Model-based Reinforcement Learning: A Survey.

Thomas M. Moerland[1,2], Joost Broekens[2], and Catholijn M. Jonker[1,2]
[1] Interactive Intelligence, TU Delft, The Netherlands
[2] LIACS, Leiden University, The Netherlands

Contents

Abstract

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

Keywords: Model-based reinforcement learning, reinforcement learning, planning, search, Markov Decision Process, review, survey.

1 Introduction

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

While model-based RL has shown great success (Silver et al., 2017a; Levine and Koltun, 2013; Deisenroth and Rasmussen, 2011), literature lacks a systematic review of the field (although Hamrick et al. (2020) does provide an overview of mental simulation in deep learning, see Sec. 8 for a detailed discussion of related work). Therefore, this article presents a systematic survey of the combination of planning and learning. It consist of four key sections. We first cover approaches to *dynamics model learning*, including important challenges like stochasticity, uncertainty, partial observability, non-stationarity, state abstraction, and temporal abstraction (Sec. 4). Then, we cover the integration of planning and learning, i.e., the ways we may use a (learned) dynamics model to solve for a (learned) policy (Sec. 5). Afterwards, Section 6 covers the implicit approach to model-based RL, as opposed to the explicit approaches of Sections 4 and 5. Finally, Sec. 7 covers the potential benefits of model-based reinforcement learning, such as data efficiency, targeted exploration, stability, transfer, safety and explainability.

Model-based RL is a fundamental approach to sequential decision making, and many other sub-disciplines in RL have a close connection to model-based RL. For example, *hierarchical reinforcement learning* (Barto and Mahadevan, 2003) can be approached in a model-free and model-based way. In the latter case, the higher-level action space defines a

model with temporal abstraction. Model-based RL is also an important approach to *transfer learning* (Taylor and Stone, 2009) (through model transfer between tasks) and *targeted exploration* (Thrun, 1992). When applicable, the survey also presents short overviews of such related RL research directions.

The remainder of this article is organized as follows. We first present a formal introduction of the MDP optimization problem (Sec. 2) and identify the categories of model-based RL (Sec. 3). Then, we present the main content of the survey, on model learning (Sec. 4), planning-learning integration (Sec. 5), implicit model-based RL (Sec. 6), and the benefits of model-based RL (Sec. 7). At the end of the survey, we also present Related Work (Sec. 8), Discussion (Sec. 9), and Summary (Sec. 10) sections.

2 Background

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

The agent acts in the environment according to a *policy* $\pi : \mathcal{S} \to p(\mathcal{A})$. In the search community, a policy is also known as a *contingency plan* or *strategy* (Russell and Norvig, 2016). By repeatedly selecting actions and transitioning to a next state, we can sample a *trace* through the environment. The *cumulative return* of a trace through the environment is denoted by: $J_t = \sum_{k=0}^{K} \gamma^k \cdot r_{t+k}$, for a trace of length K. For $K = \infty$ we call this the infinite-horizon return.

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

$$Q^\pi(s, a) \doteq \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{K} \gamma^k r_{t+k} \Big| s_t = s, a_t = a \right] \tag{1}$$

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

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)} \left[\mathcal{R}(s, a, s') + \gamma \, \mathbb{E}_{a' \sim \pi(\cdot|s')} \Big[Q^\pi(s', a') \Big] \right] \tag{2}$$

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

$$\pi^\star = \arg\max_\pi Q^\pi(s, a) = \arg\max_\pi \mathbb{E}_{\pi,\mathcal{T}} \left[\sum_{k=0}^{K} \gamma^k r_{t+k} \Big| s_t = s, a_t = a \right] \tag{3}$$

There is *at least* one optimal policy, denoted by $\pi^\star$, which is better or equal than all other policies (Sutton and Barto, 2018). In the planning and search literature, the above problem is typically formulated as a cost *minimization* problem (Russell and Norvig, 2016), instead of a reward maximization problem. That formulation is interchangeable with our presentation by negating the reward function.

3 Categories of Model-based Reinforcement Learning

The original difference between planning and learning was the way they can access the MDP dynamics. Planning has *reversible* access to the MDP, which allows it to repeatedly plan

Table 1: Categories of planning-learning integration. The top two rows show the two separate research fields of model-free RL and planning. The table shows three forms of explicit integration of planning and learning, depending on whether the model is learned and/or whether a global value or policy is learned.

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

forward from the same state (similar to the way humans plan in their mind). Reinforcement learning originally got *irreversible* access to the environment, and has to move forward after an action is executed (similar to the way we act in the real world). However, the difference in access actually led both fields to use a different type of representation of the solution. Planning typically uses *local*, *tabular/atomic* solutions which focus on one state or a subset of states. Reinforcement learning uses *global*, *learned* solutions which focus on the entire state space. We will follow the second property here to delineate pure planning from reinforcement learning.

Departing from this fundamental difference, both research fields have started to develop their own methodology. Along the way, they have also met, in a field that became known as *model-based reinforcement learning* (Sutton, 1990). Note that the principles beneath model-based RL were described around the same time in the search community, in the form of Learning Real-Time A* (Korf, 1990), and the underlying principles at least date back to the Checkers programme by Samuel (1967).

We define model-based RL as: 'any MDP approach that uses i) a model (known or learned), i.e., reversible access to the MDP dynamics, and ii) learning to approximate a global value or policy function'. Note that, in the context of planning and learning integration, learning is actually an overloaded term, since it may happen 1) to approximate a dynamics model, and 2) to approximate a value or policy function. This leads to the following three categories of explicit planning-learning integration (summarized in Table 1):

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

Note that the last category is not considered model-based RL, since it does not learn a global solution to the problem. However, it is a form of planning-learning integration (and some researchers may actually consider it model-based RL), and we therefore will include this topic in the survey. Also, note that the line between replay databases (Lin, 1992) and model-based RL with a learned tabular model is very blurry (Vanseijen and Sutton, 2015; van Hasselt et al., 2019), which of course also applies to the relation with episodic memory (Pritzel et al., 2017).

It is important to distinguish the above categories, because they need to cope with different challenges. For example, approaches with a learned dynamics model typically need to account for uncertainty, while approaches with a known/given dynamics model can

ignore this issue, an put stronger emphasis on asymptotic performance. We will extensively encounter these categories in Sec. 5 on planning-learning integration.

4 Dynamics Model Learning

The first step of model-based RL usually involves learning the dynamics model from observed data. In the control literature, dynamics model learning is better known as *system identification* (Åström and Eykhoff, 1971; Ljung, 2001). We will first cover the general considerations of learning a one-step model (Sec. 4.1). Afterwards, we extensively cover the various challenges of model learning, and their possible solutions. These challenges are stochasticity (Sec. 4.2), uncertainty due to limited data (Sec. 4.3), partial observability (Sec. 4.4), non-stationarity (Sec. 4.5), multi-step prediction (4.6), state abstraction (Sec. 4.7) and temporal abstraction (Sec. 4.8). The reader may wish to skip some of these section if the particular challenge is not relevant to your research problem or task of interest.

4.1 Basic considerations

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

Type of model The first question is: what do we actually consider to be a model? We will here focus on dynamics models. A model of the reward function can usually be easily added by predicting an additional scalar. Given a batch of one-step transition data $\{s_t, a_t, r_t, s_{t+1}\}$, there are three main types of dynamics function we might be interested in:

- *Forward model*: $(s_t, a_t) \rightarrow s_{t+1}$. This predicts the next state given a current state and chosen action. It is by far the most common type of model, and can be used for lookahead planning.
- *Backward/reverse model*: $s_{t+1} \rightarrow (s_t, a_t)$. This model predicts which states are the possible precursors of a particular state. Thereby, we can plan in the backwards direction, which is for example used in prioritized sweeping (Moore and Atkeson, 1993).
- *Inverse model*: $(s_t, s_{t+1}) \rightarrow a_t$. An inverse model predicts which action is needed to get from one state to another. It is for example used in RRT planning (LaValle, 1998). As we will later see, this function can also be useful as part of representation learning (Sec. 4.7).

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

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

- *Parametric*: Parametric methods are the most popular approach for model approximation. Compared to non-parametric methods, a benefit of parametric methods is that their number of parameters is independent of the size of the observed dataset. There are two main subgroups:
  - *Exact*: A cardinal distinction in learning is between exact/tabular and approximate methods. For a discrete MDP (or a discretized version of a continuous MDP), a tabular method maintains a separate entry for every possible transition.

For example, in a stochastic MDP (in which we need to learn a probability distribution, see next section) a *tabular maximum likelihood model* (Sutton, 1991) estimates the probability of each possible transition as

$$T(s'|s,a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}, \tag{4}$$

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

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

- *Non-parametric*: The other main supervised learning approach is non-parametric approximation. The main property of non-parametric methods is that they directly store and use the data to represent the model.
  - *Exact*: Replay buffers (Lin, 1992) can actually be regarded as non-parametric versions of tabular methods. While a table has parameters (all table entries) and thereby a fixed size determined by the MDP dynamics, a replay buffer can theoretically continue to store all data, although we of course cap the size in practice.
  - *Approximate*: We may also apply non-parametric methods when we want to be able to generalize information to similar states. For example, Gaussian processes (Wang et al., 2006; Deisenroth and Rasmussen, 2011) have been a popular non-parametric approach. Gaussian processes can also provide good uncertainty estimates, which we will further discuss in Sec. 4.3.

