basis of the remainder of this survey. We will now describe the methods, explaining how they fit together by going through the learning and planning that they use.

First, we will describe the way in which the model is *learned*, and how the accuracy of the model is improved. Among the approaches are probabilistic inference such as Gaussian processes and ensembles, and convolutional neural networks or latent models.

Second, we will describe the way in which the model is subsequently used by the *planner* to improve the behavior policy (Figure 7). These methods aim to reduce the impact of planning with inaccurate models. Among the methods are planning with (short) trajectories, model-predictive control, and integrated end-to-end learning and planning.

The effectiveness of model-based methods depends on whether they fit the application domain in which they are used, and on further aspects of the application. (In Section 4.2 we will look at the performance in applications.) There are two main types of applications, those with continuous action spaces, and those with discrete action spaces. For continuous action spaces, simulated physics robotics in MuJoCo is a favorite test bed (Todorov et al., 2012). For discrete action spaces many researchers use mazes or blocks puzzles. For large, high dimensional, problems the Arcade Learning Environment is used, where the input consists of the screen pixels, and the output actions are the joystick movements (Bellemare et al., 2013).

We will use this taxonomy to categorize and understand the recent literature on high-accuracy model-based reinforcement learning. We list some of the papers in Table 2, which provides an overview of many of the methods that we discuss in this survey. We will explain the main issues and challenges in the field step by step, using the taxonomy as guideline, illustrating solutions to these issues and challenges with approaches from the papers from the table.

Figure 8 illustrates how the approaches of the papers influence each other. Note that, as is often the case in reinforcement learning, the influence has two origins: policy-based methods for continuous action spaces (robotics, upper part), and value-based methods for discrete action spaces (games, lower part). The colors in the figure refer to approaches that are also listed in Table 2.

Table 3 categorizes the methods from Table 2 in our taxonomy. Horizontally the model learning approaches probabilistic inference, ensemble methods, and latent models (and CNN) are listed. These methods were developed to learn more accurate models (at low sample complexity). Vertically we distinguish the planning approaches: trajectory rollouts, model-predictive control, and end-to-end learning and planning. These methods were developed to

Name	Learning	Planning	Application
PILCO (Deisenroth and Rasmussen, 2011)	Probabilistic Inference	Trajectory	Pendulum
iLQG (Tassa et al., 2012)	Probabilistic Inference	MPC	Small
GPS (Levine and Abbeel, 2014)	Probabilistic Inference	Trajectory	Small
SVG (Heess et al., 2015)	Probabilistic Inference	Trajectory	Small
Local Model (Gu et al., 2016)	Probabilistic Inference	Trajectory	MuJoCo
Visual Foresight (Finn and Levine, 2017)	Video Prediction	MPC	Manipulation
PETS (Chua et al., 2018)	Ensemble	MPC	MuJoCo
MVE (Feinberg et al., 2018)	Ensemble	Trajectory	MuJoCo
Meta Policy (Clavera et al., 2018)	Ensemble	Trajectory	MuJoCo
MBPO (Janner et al., 2019)	Ensemble	Trajectory	MuJoCo
PlaNet (Hafner et al., 2019)	Latent	MPC	MuJoCo
Dreamer (Hafner et al., 2020)	Latent	Trajectory	MuJoCo
Plan2Explore (Sekar et al., 2020)	Latent	Trajectory	MuJoCo
Video-prediction (Oh et al., 2015)	Latent	Trajectory	Atari
VPN (Oh et al., 2017)	Latent	Trajectory	Atari
SimPLe (Kaiser et al., 2019)	Latent	Trajectory	Atari
Dreamer-v2 (Hafner et al., 2021)	Latent	Trajectory	Atari
MuZero (Schrittwieser et al., 2020)	Latent	e2e/MCTS	Atari/Go
VIN (Tamar et al., 2016)	CNN	e2e	Mazes
VProp (Nardelli et al., 2018)	CNN	e2e	Mazes
Planning (Guez et al., 2019)	CNN/LSTM	e2e	Mazes
TreeQN (Farquhar et al., 2018)	Latent	e2e	Mazes
I2A (Racanière et al., 2017)	Latent	e2e	Mazes
Predictron (Silver et al., 2017b)	Latent	e2e	Mazes
World Model (Ha and Schmidhuber, 2018b)	Latent	e2e	Car Racing

Table 2 Overview of High-Accuracy Model-Based Reinforcement Learning Methods; Top: Continuous/Policy-based, Bottom: Discrete/Value-based

	Probabilistic Inference	**Ensemble Methods**	**Latent Models/CNN**
Trajectory Rollouts	PILCO GPS	MVE Meta Policy MBPO	Dreamer 1,2 VPN, SimPLe Plan2Explore
Model Predictive Control	iLQG	PETS	PlaNet Video Prediction
End-to-end Learning & Planning			VIN, VProp TreeQN, Planning I2A, Predictron MuZero, World Model

Table 3 Methods in the Taxonomy of Learning (horiz.) and Planning (vert.)

allow planning with inaccurate models. The Table summarizes how the papers in this survey fall into these categories.

Let us now start with the taxonomy. We begin with *learning*, next is *planning*. After the taxonomy, we discuss the performance and applications in Section 4.2.

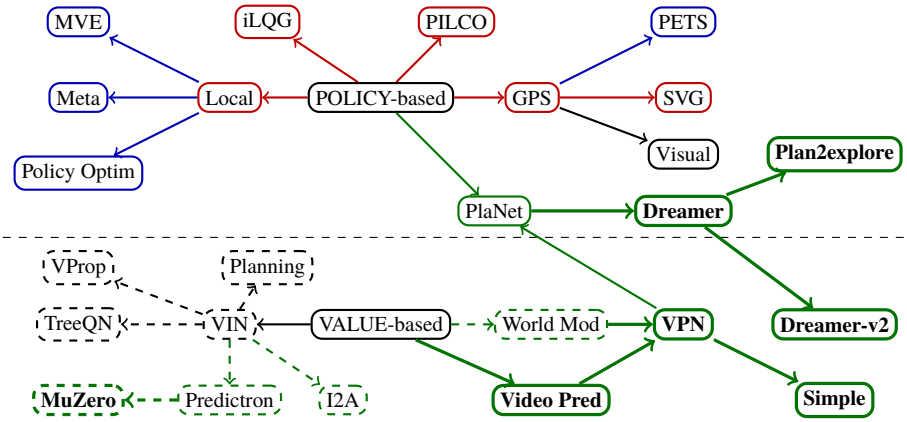

Fig. 8 Influence of Model-Based Deep Reinforcement Learning Approaches; Top: Continuous/Policy-based (MuJoCo), Bottom: Discrete/Value-based (Mazes, Atari); Red: Probabilistic inference, Blue: Ensemble, Green: Latent Models, Dashed: end-to-end, Bold: Large Problems

3.1 Learning

The main promise of model-based reinforcement learning is to reduce the sample complexity of learning the policy, by learning a local transition model. Getting high-accuracy models with few samples is challenging when the model has many parameters, since, in order to prevent overfitting, we would need many environment observations (high sample complexity).