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

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

Region in which the model is valid The third important consideration is the region of state space in which we aim to make the model valid:

- *Global*: These models approximate the dynamics over the entire state space. This is the main approach of most model learning methods. It can be challenging to generalize

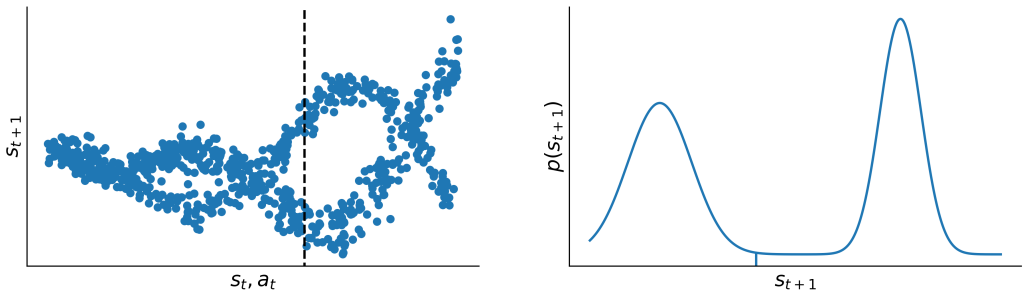

Figure 1: Illustration of stochastic transition dynamics. **Left**: 500 samples from an example transition function $\mathcal{T}(s'|s,a)$. The vertical dashed line indicates the cross-section distribution on the right. **Right**: distribution of s_{t+1} for a particular s, a. We observe a multimodal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

well over the entire state space, but it is the main way to store all information from previous observations.

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

The distinction between global and local is equally relevant for representation of a value or policy function, as we will see in Sections 5 and 7.

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

4.2 Stochasticity

In a stochastic MDP the transition function specifies a distribution over the possible next states, instead of returning a single next state (Figure 1, left). In those cases, we should also specify a model that can approximate entire distributions. Otherwise, when we for example train a deterministic neural network $f_\phi(s,a)$ on a mean-squared error loss (e.g., Oh et al. (2015)), then the network will actually learn to predict the conditional mean of the next state distribution (Moerland et al., 2017b). This problem is illustrated in Figure 1, right.

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

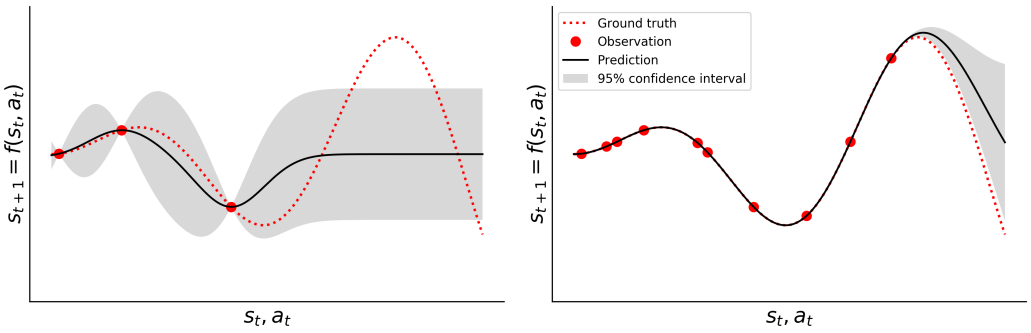

Figure 2: Illustration of uncertainty due to limited data. Red dotted line depicts an example ground truth transition function. **Left**: Gaussian Process fit after 3 observations. The predictions are clearly off in the right part of the figure, due to wrong extrapolation. The shaded area shows the 95% confidence interval, which does identify the remaining uncertainty, although not completely correct. **Right**: Gaussian Process fit after 10 observations. Predictions are much more certain now, mostly matching the true function. There is some remaining uncertainty on the far right of the curve.

In high-dimensional problems, there has been much recent effort on generative models based on neural network approximation (deep generative models). One approach is to use variational inference (VI) to estimate dynamics models (Depeweg et al., 2016; Moerland et al., 2017b; Babaeizadeh et al., 2017; Buesing et al., 2018). Competing approach include generative adversarial networks (GANs), autoregressive full-likelihood models, and flow-based density models, which were applied to sequence modeling by Yu et al. (2017), Kalchbrenner et al. (2017) and Ziegler and Rush (2019), respectively. Detailed discussion of these methods falls outside the scope of this survey, but there is no clear consensus yet which deep generative modeling approach works best.

4.3 Uncertainty

A crucial challenge of model-based learning is dealing with uncertainty due to limited data. Uncertainty due to limited data (also known as *epistemic* uncertainty) clearly differs from the previously discussed stochasticity (also known as *aleatoric* uncertainty) (Der Kiureghian and Ditlevsen, 2009), in the sense that uncertainty can be reduced by observing more data, while stochasticity can never be reduced. We clearly want to be able to estimate the remaining uncertainty in our model estimate, to assess whether our plan is actually reliable. Uncertainty is even relevant in the absence of stochasticity, as illustrated in Figure 2.

We therefore want to estimate the uncertainty around our predictions. Then, when we plan over our model, we can detect when our predictions become less trustworthy. There are two principled approaches to uncertainty estimation in statistics: frequentist and Bayesian. A frequentist approach is for example the statistical bootstrap, applied to model estimation by Fröhlich et al. (2014) and Chua et al. (2018). Bayesian RL methods were previously surveyed by Ghavamzadeh et al. (2015). Especially successful have been non-parametric Bayesian methods like Gaussian Processes (GPs), for example used for model estimation in PILCO (Deisenroth and Rasmussen, 2011). However, GPs scale (computationally) poorly to high-dimensional state spaces. Therefore, there has been much recent interest in Bayesian methods for neural network approximation of dynamics, for example based on variational dropout (Gal et al., 2016) and variational inference (Depeweg et al., 2016). Note that uncertainty estimation is also an active research topic in the deep learning community itself, and advances in those fields will likely benefit model-based RL as well. While this section discussed uncertainty estimation, we will discuss how to deal with model uncertainty during

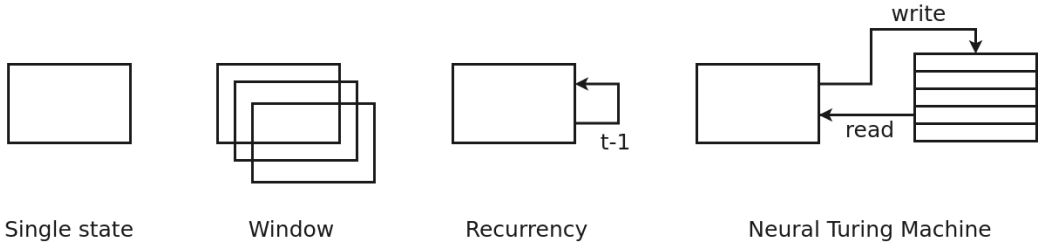

Figure 3: Example approaches to partial observability. The window approach concatenates the most recent n frames and treats this as a new state. The recurrent approach learns a recurrent mapping between timesteps to propagate information. The Neural Turing Machine uses an external memory to explicitly write away information and read it back when relevant, which is especially applicable to long-range dependencies.

planning in Sec. 5.

4.4 Partial observability

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

So how do we incorporate information from previous observations? There are four main approaches: i) windowing, ii) belief states, iii) recurrency and iv) external memory (Figure 3).

- *Windowing*: In the windowing approach we concatenate the n most recent observations and treat these together as the state (Lin and Mitchell, 1992). McCallum (1997) extensively studies how to adaptively adjust the window size. In some sense, this is the tabular solution to partial observability. Although effective in small problems, there are several important limitations. First of all, the size of the model grows linearly in n (the history length), which makes them less applicable in high-dimensional problems or with large n. More importantly, they do not generalize at all between similar histories, which makes it hard to apply them in high-dimensional problems as well (where we seldomly encounter exactly the same history twice).
- *Belief states*: Belief states explicitly partition the learned dynamics model in an *observation model* $p(o|s)$ and a *latent transition model* $\mathcal{T}(s'|s,a)$ (Chrisman, 1992). This structure reminds of the sequence modeling approach of *state-space models* (Bishop, 2006), like hidden Markov models (HMM). Estimation of model parameters is usually based on expectation-maximization (EM) schemes (Ghahramani and Hinton, 1996). There are also specific planning methods for belief state models, known as POMDP planners (Spaan and Spaan, 2004; Kurniawati et al., 2008; Silver and Veness, 2010). However, belief state models usually require prior knowledge on the belief state structure, and have trouble scaling to high-dimensional problems (since the expectation step becomes intractable).
- *Recurrency*: The most popular solution to partial observability is probably the use of *recurrent* neural networks, first applied to dynamics learning in Lin (1993); Parlos

et al. (1994). A variety of papers have studied RNNs in high-dimensional settings in recent years (Chiappa et al., 2017; Ha and Schmidhuber, 2018; Gemici et al., 2017). Since the transition parameters of the RNN are shared between all timesteps, the model size is independent of the history length, which is one the main benefits of RNNs. They also neatly integrate with gradient-based training and high-dimensional state spaces. However, they do suffer from vanishing and exploding gradients to model long-range dependencies. This may be partly mitigated by long short-term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) or temporal skip connections (El Hihi and Bengio, 1996). Beck et al. (2020) recent proposed *aggregators*, which are more robust to long-range stochasticity in the observed sequences, as frequently present in RL tasks.

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

Partial observability is an inherent property of nearly all real-world tasks. When we ignore partial observability, our solution may completely fail. Therefore, many research papers that actually focus on some other question, still need to incorporate methodology to battle the partial observability in the domain. Finally, note that the above partial observability methodology is equally applicable to a learned policy or value function.

4.5 Non-stationarity

Non-stationarity in an MDP occurs when the true transition and/or reward function change(s) over time. When the agent keeps trusting its previous model, without detecting the change, then its performance may deteriorate fast. Figure 4 illustrates the problem.

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

Transfer learning (Taylor and Stone, 2009) and meta-learning (Li et al., 2017) can actually be seen as special cases of non-stationarity, since we deal with optimization over a sequence of tasks. For example, Fu et al. (2016) aim to learn a generic neural network prior, which can be quickly adapted to new tasks. We further discuss transfer learning in Sec. 7.4.

4.6 Multi-step Prediction

After learning a model we intend to plan over it, which usually involves a multi-step look-ahead. The models we discussed so far made 1-step predictions of the next state. We can make multi-step predictions with such models by repeatedly feeding the prediction back into the learned model. However, since our learned model was never optimized to make long range predictions, accumulating errors may actually cause our multi-step predictions

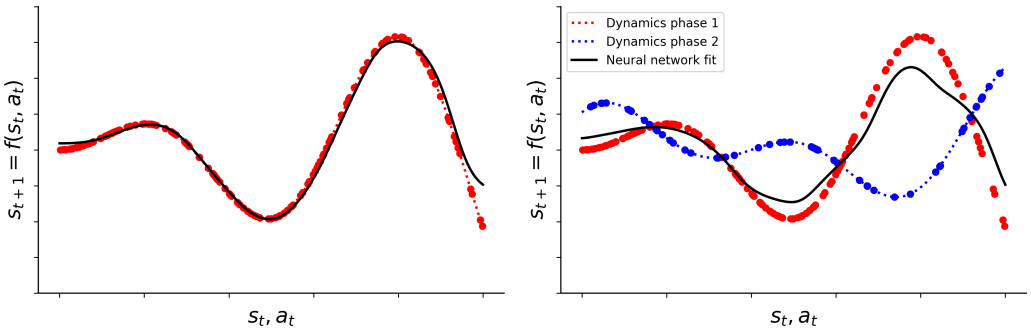

Figure 4: Illustration of non-stationarity. **Left**: First 150 data points sampled from initial dynamics. Black line shows the prediction of a neural network with 2 hidden layers of 50 units and tanh activations trained for 150 epochs. **Right**: Due to non-stationarity the dynamics changed to the blue curve, from which we sample an additional 50 points. The black curve shows the new neural network fit without detection of the dynamics change, i.e., treating all data as valid samples from the same transition distribution. We clearly see the network has trouble adapting to the new regime, as it still tries to fit to the old dynamics data points as well.

to diverge from the true dynamics. Several authors have identified this problem (Talvitie, 2014; Venkatraman et al., 2015; Talvitie, 2017; Machado et al., 2018).

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

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

4.7 State abstraction

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

Early application of representation learning in RL include (soft) state aggregation (Singh et al., 1995) and principal component analysis (PCA) (Nouri and Littman, 2010). Mahadevan (2009) covers various approaches to learning basis functions in Markov Decision Processes. However, by far the most successful approach to representation learning in re-

cent years have been deep neural networks, with a variety of example applications to model learning (Oh et al., 2015; Watter et al., 2015; Chiappa et al., 2017).

A (deep) neural network dynamics model is typically factorized in three parts: i) an *encoding* function $z_t = f_\phi^{\text{enc}}(s_t)$, which maps the observation to a latent representation z_t, ii) a *latent dynamics* function $z_{t+1} = f_\phi^{\text{trans}}(z_t, a_t)$, which transitions to the next latent state based on the chosen action, and iii) a *decoder* function $s_{t+1} = f_\phi^{\text{dec}}(z_{t+1})$, which maps the latent state back to the next state prediction. This structure, visualized in Figure 6 (item 4), reminds of an auto-encoder (with added latent dynamics), as frequently used for representation learning in the deep learning community.

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

Planning at a latent level We ideally want to be able to plan at a latent level. Since the representation space is usually smaller than the observation space, this may save much computational effort. However, we must ensure that the predicted next latent state lives in the same embedding space as the encoded current latent state. Otherwise, repeatedly feeding the latent prediction into the latent dynamics model will lead to predictions that diverge from the truth. One approach is to add an additional loss that enforces the next state prediction to be close to the encoding of the true next state (Watter et al., 2015). An alternative are deep state-space models, like deep Kalman filters (Krishnan et al., 2015) or deep variational Bayes filters (Karl et al., 2016). These require probabilistic inference of the latent space, but do automatically allow for latent level planning.

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

State abstraction is also related to *grey-box* system identification. In system identification (Åström and Eykhoff, 1971), the control term for model learning, we may discriminate 'black box' and 'grey box' approaches (Ljung, 2001). Black box methods, which do not assume any task-specific knowledge in their learning approach, are the main topic of Section 4. Grey box methods do partially embed task-specific knowledge in the model, and estimate remaining free parameters from data. The prior knowledge of grey box models is usually derived from the rules of physics. One may use the same idea to learn state abstractions. For example, in a robots task with visual observations, we may known the required (latent) transition model (i.e., f_ϕ^{trans} is known from physics), but not the encoding function from the visual observations (f_ϕ^{enc} is unknown). Wu et al. (2015) give an example of this approach, where the latent level dynamics are given by a known, differentiable physics engine, and we optimize for the encoding function from image observations.

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

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

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

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

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

Finally, there is an additional way to improve representations through *value equivalent models* (Grimm et al., 2020). These models are trained on their ability to predict a value or (optimal) action. We decide to cover this idea in Sec. 6 on implicit model-based RL, which covers methods that optimize elements of the model-based RL process for the ability to output an (optimal) action or value. In short, this section discussed the several ways in which the state representation learning of models may be improved, for example by embedding specific substructure in the networks (e.g., to extract objects and their interactions), or by constructing smarter loss functions.

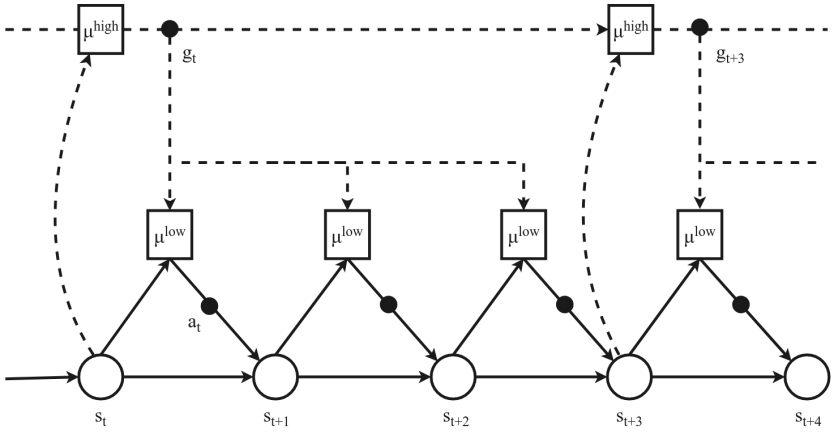

Figure 5: Conceptual illustration of a two-level hierarchy, partially based on Nachum et al. (2018). Standard low-level interaction is shown with solid lines, temporal abstraction is shown with dashed lines. The high-level controller picks a high-level action (goal) g_t according to π^{high}. After fixing g_t, the low level controller executes the relevant subpolicy, for example in the form of a goal-conditioned policy $\pi^{\text{low}}(s, g)$. The number of steps between high-level actions can be fixed or variable, depending on the framework. The illustration assumes full observability, in which case we only need to condition π^{high} on the current observation. We may also feed g back into the next high-level decision to enable temporal correlation between goals.

4.8 Temporal abstraction

The MDP definition typically involves low-level, atomic actions executed at a high-frequency. This generates deep search trees with long-range credit assignment. However, many of these paths give the same end-state, and some end-states are more useful than others. The idea of temporal abstraction, better known as *hierarchical* reinforcement learning (Barto and Mahadevan, 2003; Hengst, 2017; Thrun and Schwartz, 1995), is to identify a high-level action space that extends over multiple timesteps (Figure 5). Temporal abstraction may reduce both the sample (Brunskill and Li, 2014) and computational complexity (Mann and Mannor, 2014) of solving the MDP.