The transition model is what gives model-based reinforcement learning its name. The accuracy of the model is of great importance, planning with inaccurate models will not improve the policy much, planning with a biased model will even harm the policy, and performance of model-based methods will be worse than the model-free baseline (Gu et al., 2016).

In this section we will describe techniques that have been developed to improve model accuracy. The methods either focus on reducing bias and variance of the model, or on reducing the dimensionality of the problem. We will discuss:

1. probabilistic inference,
2. ensemble methods,
3. latent models.

3.1.1 Probabilistic Inference

One of the shortcomings of conventional reinforcement learning methods is that they only focus on expected value, ignoring the variance of values. Naive sample methods suffer from high bias or high variance. This is problematic when few samples are taken for each trajectory, and model-based approaches may suffer from bias (Schneider, 1996; Schaal, 1996). Improved sampling methods from optimization and statistics offer methods to reduce bias and variance, even when few samples are taken. Gaussian processes can learn simple processes with good sample efficiency, reducing model bias. They have been used for probabilistic inference to learn control models (Deisenroth and Rasmussen, 2011) in the PILCO system. This system was effective on Cartpole and Mountain car (Figure 12).

A related method uses nonlinear least-squares optimization (Tassa et al., 2012). Here the model learner uses quadratic approximation on the reward function, which is then used with linear approximation of the transition function. With further enhancements this method was able to teach a humanoid robot how to stand up (see Figure 10).

Another idea to reduce bias is to sample from a trajectory distribution optimized for cost, and to train the policy with a policy-based method, optimizing policies with a locally-linear model and a stochastic trajectory optimizer (Levine and Koltun, 2013). This approach, called Guided policy search (GPS), has been shown to train complex policies with thousands of parameters learning tasks in MuJoCo such as swimming, hopping and walking. Alternatively, we can compute value gradients along the real environment trajectories, instead of planned ones, and re-parameterize the trajectory through sampling, to mitigate learned model inaccuracy (Heess et al., 2015). This was done by Stochastic value gradients (SVG) with global neural network value function approximators.

Probabilistic inference methods such as Gaussian processes and nonlinear optimization methods can sample reliable prediction models from few samples, although their computational complexity is large, and they do not scale to high-dimensional problems. These methods work for problems of moderate complexity, where the state of the physics model is provided to the agent as a vector of numbers.

Learning arm and hand manipulation directly from video camera input is a challenging problem in robotics. The camera image provides a high dimensional input and increases problem size and complexity of the subsequent manipulation task substantially. Both Finn and Levine (2017); Ebert et al. (2018) introduce a method called Visual foresight. This system uses a training procedure where data is sampled according to a probability distribution. Concurrently, a video prediction model is trained. This model generates a 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 (see Section 3.2.2). This approach is able to perform multi-object manipulation, pushing, picking and placing, and cloth-folding tasks (which adds the difficulty of material that changes shape as it is being manipulated). This method combines probabilistic inference methods with dimensionality reduction. In Section 3.1.3, on latent models, more approaches are presented where dimensionality reduction is used.

3.1.2 Ensemble Models

Ensemble methods reduce bias and variance by combining results from different methods or results from different runs of the same method (Bishop, 2006). Ensemble methods, such as a random forest of decision trees (Ho, 1995), are widely used in machine learning, and they are also used in reducing bias and variance in high-dimensional modeling.

Ensemble methods are used with success in model-based deep reinforcement learning as well. Chua et al. (2018) combine variance-aware modeling with sampling-based uncertainty propagation, creating a method called Probabilistic ensembles with trajectory sampling, PETS. (This approach is also described in the next section, see Algorithm 6). An ensemble of probabilistic neural network models is used by Nagabandi et al. (2018). Ensembles perform well on medium sized problems; performance on pusher, reacher, and half-cheetah (see Figure 10) is reported to approach asymptotic model-free baselines such as PPO (Schulman et al., 2017). Ensembles of probabilistic networks (Chua et al., 2018) are also used with short rollouts, where the model horizon is shorter than the task horizon (Janner et al., 2019), MBPO. Results have been reported for hopper, walker, and half-cheetah, again matching the performamce of model-free approaches.

The ensemble approach is related to meta learning, where we try to speed up learning a new task by learning from previous, related, tasks (Brazdil et al., 2022; Hospedales et al., 2020; Huisman et al., 2021). MAML is a popular meta learning approach (Finn et al., 2017), that attempts to learn a network initialization such that for any task the policy attains maximum performance after one policy gradient step. The MAML approach can be used to improve model accuracy by learning an ensemble of dynamics models and by then meta-optimizing the policy for adaptation in each of the learned models (Clavera et al., 2018). Results indicate that such meta-learning of a policy over an ensemble of learned models indeed approaches the level of performance of model-free methods with substantially better sample complexity.

Ensembles of probabilistic models reduce bias and variance, and achieve improved results on medium sized problems. For high dimensional problems, the computational complexity of the methods remains problematic, and the improvement in accuracy is insufficient to out-perform model-free method (Wang et al., 2019). To achieve good performance on high dimensional problems, we will discuss the latent model approach.

3.1.3 Latent Models

Probabilistic inference and ensemble methods reduce bias and variance, but are computationally intensive methods. A third idea is to use dimensionality reduction methods, and then to find the transition model in a lower dimensional space.

The next group of methods that we describe are the latent models. Central to all our approaches is the need for improvement of model accuracy in complex, high-dimensional, problems. The main challenge to achieve high accuracy is to reduce the size of the high-dimensional state space. The idea behind latent models is that in most high-dimensional environments there are elements that are less important, such as background trees that never move, that have little or no relation with the reward of the agent's actions. The goal of latent models is to abstract away these unimportant elements of the input space, reducing the effective dimensionality of the space. They do so by learning the relation between the elements of the input and the reward either by auto encoding or with supervised learning conditioned on value. When we focus our learning mechanism on the changes in observations that are correlated with changes in these values, then we can improve the efficiency of learning high-dimensional problems greatly. Latent models thus learn a smaller representation, smaller than the observation space. Planning takes place in this smaller representation space.