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

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

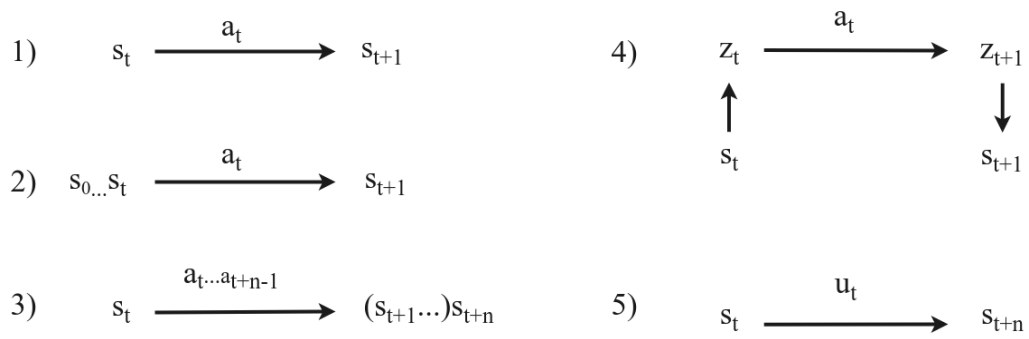

Figure 6: Overview of different types of mappings in model learning. **1)** Standard Markovian transition model $s_t, a_t \rightarrow s_{t+1}$. **2)** Partial observability (Section 4.4). We model $s_0...s_t, a_t \rightarrow s_{t+1}$, leveraging the state history to make an accurate prediction. **3)** Multi-step prediction (Section 4.6), where we model $s_t, a_t...a_{t+n-1} \rightarrow s_{t+n}$, to predict the n step effect of a sequence of actions. **4)** State abstraction (Section 4.7), where we compress the state into a compact representation z_t and model the transition in this latent space. **5)** Temporal/action abstraction (Section 4.8), better known as hierarchical reinforcement learning, where we learn an abstract action u_t that brings us to s_{t+n}. Temporal abstraction abstraction directly implies multi-step prediction, as otherwise the abstract action u_t is equal to the low level action a_t. All the above ideas (2-5) are orthogonal and can be combined.

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

- *Graph structure*: This approach identifies 'bottleneck' states as end-points for the sub-routines. A bottleneck is a state that connects two densely interconnected subgraphs in the MDP graph (Menache et al., 2002). Therefore, a bottleneck is a crucial state in order to reach another region of the MDP, and therefore a candidate subgoal. There are several ways to identify bottlenecks: McGovern and Barto (2001) identify bottlenecks from overlapping states in successful trials, Şimşek et al. (2005) run a graph partitioning algorithms on a reconstruction of the MDP graph, and Goel and Huber (2003) search for states with many predecessors, but whose successors do not have many predecessors. The bottleneck approach received much attention in smaller problems, but have received less attention in higher-dimensional problems.
- *State-space coverage*: Another idea is to spread the end-states of subroutines over the entire state-space, in order to reach good coverage. Most approaches first cluster the state space, and subsequently learn a dynamics model to move between the cluster centers (Mannor et al., 2004; Lakshminarayanan et al., 2016; Machado et al., 2017). Instead of the raw state space, we may also cluster in a compressed representation of it (Ghosh et al., 2018) (see previous section as well).
- *Compression (information-theoretic)*: We may also attempt to simply compress the space of possible end-points. This idea is close to the state space coverage ideas above. Gregor et al. (2016); Eysenbach et al. (2019); Achiam et al. (2018) associate the distribution of observed end-states with a noise distribution. After training, the noise distribution acts as a high-level action space from which we can sample. Various approaches also include additional information-theoretic regularization of this compression. For example, Gregor et al. (2016) add the criterion that action sequences in the compressed space should make the resulting state well predictable ('empower-

ment'). Other examples are provided by Florensa et al. (2017); Hausman et al. (2018); Fox et al. (2016).

- *Reward relevancy*: The idea of this approach is that relevant subroutines will help incur extra reward, and they should therefore automatically emerge from a black-box optimization approach. These approaches embed the structure of subroutines into their algorithms, ensure that the overall model is differentiable, and run an end-to-end optimization. Examples are the Option-Critic (Bacon et al., 2017; Riemer et al., 2018) and Feudal Networks (Vezhnevets et al., 2017), with more examples in Frans et al. (2018); Levy et al. (2019); Heess et al. (2016); Nachum et al. (2018). Daniel et al. (2016); Fox et al. (2017) use probabilistic inference based on expectation-maximization, where the E-step infers which options are active, and the M-step maximizes with respect to the value. A challenge for end-to-end approaches is ensuring diversity, i.e., preventing that a single subroutine starts to solve the entire task (or that every subroutine terminates after one step).
- *Priors:* Finally, we may also use prior knowledge to identify useful subroutines. Sometimes, the prior knowledge is domain-specific, like pre-training on hand-coded sub-tasks (Tessler et al., 2017; Heess et al., 2016). Kulkarni et al. (2016) identify all objects in the scene as end-points, which may generalize over domains when combined with a generic object recognizer. Several papers also infer relevant subroutines from expert demonstrations (Konidaris et al., 2012; Fox et al., 2017; Hamidi et al., 2015), which is of course also a form of prior knowledge.

This concludes our discussion of temporal abstraction, and of model learning as a whole. As a summary, Figure 6 present a conceptual overview of four of the challenges we discussed (partial observability, multi-step prediction, state abstraction and temporal abstraction), and the type of connectivity that they require. As we have seen, there are a variety of challenges and issues in model learning. In the next section, we will discuss how this learned model may actually be used to act and learn in the environment.

5 Integration of Planning and Learning

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

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

These dimensions each have several important subconsiderations. The overall framework is summarized in Table 2, and will be discussed in the following sections.

5.1 At which state to start planning?

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

Table 2: Overview of dimensions of planning-learning integration. These considerations are discussed throughout Sec. 5. Table 3 summarizes several model-based RL algorithms on these dimensions.

Dimension	**Consideration**	**Choices**
1. Start state (5.1)	- Start state	Random ↔ visited ↔ prioritized ↔ current
2. Budget (5.2)	- Number of real steps before planning	1 ↔ n, episode, etc.
	- Effort per planning cycle	1 ↔ n ↔ convergence
3. Planning approach (5.3)	- Type	Discrete ↔ gradient-based
	- Direction	Forward ↔ Backward
	- Breadth	1 ↔ adaptive ↔ full
	- Depth	1 ↔ interm./adaptive ↔ full
	- Uncertainty	Data-close ↔ Uncertainty propagation (-Prop.method: parametric ↔ sample)
4. Integration in learning loop (5.4)	- Planning input from learned function	Yes (value/policy) ↔ No
	- Planning output for training targets	Yes (value/Policy) ↔ No
	- Planning output for action selection	Yes ↔ No

- *Random*: A straightforward approach is to randomly select states throughout state space. This is for example the approach of Dynamic Programming (Bellman, 1966), which selects all possible states in a sweep. The major drawback of this approach is that it does not scale to high dimensional problems, since the total number of states grows exponentially in the dimensionality of the state space. The problem is that we will likely update many states that are not even reachable from the start state.
- *Visited*: We may ensure that we only plan at reachable states by selecting previously visited states as starting points. This approach is for example chosen by Dyna (Sutton, 1990).
- *Prioritized*: Sometimes, we may be able to obtain an ordering over the reachable states, identifying their relevancy for a next planning update. A good example is Prioritized Sweeping (Moore and Atkeson, 1993), which identifies states that likely need updating. Prioritization is also popular in replay database, but these are not considered model-based RL.
- *Current*: Finally, a common approach is to only spend planning effort at the current state of the real environment. This puts much emphasis at finding a better solution or more information in the region where we are currently operating. Even model-based RL methods with a known model, like AlphaGo Zero (Silver et al., 2017a), sometimes voluntarily introduce the notion of a real environment and current state. The real environment step introduces a form of pruning, as it ensures that we move forward at some point, obtaining information about deeper nodes (see Moerland et al. (2020a) as well).

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

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

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

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

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

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

Some approaches, especially the ones that target high data efficiency (see Sec. 7.1) in the real environment, allow for a very high planning budget once they start planning. These methods for example plan until convergence on an optimal policy (given the remaining uncertainty) (Deisenroth and Rasmussen, 2011). We call this a *squeezing* approach, since we attempt to squeeze as much information out of the available transition data as possible. We further discuss this approach in Sec. 7.1.

Adaptive trade-off Our choice on the above two dimensions essentially specifies a trade-off between planning and real data collection, with model-free RL (no planning effort) and exhaustive search (infinite planning effort) on both extremes. Most model-based RL approaches set the above two considerations to fixed (intermediate) values. However, humans clearly make a much more adaptive trade-off, where they adaptively decide a) when to start planning, and b) how much time to spend on that plan (i.e., the two considerations discussed above). This has indeed been an active topic of research in human psychology as well. Keramati et al. (2011) present this as a speed/accuracy trade-off, where habitual, reactive behaviour (no planning) is fast but potentially inaccurate, while extensive planning is slow but more accurate. See Hamrick (2019) for a more detailed discussion. We also return to this topic in Sec. 7.3.

A few authors have investigated an adaptive trade-off between planning and acting in model-based RL. Pascanu et al. (2017) add a small penalty for every planning step to the overall objective, which ensures that planning should provide reward benefit. This approach is very task specific. Hamrick et al. (2017) learn a meta-controller over tasks that learns to select the planning budget per timestep. In contrast to these optimization-based approaches, Kalweit and Boedecker (2017) derive the ratio between real and planned data from the variance of the estimated Q-function. When the variance of the Q-function is high, they sample additional data from the model. This ensures that they only use ground-truth data near convergence, but accept noisier model-based data in the beginning. Lu et al. (2019) propose a similar idea based on the epistemic uncertainty of the value function, by also increasing planning budgets when the uncertainty rises above a threshold. However, when we have a learned model, we probably do not want to plan too extensively in the beginning of training either (since the learned model is then almost random), so there are clear open research questions here.

5.3 How to plan?

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

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

- *Discrete planning*: This is the main approach in the classic AI and reinforcement learning communities, where we make discrete back-ups which are stored in a tree, table or used as training targets to improve a value or policy function. We can in principle use any preferred planning method. Examples in the context of model-based

RL include the use of probability-limited search (Lai, 2015), breadth-limited depth-limited search (François-Lavet et al., 2019), Monte Carlo search (Silver et al., 2008), Monte Carlo Tree Search (Silver et al., 2017a; Anthony et al., 2017; Jiang et al., 2018; Moerland et al., 2018b), minimax-search (Samuel, 1967; Baxter et al., 1999), or a simple one-step search (Sutton, 1990). These methods do not require any differentiability of the model.

- *Differential planning*: The gradient-based approach requires a differentiable model. If the transition and reward models are differentiable, and we specify a differentiable policy, then we can directly take the gradient of the cumulative reward objective with respect to the policy parameters. While a real world environment or simulator is by definition not differentiable, our learned model of these dynamics (for example a neural network) usually is differentiable. Therefore, model-based RL can suddenly utilize differential planning methods, exploiting the differentiability of the learned model. Note that differentiable models may also be obtained from the rules of physics, for example in differentiable physics engines (Degrave et al., 2019; de Avila Belbute-Peres et al., 2018).

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

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

Direction We also have to decide on the direction of planning (see also Sec. 4.1):

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

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

Breadth and depth A new planning iteration starts to lookahead from a certain start state. We then still need to decide on the the breadth and the depth of the lookahead. For

model-free RL approaches, breadth is not really a consideration, since we can only try a single action in a state (a breadth of one). However, a model is by definition reversible, and we are now free to choose and adaptively balance the breadth and depth of the plan. We will list the possible choices for both breadth and depth, which are summarized in Figure 7.

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

- *Breadth = 1*: These methods only sample single transitions or individual traces from the model, and still apply model-free updates to them. Therefore, they still use a breadth of one. The cardinal example of this approach is Dyna (Sutton, 1990), which sampled additional one-step data for model-free Q-learning (Watkins and Dayan, 1992) updates. More recently, Kalweit and Boedecker (2017) applied the above principle to deep deterministic policy gradient (DDPG) updates, Kurutach et al. (2018) to trust region policy optimization (TRPO) updates and Gu et al. (2016) to normalized advantage function (NAF) updates.
- *Breadth = adaptive*: Many planning methods adaptively scale the breadth of planning. The problem is of course that we cannot afford to go full breadth and full depth, because exhaustive search is computationally infeasible. A method that adaptively scales the breadth of the search is for example Monte Carlo Tree Search (Browne et al., 2012), by means of the upper confidence bounds formula. This ensures that we do go deeper in some arms, before going full wide at the levels above. This approach was for example applied in AlphaGo Zero (Silver et al., 2017a).
- *Breadth = full*: Finally, we may of course go full wide over the action space, before we consider searching on a level deeper. This is for example the approach of Dynamic Programming, which goes full wide with a depth of one. In the context of model-based RL, few methods have taken this approach.

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

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

This is of course a very shallow treatment of the crucial breadth versus depth balancing in planning, which has a close relation to exploration methods as well. However, the focus of this survey is the integration of planning and learning, not the actual planning method itself. From a model-based RL perspective, the crucial realization is that compared to model-free RL, we can suddenly use a breadth larger than one. Nevertheless, many model-based RL methods still choose to stick to a breadth of one in their model samples, likely because this gives seamless integration with model-free updates. We further discuss this topic in Sec. 7.1.

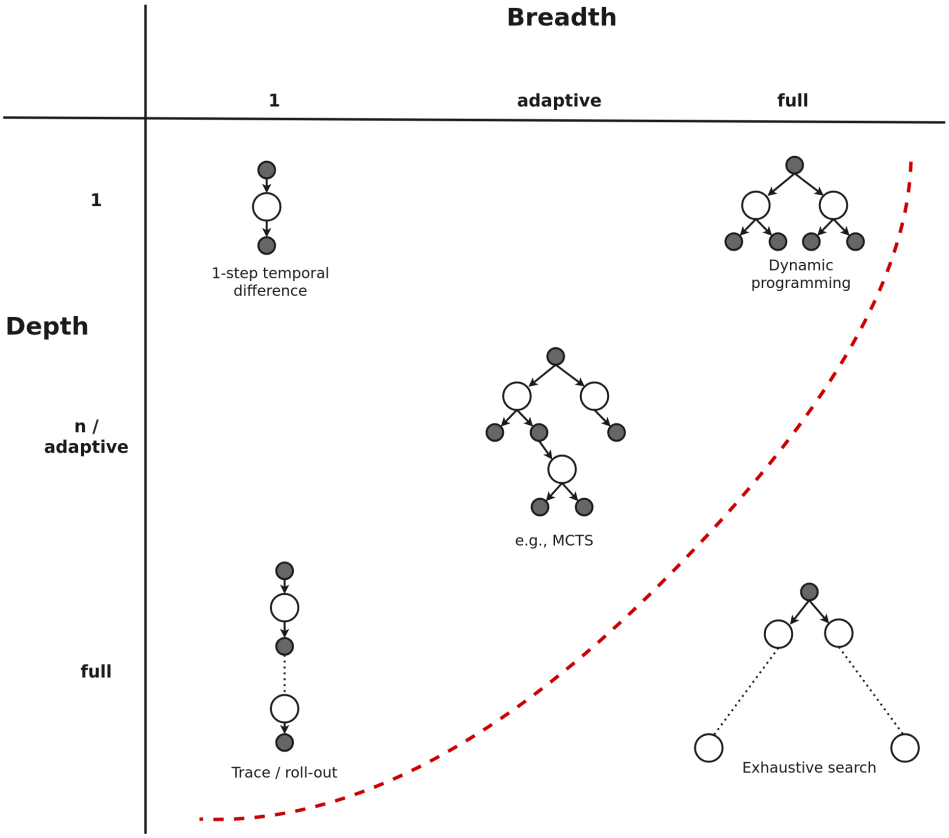

Figure 7: Breadth and depth of a single planning iteration. For every planning iteration, we need to decide on the breadth and depth of the lookahead. In practice, planning iterations usually reside somewhere left of the red dashed line, since we cannot afford a full breadth, full depth (exhaustive) search. Most planning methods, like MCTS, adaptively balance breadth and depth throughout the tree, where the breadth and depth differ throughout the tree. Figure is based on Sutton and Barto (2018), who used it to categorize different types of back-ups. A single planning iteration, which we define by fixing a new root state, can indeed be seen as a large back-up.

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

- *Data-close planning*: The first approach is to ensure that the planning iterations stay close to regions where we have actually observed data. For example, Dyna (Sutton, 1990) samples start states at the location of previously visited states, and only samples one-step transitions, which ensures that we do not depart from the known region of state space. Other approaches, like Guided Policy Search (Levine and Abbeel, 2014), explicitly constraint the new plan to be close to the current policy (which generated the data for the model).
- *Uncertainty propagation*: We may also explicitly estimate model uncertainty, which allows us to robustly plan over long horizons. Once we depart too far from the observed data, model uncertainty will increase, predictions will start to spread out over state space, and the learning signal will naturally vanish. Estimation of model uncertainty was already discussed in Sec. 4.3. We will here focus on propagation of uncertainty over timesteps, since the next state uncertainty is of course conditioned on the uncertainty

of the previous step. There are two main propagation approaches:

- *Parametric*: This propagation method fits a parametric distribution to the uncertainty at every timestep. This approach is for example used by PILCO (Deisenroth and Rasmussen, 2011), which derives closed form analytic expressions to track the uncertainty. However, analytic parametric propagation is not possible for more complicated models, like for example neural networks.
- *Sample-based*: This propagation approach, also known as *particle methods*, tracks the distributions of uncertainty by propagating a set of particles forward. The particles together represent the predicted distribution at a certain number of steps. Particle methods are for example used in Deep PILCO (Gal et al., 2016) and PETS (Chua et al., 2018). Note that fitting to a distribution, or matching moments of distributions, may have a regularizing effect. Therefore, Deep PILCO (Gal et al., 2016) does propagate particles through the dynamics function, but then refits these particles to a (Gaussian) distribution at every step. See Chua et al. (2018) for a broader discussion of uncertainty propagation approaches.

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

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

5.4 How to integrate planning in the learning and acting loop?

We have now specified how to plan (the start point, budget and planning method). However, we still need to integrate planning in the larger learning and acting loop. Figure 8 presents a conceptual overview of the overall training loop. We have so far focused on the planning box (arrow a), but we will now focus on the connection of planning to other aspects of the learning loop. These include: i) directing new planning iterations based on learned knowledge in value or policy functions (Fig. 8, arrow b), ii) using planning output to update learned value or policy functions (Fig. 8, arrow c), and iii) using planning output to select actions in the real world (Fig. 8, arrow d).