The value prediction network (VPN) was introduced by Oh et al. (2015, 2017) to achieve this goal. They ask the question in their paper: "What if we could predict future rewards and values directly without predicting future observations?" and describe a nework architecture and learning method for such focused value prediction models. The core idea is not to learn directly in actual observation space, but first to transform the actual state respresentation to a smaller latent representation model, also known as abstract model. The other functions, such as value, reward, and next-state, then work with the smaller latent representations, instead of the actual high-dimensional states. By training all functions based on the values (Grimm et al., 2020), planning and learning occur in a space where states are encouraged only to contain the elements that influence value changes. In VPN the latent model consists of four networks: an (LSTM) encoding function, a reward function, a value function, and a transition function. All functions are parameterized with their own set of parameters (Figure 9). Latent space is lower-dimensional, and training and planning become more efficient. The figure shows a single step rollout, planning one step ahead, as in Dyna-Q (Algorithm 4).

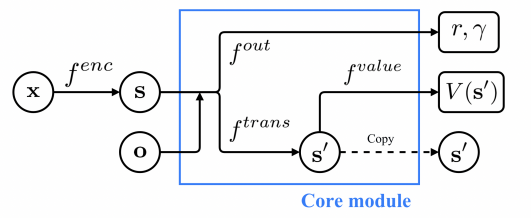

Fig. 9 Architecture of latent model (Oh et al., 2017)

The training of the networks can in principle be performed with any value-based reinforcement learning algorithm. Oh et al. (2017) report results with n-step Q-learning and temporal difference search (Silver et al., 2012).

VPN (Oh et al., 2017) showed impressive results on Atari games such as Pacman and Seaquest, outperforming model-free DQN (Mnih et al., 2015), and outperforming observation-based planning in stochastic domains. Subsequently, many other works have been published that further improved results (Kaiser et al., 2019; Hafner et al., 2019, 2020, 2021; Sekar et al., 2020; Silver et al., 2017b; Ha and Schmidhuber, 2018b). Many of these latent-model approaches are complicated designs, with multiple neural networks, and different learning and planning algorithms.

The latent-model approach is related to 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 often generative recurrent neural networks that are trained unsupervised using a variational autoencoder (Kingma and Welling, 2014, 2019; Goodfellow et al., 2014) and a recurrent network. They learn a compressed spatial and temporal representation of the environment. In world models multiple neural networks are used, for a vision model, a memory model, and a controller (Ha and Schmidhuber, 2018b). 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. World models have been applied by Ha and Schmidhuber (2018a,b) on a car racing game (Kempka et al., 2016). The term *world model* actually goes back to 1990, where it was used by Schmidhuber (1990b). Latent models are related to dimensionality reduction (Van Der Maaten et al., 2009).

The architecture of latent models, or world models, is elaborate. The dynamics model typically includes an observation model, a representation model, a transition model, and a value or reward model (Karl et al., 2016; Buesing et al., 2018; Doerr et al., 2018). The task of the observation model is to reduce the high-dimensional world into a lower-dimensional world, to allow more efficient planning. Often a (variational) autoencoder or LSTM is used.

The Arcade learning environment is one of the main benchmarks in reinforcement learning. The high-dimensionality of Atari video input has long been problematic for model-based reinforcement learning. Latent models were instrumental in reducing the dimensionality of Atari, producing the first successes for model-based approaches on this major benchmark.

Related to the VPN approach (Oh et al., 2015, 2017) are latent model approaches, such as Kaiser et al. (2019), that are aimed at video prediction, outperforming model-free baselines (Hessel et al., 2018), reaching comparable accuracy with up to an order of magnitude better sample efficiency. The approach by Kaiser et al. (2019) uses a variational autoencoder (VAE) to process input frames, conditioned on the actions of the agent, to learn the world model, using PPO (Schulman et al., 2017). The policy π is then improved by

planning inside the reduced world model, with short rollouts. This behavior policy π then determines the actions a to be used for learning from the environment.

Latent models are also used on continuous MuJoCo problems. Here the work by Hafner et al. (2019, 2020) on the PlaNet and Dreamer systems is noteworthy. They include the application of their work back to Atari (Hafner et al., 2021), which achieved human-level performance. PlaNet uses a Recurrent state space model (RSSM) that consists of a transition model, an observation model, a variational encoder and a reward model (Karl et al., 2016; Buesing et al., 2018; Doerr et al., 2018). Based on these models a Model-predictive control agent is used to adapt its plan, replanning each step (Richards, 2005). The RSSM is used by a Cross entropy method search (Botev et al. (2013), CEM) for the best action sequence. In contrast to model-free approaches, no explicit policy or value function network is used; the policy is implemented as MPC planning (see next section) with the best sequence of future actions. PlaNet is tested on continuous tasks and reaches performance that is close to strong model-free algorithms.

A further system, called Dreamer (Hafner et al., 2020), builds on PlaNet. Using an actor critic approach (Mnih et al., 2016) and backpropagating value gradients through predicted sequences of compact model states, the improved system solves a diverse collection of continuous problems from the Deepmind control suite (Tassa et al., 2018), see Figure 10. Dreamer is also applied to discrete problems from the Arcade learning environment, and to few-shot learning (Sekar et al., 2020). A further improvement achieved human-level performance on 55 Atari games, a first for a model-based approach (Hafner et al., 2021), showing that the latent model approach is well-suited for high-dimensional problems.

3.2 Planning

After the transition model has been learned, it will be used with a planning algorithm to improve the behavior policy (Figure 6). Since the transition model will contain inaccuracies, the challenge is to find a planning algorithm that performs well despite the inaccuracies. We describe three groups of methods that have been developed for planning algorithms to cope with inaccurate models. These are:

1. trajectory rollouts,
2. model-predictive control,
3. end-to-end learning and planning.

Trajectory rollouts and model-predictive control have been shown to work for both continuous and discrete action spaces; model-predictive control can be regarded as a refinement of short trajectory rollout. End-to-end learning and planning has been developed in the context of discrete action spaces (mazes and games), and build on a long history of differentiable learning algorithms.

Of the three planning methods, we will start with the trajectory rollouts.

3.2.1 Trajectory Rollouts

As we saw in Section 2.1, methods for continuous action spaces typically sample full trajectory rollouts to get stable actions. At each planning step, the transition model $T_a(s) \rightarrow s'$ computes the new state, using the reward to update the policy. Due to the inaccuracies of the internal model, planning algorithms that perform many steps will quickly accumulate model errors (Gu et al., 2016). Full rollouts of long and inaccurate trajectories are therefore

problematic. We can reduce the impact of accumulated model errors by not planning too far ahead. For example, Gu et al. (2016) perform experiments with locally linear models that roll out planning trajectories of length 5 to 10. This reportedly works well for MuJoCo tasks gripper and reacher.

In their work on model-based value expansion (MVE), Feinberg et al. (2018) also allow imagination to fixed depth, value estimates are split into a near-future model-based component and a distant future model-free component. They experiment with model horizons of 1, 2, and 10. They find that 10 generally performs best on typical MuJoCo tasks such as swimmer, walker, and cheetah. The sample complexity in their experiments is better than model-free methods such as DDPG (Silver et al., 2014). Similarly good results are reported by Janner et al. (2019); Kalweit and Boedecker (2017), both approaches use a model horizon that is much shorter than the task horizon.

3.2.2 Model-Predictive Control

Taking the idea of shorter planning trajectories further, we arrive at Model-predictive control (MPC) (Kwon et al., 1983; Garcia et al., 1989). Model-predictive control is a well-known approach in process engineering, to control complex processes with frequent re-planning of a limited time horizon. Model-predictive control uses the fact that while many real-world processes are not linear, they are approximately linear over a small operating range. Applications are found in the automotive industry and in aerospace, for example for terrain-following and obstacle-avoidance algorithms (Kamyar and Taheri, 2014). In optimal control, four MPC approaches are identified: linear model MPC, nonlinear prediction model, explicit control law MPC, and robust MPC to deal with disturbances (Garcia et al., 1989). In this survey, we focus on how the principle of continuous replannning with a rolling planning horizon performs in nonlinear model-based reinforcement learning.

It is instructive to compare the MPC and linear quadratic regulators (LQR) approach, since both methods come from the field of optimal control in engineering. MPC computes the target function with a small time window that rolls forward as new information comes in; it is dynamic. LQR computes the target function in a single episode, using all available information; it is static. We observe that in model-based reinforcement learning MPC is used in the planning part with the behavior policy π being the target and the transition function $T_a(\cdot)$ the input; for LQR the transition function $T_a(\cdot)$ is the target, and the environment samples (s_t, r_t) are the input. Thus, one could conceivably use both MPC and LQR, the first as planning and the second as learning algorithm, in a model-based approach.

An iterative form of LQG has indeed been used together with MPC on a smaller MuJoCo problem (Tassa et al., 2012), achieving good results. MPC used step-by-step real-time local optimization; Tassa et al. (2012) used many further improvements to the trajectory optimization, physics engine, and cost function to achieve good performance.

MPC has also been used in other model learning approaches. Both Finn and Levine (2017); Ebert et al. (2018) use a form of MPC in the planning for their Visual foresight robotic manipulation system (that we have seen in a previous section). The MPC part uses a model that generates the corresponding sequence of future frames based on an image to select the least-cost sequence of actions.

Another approach uses ensemble models for learning the transition model, while using MPC for planning. PETS (Chua et al., 2018) uses probabilistic ensembles (Lakshminarayanan et al., 2017) for learning. In MPC fashion only the first action from the CEM-optimized sequence is used, re-planning at every time-step (see Algorithm 6).

Algorithm 6 PETS MPC (Chua et al., 2018)

Initialize data D with a random controller for one trial
for Trial $k = 1$ to K **do**
  Train a PE dynamics model $\widetilde{T}$ given D
  for Time $t = 0$ to TaskHorizon **do**
    for Actions sampled $\vec{a}_{t:t+T} \sim$CEM$(\cdot)$, 1 to NSamples **do**
      Propagate state particles $\vec{s}_\tau^{\,p}$ using TS and $\widetilde{T}\,|\,\{D, \vec{a}_{t:t+T}\}$
      Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(\vec{s}_\tau^{\,p}, \vec{a}_\tau)$
      Update CEM$(\cdot)$ distribution
    end for
    Execute first action $\vec{a}_t^{\,*}$ (only) from optimal actions $\vec{a}_{t:t+T}^{\,*}$
    Record outcome: $D \leftarrow D \cup \{\vec{s}_t, \vec{a}_t^{\,*}, \vec{s}_{t+1}\}$.
  end for
end for

MPC is a simple and effective planning method that is well-suited for model inaccuracy, by restricting the planning horizon. MPC is used with success in model-based reinforcement learning, with high-variance or complex transition models. MPC has also been used with success in combination with latent models (Hafner et al., 2019; Kaiser et al., 2019).

3.2.3 End-to-End Learning and Planning

Our third planning approach is different, it integrates planning with learning in a fully differentiable algorithm. Let us see how this works.

Model-based reinforcement learning consists of two distinct procedures: *learning* the transition model and *planning* with the model to improve the behavior policy (Figure 6). In classical, tabular, reinforcement learning, both learning and planning procedures are designed by hand, by a human programmer (Sutton and Barto, 2018). In deep reinforcement learning, one of these procedures is approximated by deep learning—the model learning—while the planner is still hand-written. End-to-end learning and planning breaks this hand-written planning barrier. End-to-end approaches integrate the planning into deep learning, using differentiable planning algorithms for the planning part as well, extending the backpropagation fully from reward to observation in all parts of the model-based approach.

Learning the a differentiable model and planning algorithm in an integrated way solves an impedance mismatch (Tamar et al., 2016; Schmidhuber, 1990a): models are trained on single-step targets, yet subsequently used in multi-step planning sequences, for states that no longer match those they were trained on. In integrated end-to-end learning and planning the multi-step planning is inside the differentiable optimization loop, making sure that the model matches its use in the planning algorithm.

How can a neural network learn to plan? While conceptually exciting and appealing, there are challenges to overcome. Among them are finding suitable differentiable planning algorithms and the increase in computational training complexity, since now the planner must also be learned.

The idea of planning by gradient descent exists for some time, several authors explored learning approximations of state transition dynamics in neural networks (Kelley, 1960; Schmidhuber, 1990a; Ilin et al., 2007). Neural networks are typically used to transform and filter, to learn selection and classification tasks. A planner unrolls a state, computes values, using selection and value aggregation, and backtracks to try another state. Although counter-intuitive at first, these operations are not that different from what classic neural networks are performing. A progression of papers has published methods on how this can be achieved.

We will start at the beginning, with convolutional neural networks (CNN) and value iteration. We will see how the state and 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 and state 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.

Let us start to see what is possible with a CNN. A CNN can be used to implement value iteration. This was first shown by Tamar et al. (2016), who introduced value iteration networks (VIN). The core idea is that value iteration (VI, see Algorithm 2) can be implemented step-by-step by a multi-layer convolutional network: each layer does a step of lookahead. In this way VI is implemented in a CNN. The VI iterations for the Q-action-value-function are rolled out in the network layers Q with A channels. Through backpropagation the model learns the value function and the transition function. The aim is to learn a general model, that can navigate in unseen environments.