Planning input from learned functions The learned value or policy functions may store much information about the current environment, which may help to focus the next planning iteration. We distinguish the use of value and policy information:

- *Value priors*: The most common way to incorporate value information is through *bootstrapping* (Sutton and Barto, 2018), where we plug in the current prediction of a state or state-action value to prevent having to search deeper (reducing the depth of the search). Various model-based RL algorithm use bootstrapping in their planning approach, for example Baxter et al. (1999); Silver et al. (2017a); Jiang et al. (2018); Moerland et al. (2018b). Note that bootstrapping is also a very common principle in model-free RL. We may also use the learned value function to initialize the values of the action nodes at the root of the search (Silver et al., 2008; Hamrick et al., 2020), which we could interpret as a form of bootstrapping at depth 0.
- *Policy priors*: We can also leverage a learned policy in a new planning iteration. Several ideas have been proposed. AlphaGo Zero (Silver et al., 2017a) uses the probability

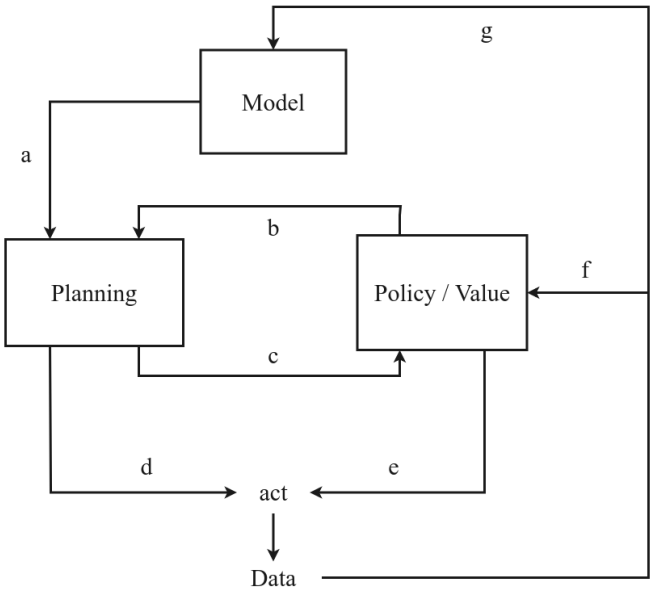

Figure 8: Procedural details of planning/learning integrations. Arrows (numbered a-g) indicate possible connections. a) plan over a learned model, b) use information from a policy/value network to improve the planning procedure, c) use the result from planning as training targets for a policy/value, d) act in the real world based on the planning outcome, e) act in the real world based on a policy/value function, f) generate training targets for the policy/value based on real world data, g) generate training targets for the model based on real world data. Most algorithms only implement a subset of these connections. See Figure 9 for an illustration of the subsets used by different algorithms.

of an action as a prior multiplication term on the upper confidence bound term in MCTS planning. This gives extra exploration pressure to actions with high probability under the current policy network. Guided Policy Search (GPS) (Levine and Koltun, 2013) penalizes a newly planned trajectory for departing too much from the trajectory generated by the current policy network. As a final example, Guo et al. (2014) let the current policy network decide at which locations to perform the next search, i.e., the policy network influences the distribution of states used as a starting point for planning (Sec. 5.1, a form of prioritization). There are various ways in which we may incorporate prior knowledge from a policy, and there seems to be open research ground to identify the best of these approaches.

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

- *Value update*: A typical choice for a value target is the state(-action) value estimate at the root of the search tree. The estimate of course depends on the back-up policy, which can either be on- or off-policy. For methods that do not go wide over the actions, like Dyna (Sutton, 1990), we may use a classic Q-learning target (one-step, off-policy). For planning cycles that do go wide (and deep), we can combine on- and off-policy back-ups throughout the tree in various ways. Willemsen et al. (2020) present a recent study of the different types of back-up policies in a tree search. After constructing

the value target, the value approximation is usually trained on a *mean-squared error* (MSE) loss (Veness et al., 2009; Moerland et al., 2018b). However, other options are possible as well, like a cross-entropy loss between the softmax of the Q-values from the search and the Q-values of a learned neural network (Hamrick et al., 2020).

- *Policy update*: For the policy update we again observe a variety of training targets and losses, depending on the type of planning procedure that is used. For example, AlphaGo Zero (Silver et al., 2017a) uses MCTS planning, and constructs a policy training target by normalizing the visitation counts at the root node. The policy network is then trained on a cross-entropy loss with this distribution. Guo et al. (2014) apply the same idea with a one-hot encoding of the best action, while Moerland et al. (2018b) cover an extension to a loss between discrete counts and a continuous policy network. As a completely different approach, Guided policy search (GPS) (Levine and Abbeel, 2014) minimizes the Kullback-Leibler (KL)-divergence between a planned trajectory and the output of the policy network. Some differential planning approaches also directly update a differentiable global representation (Deisenroth and Rasmussen, 2011).

  We may also train a policy based on a value estimate. For example, Policy Gradient Search (PGS) (Anthony et al., 2019) uses the policy gradient theorem (Williams, 1992) to update a policy from value estimates in a tree. Note that gradient-based planning (discussed in Sec. 5.3) also belongs here, since it directly generates gradients to update the differentiable policy.

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

Note that arrows b and c in Figure 8 form a closed sub-loop in the overall integration. There has been much recent interest in this sub-loop, which iterates planning based on policy/value priors (arrow b), and policy/value learning based on planning output (arrow c). A successful algorithms in this class is AlphaGo Zero (Silver et al., 2017a), which is an instance of *multi-step approximate real-time dynamic programming* (MSA-RTDP). MSA-RTDP extends the classic DP ideas by using a 'multi-step' lookahead, learning the value or policy ('approximate'), and operating on traces through the environment ('real-time'). Efroni et al. (2019) theoretically study MSA-RTDP, showing that higher planning depth d decreases sample complexity in the real environment at the expense of increased computational complexity. Although this result is intuitive, it does show that planning may lead to better informed real-world decisions, at the expense of increased (model-based) thinking time. In addition, iterated planning and learning may also lead to more stable learning, which we discuss in Sec. 7.3.

Planning output for action selection in the real environment We may also use planning to select actions in the real environment. While model-free RL has to use the value or policy approximation to select new action in the environment (Fig. 8, arrow e), model-based RL may also select actions directly from the planning output (Fig. 8, arrow d). Some methods only use planning for action selection, not for value/policy updating (Tesauro and Galperin, 1997; Silver et al., 2008), for example because planning updates can have uncertainty. However, many methods actually combine both uses (Silver et al., 2017a, 2018; Anthony et al., 2017; Moerland et al., 2018b).

Selection of the real-world actions may happen in a variety of ways. First of all, we may greedily select the best action from the plan. This is the typical approach of methods

that 'plan over a learned model' (Table 1). The cardinal example in this group are *model predictive control* (MPC) or *receding horizon control* approaches. In MPC, we find the greedy action of a k-step lookahead search, execute the greedy action, observe true next state, and repeat the same procedure from there. The actual planning algorithm in MPC may vary, with examples including iLQR (Watter et al., 2015), direct optimal control (Nagabandi et al., 2018c; Chua et al., 2018), Dijkstra's algorithm (Kurutach et al., 2018), or repeated application of an inverse model (Agrawal et al., 2016). MPC approaches do not use learning for the value or policy function, and are therefore not model-based RL. MPC easily deals with (changing) constraints on the state and action space (Kamthe and Deisenroth, 2017), and is especially popular in robotics and control tasks.

Note that we do not have to execute the greedy action after planning. Some approaches introduce additional exploration noise over the greedy planning action (Silver et al., 2017a). Other methods intentionally include exploration criteria for their real action. For example, Dearden et al. (1998) explore based on the 1value of perfect information' (VPI), which estimates from the model what exploratory action has the highest potential to change the greedy policy. Indeed, we may actually 'plan for exploration', i.e., decide which exploratory sequence of real world actions is most promising by means of planning (Lowrey et al., 2018; Sekar et al., 2020). Planning may identify temporally correlated action sequences that perform deep exploration towards new reward regions, which local exploration methods would fail to identify due to jittering behaviour (Osband et al., 2016). We call this approach *two-phase exploration* (first exploring within a plan, then exploring in the real, irreversible environment), and extensively discuss the benefits of this idea in Sec. 7.

This concludes our discussion of the main considerations in planning-learning integration. Table 2 summarizes the framework, showing the potential decisions on each dimension.

5.5 Conceptual comparison of approaches

This chapter discussed the various ways in which planning and learning can be integrated. We will present two summaries of the discussed material. First of all, Figure 9 summarizes the different types of connectivity that may be present in planning-learning integration. The figure is based on the scheme of Figure 8, as used throughout this section, and the classification of model-based RL methods described in Table 1.

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

As a second illustration, Table 3 compares several well-known model-based RL algorithms on the dimensions of our framework for planning-learning integration (Table 2). We see how different integration approaches make widely different choices on each of the dimensions. It is hard to judge whether some integration approaches are better than others. These aspects also partially depend on the problem type at hand. For example, when we have large computational resources available, we may be able to afford the high planning budget per timestep used by AlphaGo Zero (Silver et al., 2017a), which aims for high asymptotic performance. Gradient-based planning can be useful, but is mostly applicable to continuous control tasks. Backward planning (prioritized sweeping) can be useful, but does require us to learn a backward model. For many considerations, there are both pros and cons. Usually, the eventual decision depends on the type of benefit (of model-based RL) we aim for, which will be discussed in the next section.