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. Follow-up studies focus on making end-to-end learning and planning more generally applicable. Schleich et al. (2019) extend VINs by adding abstraction, and Srinivas et al. (2018) introduce universal planning networks, UPN, which generalize to modified robot morphologies. Value propagation (Nardelli et al., 2018) uses a hierarchical structure to generalize end-to-end methods to large problems. TreeQN (Farquhar et al., 2018) incorporates a recursive tree structure in the network, modeling the different functions of an MDP explicitly. TreeQN is applied to Sokoban and nine Atari games.

A further step is to model more complex planning algorithms, such as Monte Carlo Tree Search (MCTS), a successful planning algorithm (Coulom, 2006; Browne et al., 2012). This has been achieved to a certain extent by Guez et al. (2018) who implement many elements of MCTS in MCTSnets and (Guez et al., 2019). In this method planning 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). The architecture was used on Sokoban and boxworld (Zambaldi et al., 2018), and was able to perform full planning steps. Future work should investigate how to achieve sample-efficiency with this architecture.

The question whether model-based planning can be learned by a neural network has been studied by (Pascanu et al., 2017), who showed that imagination-based planning steps can indeed be learned for a small game with an LSTM. Related to this, imagination-augmented agents (I2A) has been designed as a fully end-to-end differentiable architecture for model-based imagination and model-free reinforcement learning (Racanière et al., 2017). It consists of an LSTM-based encoder (Chiappa et al., 2017; Buesing et al., 2018), a ConvLSTM rollout module, and a standard CNN-based model-free path. The policy improvement algorithm is A3C. Racanière et al. (2017) report that on Sokoban and Pacman I2A performs better than model-free learning and MCTS. I2A has been specifically designed to handle model imperfections well and uses a manager or meta-controller to choose between rolling out actions in the environment or by imagination (Hamrick et al., 2017).

In VIN there is a tight connection between the network architecture and the application structure. One way to remedy this restriction is with a latent model, such as the ones that were discussed earlier. One of the first attempts is the Predictron (Silver et al., 2017b),

where the familiar four elements appear: a representation model, a transition model, a reward model, and a value model. The goal of the latent model is to perform value prediction (not state prediction), including being able to encode special events such as "staying alive" or "reaching the next room." Predictron performs limited-horizon rollouts, and has been applied to procedurally generated mazes.

One of the main success stories of model-based reinforcement learning is AlphaZero (Silver et al., 2017a, 2018). AlphaZero combines planning and learning in a highly successful way. The MCTS planner was still hand-coded, in a separate algorithm. Inspired by this approach, a fully differentiable version of this architecture has been introduced by Schrittwieser et al. (2020) which is named MuZero. This system is able to learn the rules and to learn to play games as different as Atari, chess, and Go, purely from the environment, with end-to-end learning and planning. The MuZero architecture is based on Predictron, with an abstract model consisting of a representation, transition, reward, and prediction function (policy and value). For planning, MuZero uses an explicitly coded version of MCTS that uses policy and value input form the network (Rosin, 2011), but that is executed separately from the network.

End-to-end planning and learning has shown impressive results, but there are still open questions concerning the applicability to different applications, and especially the scalability to larger problems.

4 Discussion and Outlook

We have now discussed in depth many methods in a taxonomy of learning and planning. We have seen different innovative approaches, all aiming to achieve similar goals. Let us discuss how well they succeed.

Model-based reinforcement learning promises high accuracy and low sample complexity. Sutton's work on imagination, where a transition model is created with environment samples that are then used to create extra "imagined" samples for the policy 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, as the agent builds up its own model of the environment.

Another, and perhaps more important aspect, is generalization performance. Model-based reinforcement learning builds a dynamics model of the environment. This model can be used multiple times, not only for the same problem instance, but also for new problem instances, and for variations. By learning the state-to-state transition model and the reward model, model-based reinforcement learning captures the essence of a domain, where model-free methods only learn best response actions. Model-based reinforcement learning may thus be better suited for solving transfer learning problems, and for solving long sequential decision making problems. It is the difference between learning how to respond to certain actions of a difficult boss, and *knowing* the boss.

4.1 The Problem

Classical tabular approaches and Gaussian process approaches have been quite succesful in achieving low sample complexity for problems of small to medium complexity (Sutton and Barto, 2018; Deisenroth et al., 2013; Kober et al., 2013). However, the topic of the current

survey is *deep* models, for large, high dimensional, problems with complex, non-linear, and discontinuous functions. These application domains pose a problem for classical approaches.

The main challenge that the model-based reinforcement learning algorithms in this survey address is the following. 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 sample sizes, and model-free methods use many millions of environment samples. For high-dimensional problems, the accuracy of transition models is under pressure. Model-based methods that use poor models make poor planning predictions far into the future (Talvitie, 2015).

The challenge is therefore to learn deep, high-dimensional transition functions from limited data that are accurate—or that account for model variance—and plan over these models to achieve policy and value functions that are as accurate or better than model-free methods.

Our survey shows that the first attempts to improve accuracy is with probabilistic inference methods such as Gaussian processes and ensembles that can reduce bias and variance. Unfortunately, these methods do not scale to high-dimensional problems. Next, latent models (or world models) were use for dimensionality reduction, so that planning and learning can occur in a smaller, lower-dimensional, space, achieving good results. For planning, short roll-outs and re-planning are used to reduce the impact of inaccurate models. Another approach is end-to-end learning and planning, to make sure that models and planners are learned together for multi-step usage.

Let us see how these methods succeed. We recall that the main problem statement is twofold: (1) whether model-based methods perform as good as model-free methods, with better sample complexity, on high dimensional problems, and (2) whether the models that are learned allow generalization to other applications. Section 4.2 discusses the first part, Section 4.3 the second.

4.2 Application Performance

After we have discussed in some depth the learning and planning methods, we must see whether the methods have achieved their goal: achieve adequate performance at lower sample complexity in high-dimensional sequential problems. We will see that the type of application plays an important role in the success of the learning and planning methods.

Which types of sequential decision problems can we distinguish? Two main application areas are robotics and games (although many other sequential decision applications exist, most papers on deep model-based methods either use simulated robotics or Atari games). The actions in robotics are continuous, and the environment is non-deterministic. The actions in games are typically discrete and the environment is often deterministic. To summarize our applications, Table 4 shows which of the methods of Table 2 are applied to small or large, low or high dimensional problems.

4.2.1 Continuous Actions

Sequential decision problems are well-suited to model robotic actions, such as how to move the joints in a robotic arm to pour a cup of tea, how to move the joints of a humanoid figure to stand up when lying down, and how to develop gaits of a four-legged animal. The action space of such problems is continuous since the angles over which robotic joints move span a continuous range of values. Furthermore, the environment in which robots operate mimics

	Probabilistic Inference	Ensemble Methods	Latent Models/CNN
Small tasks	PILCO	MVE	VIN, VProp
Low dimensional	GPS	Meta Policy	TreeQN, Planning
	iLQG	PETS	I2A, Predictron
			World Model
Large tasks			PlaNet, Dreamer 1,2
High dimensional			Video Prediction
			VPN, SimPLe
			MuZero, Plan2Explore

Table 4 Methods in the Application classes

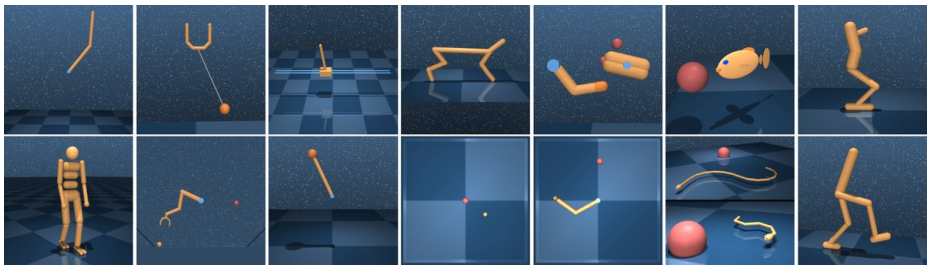

Fig. 10 DeepMind Control Suite. Top: Acrobot, Ball-in-cup, Cart-pole, Cheetah, Finger, Fish, Hopper. Bottom: Humanoid, Manipulator, Pendulum, Point-mass, Reacher, Swimmer (6 and 15 links), Walker (Tassa et al., 2018)

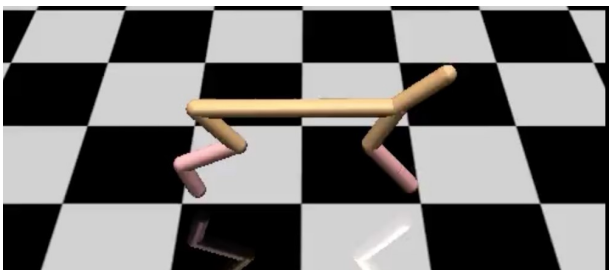

Fig. 11 Half-Cheetah (Todorov et al., 2012)

the real world, and is non-deterministic. Things move, objects do not always respond in a predictable fashion, and unexpected situations arise.

In reinforcement learning, where agent algorithms are trained by the feedback on their many actions, working with real robots would get prohibitively expensive due to wear. Most reinforcement learning systems use physics simulations such as offered by MuJoCo (Todorov et al., 2012). MuJoCo allows the creation of experiments that provide environments for an agent. Tasks can range form small to large.

MuJoCo tasks differ in difficulty, depending on how many joints or degrees of freedom are modeled, and which task is being learned. Figure 10 shows some of the tasks that have been modeled in MuJoCo as part of the DeepMind Control Suite (Tassa et al., 2018). Some of the small tasks are ball-in-cup and reacher. The iterative quadratic non-linear optimization

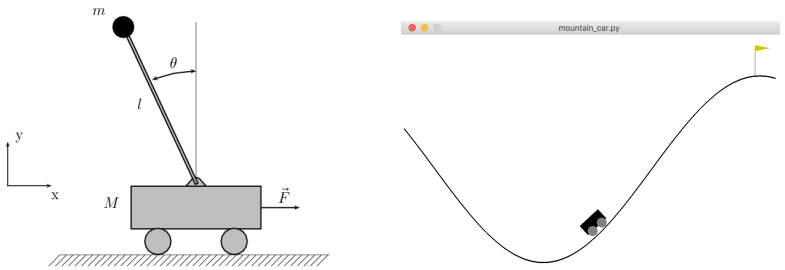

Fig. 12 Cart Pole and Mountain Car (Sutton and Barto, 2018)

method iLQG (Tassa et al., 2012) is able to teach a humanoid to stand up, and Guided policy search (Levine and Koltun, 2013) and Stochastic value gradients (Heess et al., 2015) can learn tasks such as swimmer, reacher, half-cheetah (Figure 11) and walker. Also ensemble methods such as PETS, MVE, and meta ensembles achieve good results on these applications (Gu et al., 2016; Chua et al., 2018; Feinberg et al., 2018; Clavera et al., 2018; Janner et al., 2019).

MuJoCo has enabled progress in model-based reinforcement learning in small continuous tasks, and also in larger tasks, such as how to develop gaits of a four-legged robotic animal, or how to scale an obstacle course. PlaNet (Hafner et al., 2019), Dreamer (Hafner et al., 2020) and MBPO (Janner et al., 2019) achieve good results on more complicated MuJoCo tasks using latent models to reduce the dimensionality.

4.2.2 Discrete Actions

There is a long tradition in reinforcement learning to see if we can teach a computer to play complicated games and puzzles (Plaat, 2020). Games and puzzles are often played on a board with discrete squares. Actions in such games are discrete, a move to square e3 is not a move to square e4. The environments are also deterministic, we assume that pieces do not move by itself.

Most games that are used in deep model-based reinforcement learning papers fall into this category. More complex games, such as partial information (card games such as poker (Brown and Sandholm, 2019)) or games with multiple actors (real-time strategy video games such as StarCraft (Vinyals et al., 2019; Ontanón et al., 2013; Wong et al., 2022)) are not used in the approaches that we survey here.

Among low-dimensional applications that are used in model-based reinforcement learning are simple pendulum problems, Cartpole and Mountain car, where the challenge is to reverse engineer the laws of impulse and gravity (Figure 12). The action space consists of two discrete actions, push left or push right, the environment is continuous and deterministic. PILCO (Deisenroth and Rasmussen, 2011) achieves good results with Gaussian process modeling and gradient based planning on the pendulum task.