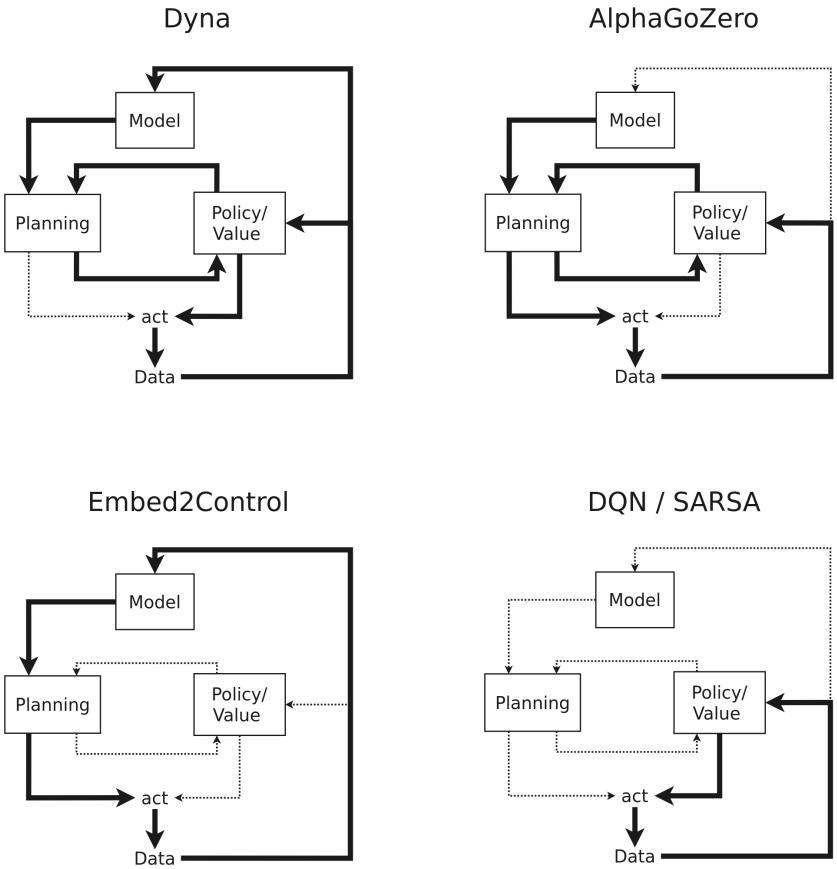

Figure 9: Comparison of planning and learning algorithms, based on the general visualization of learning/planning integration from Figure 8. Thick lines are used by an algorithm. Dyna (Sutton, 1991) (top-left) is an example of model-based RL with a learned model. AlphaGo Zero (Silver et al., 2017a) (top-right) is an example of model-based RL with a known model. Note that therefore the model does not need updating from data. Embed2Control (Watter et al., 2015) (bottom-left) is an example of planning over a learned model. For comparison, the bottom right shows a model-free RL algorithm, like Deep Q-Network (Mnih et al., 2015) or SARSA (Rummery and Niranjan, 1994) with eligibility traces

.

Table 3: Systematic comparison of different model-based RL algorithms on the dimensions of planning-learning integration (Sec. 5). Colour coding: green = model-based RL with a learned model, red = model-based RL with a known model, blue = planning over a learned model (see Table 1). Uncertainty estimation methods: GP = Gaussian Process, BE = bootstrap ensemble. Uncertainty propagation methods: Par = parametric propagation, Sam = sample-based propagation (particle methods). † = Before learning, the authors collect an initial batch of training data for the model. The number of real steps before the *first* plan is therefore 3.000-30.000, depending on the task. Afterwards, they start to interact with the environment, planning at every step. ⋆ = gradient-based planners improve a reference trajectory based on gradients. Although there is only trajectory, the gradient does implicitly go wide over the actions, since it tells us in which direction the continuous action should be moved.

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

6 Implicit Model-based Reinforcement Learning

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

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

In particular, we will focus on methods that use gradient-based optimization. In those case, we embed (parts of) the model-based RL process within a computational graph, which eventually outputs a value or action recommendation. Since the graph remains end-to-end differentiable, we may optimize one or more elements of our model-based RL procedure for a value or action recommendation. One would be tempted to call the field *end-to-end* model-based RL, but note that the underlying principles are more general, and could also work with gradient-free optimization.

We may use implicit model-based RL to replace each (or both) of the steps of explicit model-based RL: 1) to learn implicit transition models, better known as *value equivalent models* (Sec. 6.1), and 2) to implicitly *learn how to plan* (Sec. 6.2). We will first discuss each category individually, and afterwards discuss how they can also be combined (Sec. 6.3). An overview of the ideas and papers in the following sections is provided in Table 4.

6.1 Value equivalent models

Standard model learning approaches, as discussed in Section 4, learn a forward model that predicts the next state of the environment. However, such models may predict several aspects of the state that are not relevant for the value. In some domains, the forward dynamics might be complicated to learn, but the aspects of the dynamics that are relevant for value prediction might be much smoother and easier to learn. This is the key insight below *value equivalent models* (Grimm et al., 2020). Value equivalent models are unrolled inside the computation graph to predict a future value, instead of a future state. As such, these models are enforced to emphasize value-relevant characteristics of the environment. Thereby, value equivalent models are really representation learning technique for model learning, as already mentioned at the end of Sec. 4.7. They can be combined with any other type of loss function as well, like the ability to predict a future reward.

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

A slightly different approach is taken by the Predictron (Silver et al., 2017b). It only receives a state as input (not a sequence of actions), and internally unrolls its models to

predict the value of that state. When we want to select an optimal action (plan), we can unroll the Predictron for each available action in a state. Note that MuZero, VPN and the Predictron all include a state encoding function that gets trained on the same value-equivalent target, which can therefore helps for representation learning (Sec. 4.7) as well. Moreover, all three unroll their model in an *implicit policy evaluation* setting, along a single trace. Planning, which includes policy improvement, is still explicit, and does not happen inside the computational graph.

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

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

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

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

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

6.2 Learning to plan

We may also use the implicit planning idea to optimize for the planning operations themselves. So far, we encountered two ways in which learning may enter model-based RL: i) to learn a dynamics model (Sec. 4), and ii) to learn a value or policy function (from planning output) (Sec. 5). We now encounter a third level in which learning may enter model-based RL: to *learn to plan*. The idea is to optimize our planner over a sequence of tasks to eventually obtain a better planning algorithm, which is a form of meta-learning (Vanschoren, 2019).

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

We will discuss three examples of learning to plan: MCTSNets (Guez et al., 2018), Imagination-augmented agents (I2A) (Racanière et al., 2017), and Imagination-based planner (IBP) (Pascanu et al., 2017). MCTSNets optimize elements of the MCTS algorithm, like selection, back-up and final recommendation, against the ability to output the correct optimal action in the game Sokoban. The dynamics model in MCTSNets is assumed to be known. In contrast, both I2A and IBP first separately learn a standard forward dynamics model, as extensively discussed in Sec. 4. In the planning graph, I2A then learns how to aggregate the information in these roll-outs, and how this should influence the learned policy.

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

6.3 Combined learning of models and planning

We may also combine both ideas introduced in the previous sections (value equivalent models and learning to plan). If we specify a parameterized differentiable model and a parameterized differentiable planning procedure, then we can optimize the resulting computational graph jointly for the model and the planning operations. This of course creates a harder optimization problem, since the gradients for the planner depend on the quality of the model, and vice versa. However, it is the most end-to-end approach to model-based RL we can imagine, as all aspects discussed in Sections 4 and 5 get wrapper into a single optimization.

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

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

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

This concludes our discussion of implicit model-based RL. An overview of the discussed papers and methodology is presented in Table 4. The strength of the implicit model-based RL approach is tied to the strength of optimization in general, and other fields of machine learning have already shown that optimization may beat human design intuition (given enough data and computational resources). Moreover, value equivalent models may be beneficial in problems where dynamics are complicated, but the dynamics relevant for value estimation are much smoother.

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

7 Benefits of Model-based Reinforcement Learning

Model-based RL may provide several benefits, which we will discuss in this section. However, in order to identify benefits, we first need to discuss performance criteria, and establish terminology about the two types of exploration in model-based RL.

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

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

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

Table 4: Comparison of implicit model-based RL approaches. **Rows**: algorithm colour coding, yellow = explicit model-based RL (for comparison), green = value equivalent models (Sec. 6.1), red = learning to plan (Sec. 6.2), blue = combination of value equivalent models and learning to plan (Sec. 6.3). **Columns**: Implicit planning implies some form of policy improvement in the computation graph. Learning to plan implies that this improvement operation is actually optimized. For planning, we shortly mention the specific planning structure between brackets. MCTS = Monte Carlo Tree Search, iLQR = iterative Linear Quadratic Regulator, RNN = recurrent neural network.