Perhaps the most frequently used low-dimensional application area is grid-world, where various navigation tasks are tested (Figure 1). VIN (Tamar et al., 2016), VProp (Nardelli et al., 2018) and the Predictron (Silver et al., 2017b) that use maze navigation to test their approaches to integrating end-to-end learning and planning. Grid worlds and mazes can be

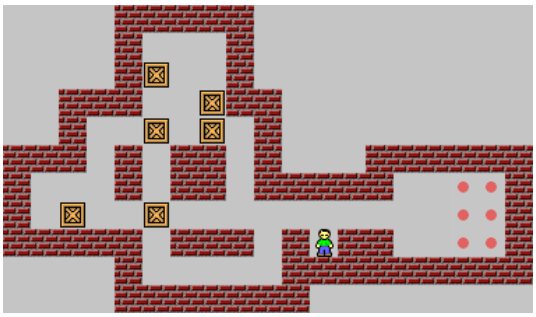

Fig. 13 Sokoban Puzzle (Chao, 2013)

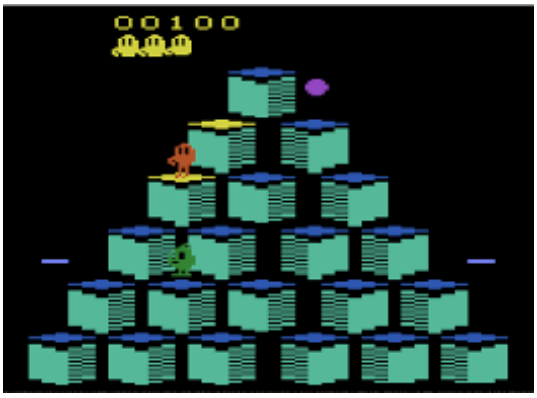

Fig. 14 Q*bert, Example Game from the Arcade Learning Environment (Bellemare et al., 2013)

designed and scaled in different forms and sizes, making them well suited for testing new ideas.

Other low-dimensional games are board games such as chess and Go. These games are low-dimensional (their input has few atttributes, compared to a mega-pixel image) but they nevertheless have a large state space. Finding good policies for chess and Go was one of the most challenging feats in reinforcement learning (Campbell et al., 2002; Silver et al., 2016; Plaat, 2020). Block puzzles such as Sokoban (Figure 13), are also often used to test reinforcement learning methods. Sokoban is a block-pushing puzzle that derives much of its complexity from the facft that the agent can push a box, but cannot pull (undo) a mistake, giving rise to many dead ends that are hard to detect. It has been used by I2A (Racanière et al., 2017) and MCTS network planning (Guez et al., 2019) approaches that implement planning by unrolling steps within a neural network.

Most recent success in model-free reinforcement learning has been achieved in high-dimensional problems, such as the Arcade Learning Environment (Bellemare et al., 2013), see for example (Mnih et al., 2015; Hessel et al., 2018). Atari games were popular video games in the 1980s in game arcades. Figure 14 shows a screenshot of Q*bert, a typical Atari arcade game.

Deep learning methods are well suited to process high-dimensional inputs. The challenge for model-based methods is to learn accurate models with a low number of observations.

Latent model approaches have been tried for Atari with some success (Oh et al., 2015, 2017; Kaiser et al., 2019; Hafner et al., 2021).

4.2.3 Outcome

We can now consider the question if finding methods to achieve high-accuracy models with low sample complexity has been solved. This is the central research question that many researchers have worked on. Unfortunately, authors have used different tasks within benchmark suites—or even different benchmarks—making comparisons between different publications challenging. A study by Wang et al. (2019) reimplemented many methods for continuous problems to perform a fair comparison. They found that ensemble methods and model-predictive control indeed achieve good results on MuJoCo tasks, and do so in significantly fewer time steps than model-free methods. Typcially model-based used 200k time steps versus 1 million for model-free. However, they also found that although the sample complexity is less, the wall-clock time may be different, with model-free methods such as PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018a,b) being much faster for some problems. The accuracy for the policy varies greatly for different problems, as does the sensitivity to different hyperparameter values; i.e., results are brittle.

Taking these caveats into consideration, we conclude that the papers that we survey report that, for high-dimensional problems, model-based methods do indeed approach an accuracy as high as model-free baselines, with substantially fewer environment samples. Therefore we conclude that the methods that we survey overcome the difficulties posed by overfitting, although model-based methods may sometimes be computationally inefficient.

4.3 Generalization Performance

Although in some important applications high accuracy at lower sample complexity has been achieved, and although quite a few results are impressive, challenges remain. To start, the algorithms that have been developed are quite complex. Latent models, end-to-end learning and planning, and probabilistic inference are complex algorithms, that require effort to understand, and even more to implement correctly in new applications.

Nevertheless, generalization beyond a single application has been achieved by model-based approaches, in non-trivial high-dimensional applications. The learned transition models generalize, they are not just used in single problems only. Already some results are reported where they are used in a transfer learning or meta learning setting (Brazdil et al., 2022; Hospedales et al., 2020; Huisman et al., 2021), and also in Sekar et al. (2020).

Indeed, we have seen that new classes of applications have become possible, both in continuous action spaces—learning to perform complex robotic behaviors. In discrete action spaces the MuZero approach was even able to learn the rules of very different games: Atari, Go, shogi (Japanese chess) and chess (Schrittwieser et al., 2020).

Model-based approaches are being used to learn causal models (Schölkopf et al., 2021; Sauter et al., 2021). Efforts are under way to improve the efficiency of MuZero (Ye et al., 2021) and the MuZero approach is being applied to video compression on YouTube (Mandhane et al., 2022). Latent model approaches are also related to other fields, such as coarse graining methods and abstraction in organisms (Itzkovitz et al., 2005; Flack, 2017), and model-based approaches connect reinforcement learning to explainable AI (Heuillet et al., 2021).

Despite these encouraging results, reproducibility remains a challenge due to the use of different benchmarks. Also, high sensitivity to differences in hyperparameter values leads

to brittleness in results, making reprodicibility difficult. Finally, we note that continuous problems that are solved appear to be of lower dimensionality than some discrete problems.

4.4 Research Agenda

The deep model-based approaches achieve good results in some applications, showing a promise for future work to extend these approaches. Based on the results that have been achieved, and the challenges that remain, we come to the following research agenda.

We note that different approaches were developed for different applications. Reproducibility of results is a challenge, different hyperparameters can have a large influence on performance. Furthermore, there are many different benchmarks in use in the field, and the complexity of some continuous benchmarks is less than for discrete. Reproducibility and benchmarking are the *first* item on our research agenda.

We also note that end-to-end learning and planning is a complex approach, that does not work on all applications, and that requires a large computational effort. Applying end-to-end to large problems is still a challenge. The *second* item on our research agenda is to develop end-to-end learning and planning further, to find more efficient end-to-end algorithms, to be able to apply them to more and different applications.

We further note that latent models and world models are also complex. various approaches use different types of modules, some consist of submodules. The *third* item on our research agenda is to integrate latent models with end-to-end learning and planning. As *fourth* item, we would like to simplify latent models and to standardize them, if possible for different applications, and apply latent models to higher-dimensional continuous problems.

Finally, we would like to enter on our research agenda meta and transfer learning experiments for model-based reinforcement learning, as *fifth* item, to increase generalization, also to causal modeling, and to more applications and fields.

In summary, to improve the accuracy and applicability of model-based methods we suggest to work on the following:

1. Improve reproducibility of model-based reinforcement learning, standardize benchmarking, and improve robustness (hyperparameters)
2. Improve efficiency of integrated end-to-end learning and planning; improve applicability to more and larger applications
3. Integrate latent models and end-to-end learning and planning
4. Simplify the latent model architecture across different applications, and apply latent models to higher-dimensional continuous problems
5. Use model-based reinforcement learning transition models for generalization in meta and transfer learning, causal modeling, XAI, coarse graining, and connect to other applications and fields, such as biology

5 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 have come within reach.

Model-based methods offer the advantage of lower sample complexity than model-free methods, because agents learn their own transition model of the environment. However, traditional methods to reduce bias and variance, such as Gaussian processes, that work

well on moderately complex problems with few samples, do not perform well on high-dimensional problems, due to their computational complexity. High-capacity models may have high sample complexity to create high-accuracy models, and finding methods that generalize well with low sample complexity has been difficult.

In the last five years new methods have been devised, and great success has been achieved in model-based deep reinforcement learning. This survey summarizes the main ideas of recent papers in a taxonomy based on learning and planning. Latent models condense complex problems into compact abstract representations that are easier to learn and plan; limited horizon planning reduces the impact of low-accuracy models; end-to-end methods have been devised to integrate learning and planning in one fully differentiable approach that match the look-ahead depth of planning and learning.

The Arcade Learning Environment has been one of the main benchmarks in model-free reinforcement learning, starting off the recent interest in the field with the work by Mnih et al. (2013, 2015). The high-dimensionality of Atari video input has long been problematic for model-based reinforcement learning. Latent models were instrumental in reducing the dimensionality of Atari, producing the first successes for model-based approaches on this major benchmark.

Model-based methods build transition models that generalize better to new applications, and are used in meta learning and transfer learning. These successes generate interest in related fields such as causal modeling, XAI, abstraction and coarse graining in fields such as biology.

Despite this success, limitations of current approaches remain. In the discussion we mentioned open problems for each of the approaches, where we expect worthwhile future work to occur. Impressive results have been reported; future work can be expected in transfer learning with latent models, and the interplay of 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 high-dimensional games and complicated camera-arm 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, competition 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 (Bellemare et al., 2013; Brockman et al., 2016; Vinyals et al., 2017; Tassa et al., 2018) and a quick turnaround of ideas. We hope that this survey will contribute to the low barrier of entry.

Acknowledgments

We thank the members of the Leiden Reinforcement Learning Group, and especially Thomas Moerland, Mike Huisman, Matthias Müller-Brockhausen, Zhao Yang, Erman Acar, and Andreas Sauter for many discussions and insights. We thank the anonymous reviewers for their valuable insights, which improved the paper greatly.

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.

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.

Dimitri P Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. MIT Press Cambridge, 1996.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Information science and statistics. Springer Verlag, Heidelberg, 2006.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of Statistics*, volume 31, pages 35–59. Elsevier, 2013.

Pavel Brazdil, Jan van Rijn, Carlos Soares, and Joaquin Vanschoren. *Metalearning: Applications to Automated Machine Learning and data mining*. Springer, 2022.

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 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, Théophane Weber, Sébastien Racaniere, SM Ali Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frédéric Besse, Karol Gregor, Demis Hassabis, and Daan Wierstra. 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. Share and play new sokoban levels. `http://Sokoban.org`, 2013.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *International Conference on Learning Representations*, 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. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland*, pages 617–629, 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.

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

Thomas G Dietterich. The MAXQ method for hierarchical reinforcement learning. In *International Conference on Machine Learning*, volume 98, pages 118–126, 1998.

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.

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. In *International Conference on Learning Representations*, 2018.

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.

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. IEEE, 2017.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

Jessica C Flack. Coarse-graining as a downward causation mechanism. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2109):20160338, 2017.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

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.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

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, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy P. Lillicrap. An investigation of model-free planning. In *International Conference on Machine Learning*, pages 2464–2473, 2019.

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. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

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.

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.

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. In *AAAI*, pages 3215–3222, 2018.

Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

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 N. van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, (54):4483–4541, 2021.

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.

Shalev Itzkovitz, Reuven Levitt, Nadav Kashtan, Ron Milo, Michael Itzkovitz, and Uri Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical Review E*, 71(1): 016127, 2005.

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.

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, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for Atari. *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.

Reza Kamyar and Ehsan Taheri. Aircraft optimal terrain/threat-based trajectory planning and control. *Journal of Guidance, Control, and Dynamics*, 37(2):466–483, 2014.

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.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *Found. Trends Mach. Learn.*, 12(4):307–392, 2019.

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.

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

W Hi Kwon, AM Bruckstein, and T Kailath. Stabilizing state-feedback design via the moving horizon method. *International Journal of Control*, 37(3):631–643, 1983.

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.

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

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.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

Amol Mandhane, Anton Zhernov, Maribeth Rauh, Chenjie Gu, Miaosen Wang, Flora Xue, Wendy Shang, Derek Pang, Rene Claus, Ching-Han Chiang, et al. Muzero with self-competition for rate control in vp9 video compression. *arXiv preprint arXiv:2202.06626*, 2022.

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 A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 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.

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. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2018.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

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.

Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(4): 293–311, 2013.

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.

Aske Plaat. *Learning to Play: Reinforcement Learning and Games*. Springer Verlag, Heidelberg, `https://learningtoplay.net`, 2020.

Aske Plaat. *Deep Reinforcement Learning*. Springer Verlag, Singapore, `https://deep-reinforcement-learning.net`, 2022.

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.

Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter W. Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5690–5701, 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.

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

Andreas Sauter, Erman Acar, and Vincent François-Lavet. A meta-reinforcement learning algorithm for causal discovery. 2021.

Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

Daniel Schleich, Tobias Klamt, and Sven Behnke. Value iteration networks on multiple levels of abstraction. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany*, 2019.

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. Technical report, Inst. für Informatik, 1990b.

Jeff Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. *Advances in neural information processing systems*, 9, 1996.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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. In *International Conference on Machine Learning*, 2020.

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

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

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, and Thomas Degris. 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. In *International Conference on Machine Learning*, pages 4739–4748, 2018.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 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 *Advances 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, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

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.

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.

Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso,

David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A new challenge for reinforcement learning. *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, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

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. *arXiv:1907.02057*, 2019.

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

Annie Wong, Thomas Bäck, Anna V. Kononova, and Aske Plaat. Deep multiagent reinforcement learning: Challenges and directions. *Artificial Intelligence Review*, 2022.

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.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34, 2021.

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.