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

We will now discuss the potential benefits of model-based RL (Figure 10). First, we will discuss enhanced data efficiency (Sec. 7.1), which uses planning (increased model sample complexity) to reduce the real-world sample complexity. Second, we discuss exploration methods that use model characteristics (Sec. 7.2). As a third benefit, we discuss the potential of model-based RL with a known model to reach higher asymptotic performance (optimality/cumulative reward) (Sec. 7.3). A fourth potential benefit is transfer (Sec. 7.4), which attempts to reduce the sample complexity on a sequence of tasks by exploiting commonalities. Finally, we also shortly touch upon safety (Sec. 7.5), and explainability (Sec. 7.6).

7.1 Data Efficiency

A first approach to model-based RL uses planning to reduce the real-world sample complexity. Real-world samples are expensive, both due to wall-clock time restrictions and hardware vulnerability. Enhanced data efficiency papers mostly differ by how much effort they invest per planning cycle (Sec. 5.2). A first group of approaches tries to *squeeze* out as much information as possible in every planning loop. These typically aim for maximal data efficiency, and apply each planning cycle until some convergence criterion. Note that batch reinforcement learning (Lange et al., 2012), where we only get a single batch of data from a running system and need to come up with an improved policy, also falls into this group. The second group of approaches continuously plans in the background, but does not aim to squeeze all information out of the current model.

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

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

- *Mixing*: The second group of approaches simply mixes model-based updates with model-free updates, usually by making model-based updates (in the background) throughout the (reachable) state space. The original idea dates back to the Dyna architecture of Sutton (1990), who reached improved data efficiency of up to 20-40x in a gridworld problem. In the context of high-dimensional function approximation, Gu et al. (2016); Nagabandi et al. (2018c) used the same principle to reach a rough 2-5 times improvement in data efficiency.

  An added motivation for the mixing approach is that we may still make model-free updates as well. Model-free RL generally has better asymptotic performance than

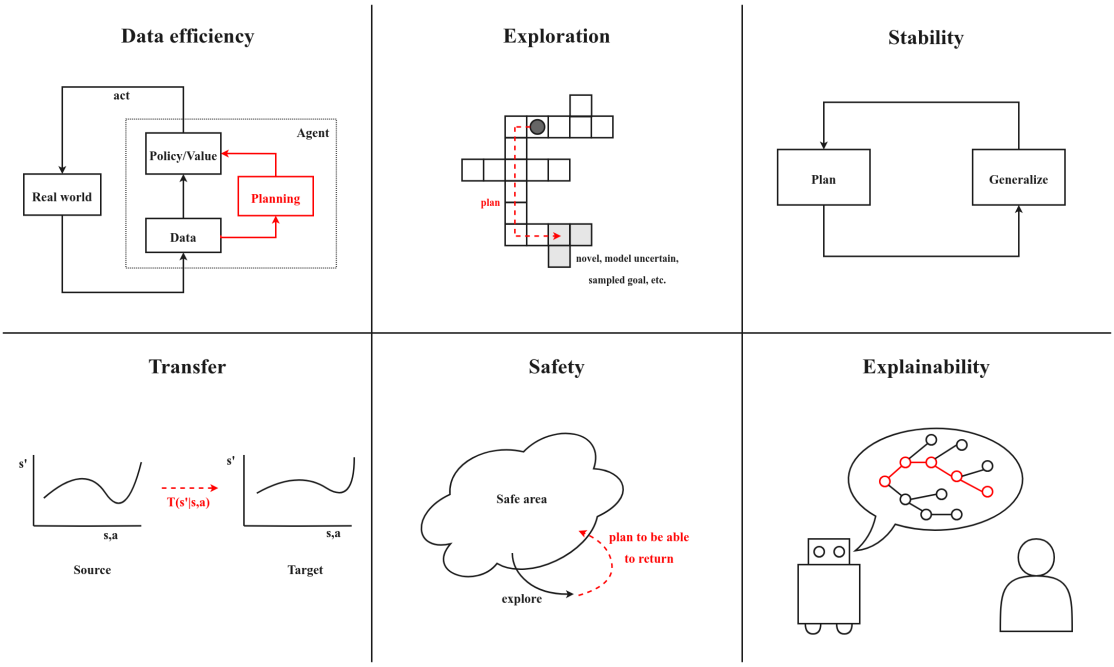

Figure 10: Benefits of model-based reinforcement learning, as discussed in Section 7.

model-based RL with a learned model. By combining model-based an model-free updates, we may speed-up learning with the model-based part, while still reaching the eventual high asymptotic performance of model-free updates. Note that model-based RL with a known model may actually reach higher asymptotic performance (Sec. 7.3) than model-free RL, which shows that the instability is really caused by the uncertainty of a learned model.

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

7.2 Exploration

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

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

  In contrast, model-based RL agents use 'two-phase exploration', since they may combine 1) an exploration strategy within the planning cycle, and 2) a (usually more

Table 5: Categories of exploration methods. Grey cells are considered 'model-based exploration', since they either use state-based characteristics and/or plan over the model to find better exploration decisions (two-phase exploration).

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

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

- *Value-based versus state-based exploration (intrinsic motivation)*: Most RL approaches use a form of 'value-based' exploration. Value-based methods base their exploration strategy on the current value estimates of the available actions. Actions with a higher value estimate will also get a higher probability of selection, where the perturbation may for example be random (Plappert et al., 2017; Mnih et al., 2015) or based on uncertainty estimates around these values (Auer, 2002; Osband et al., 2016; Moerland et al., 2017a). The model-based alternative is to use 'state-based' exploration. In this case, we do not determine the exploration potential of a state based on reward or value relevancy, but rather based on state-specific, reward independent properties derived from the interaction history with that state. A state may for example be interesting because it is novel or has high uncertainty in its model estimates. These approaches are better known as *intrinsic motivation* (IM) (Chentanez et al., 2005).

The two above distinctions form four combinations, as visualized in Table 5. We define *model-based exploration* as 'any exploration approach that uses either state-based exploration and/or two-phase exploration' (indicated by the grey boxes in Table 5). Note that we consider all state-based exploration methods to be model-based RL. State-based exploration methods often use model-based characteristics or a density model over state space, which in the tabular case can directly be derived from a tabular model. We therefore include all state-based exploration as model-based RL, even when it is applied in one-phase (e.g, with model-free value approximation).

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

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

future, first use exploitation to get close to that new target, and only then start to explore (i.e., commit to a sequence of actions).

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

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

  As an alternative, we may also use *task-separated* exploration back-ups. In these cases, the global solution (value or policy function) is explicitly separated from the exploration information, like the way to get back to a particular interesting region. For example, Shyam et al. (2019) propose to store separate value functions for the intrinsic and extrinsic rewards. We may also learn a policy/value function towards any goal in the state space, i.e., a generalized policy/value function (see Sec. 4.8). As an alternative, we may also store the exact trace towards particular goals, which has a relation to episodic memory (Pritzel et al., 2017). For example, Ecoffet et al. (2019) assume that we can exactly reset an agent to any state we previously reached. Finally, we may also separately learn the policy parameters that reached a particular state (Laversanne-Finot et al., 2018). Task-separated back-ups introduce more complexity for learning, but also seem a more principled approach to separate exploration from exploitation.
- *Shallow versus deep exploration back-ups*: Similar to shallow and deep exploration (in the forward sense), the depth of the back-up is also important for exploration. As well-known from classic RL theory, a back-up can be *shallow* (e.g., a one-step target) or *deep* (e.g., a Monte Carlo target or n-step method) (Sutton and Barto, 2018). When we search for the optimal solution, one-step back-ups can be off-policy and therefore have the benefit of converging to the optimal solution. Therefore, shallow back-ups clearly have their benefit for convergence. However, from an exploration perspective, they have a potential drawback. Imagine we just encountered an interesting novel state, but we only back-up this information for one-step towards the previous state. In the next episode, we will not be able to see the information near the start location (it has not been propagated far enough). Ecoffet et al. (2019) call this the 'detachment' problem for exploration. The agent then has to stumble around again until it finds a new intrinsic reward, or a state to which the previous intrinsic reward was backed-up.

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

solution. We may then not store the quickest route back to the novel state, but we do not care about optimality yet, we just want to get back there. Alternatively, we can also store the entire trace or policy parameters needed to reach a particular interesting state or region (Laversanne-Finot et al., 2018), which is by definition deep. In any case, deep back-ups may strongly accelerate our ability to build on novel discoveries.

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

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

$$r_t(s, a, s') = r^e(s, a, s') + \eta \cdot r^i(s, a, s'), \tag{5}$$

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

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

$$r^i(s, a, s') \propto 1/(1 + n(s, a)), \tag{6}$$

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

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

$$r^i(s, a, s') = \sqrt{l(s, a)}, \tag{7}$$

where $l(s, a)$ denotes the number of timesteps since the last trial at (s, a). More recent examples of intrinsic rewards include *model prediction error* (Stadie et al., 2015; Pathak et al., 2017), *surprise* (Achiam and Sastry, 2017), *information gain* (Houthooft et al., 2016), and *feature control* (the ability to change elements of our state over time) (Dilokthanakul et al., 2019). Note that intrinsic rewards for recency and model prediction error may help overcome non-stationarity (Sec. 4.5) as well (Lopes et al., 2012). Multiple intrinsic rewards can also be combined, like a combination of novelty and model uncertainty (Hester and Stone, 2012a). Note that many of these intrinsic motivation ideas originate in emotion theory, which was surveyed for RL agents by Moerland et al. (2018a).

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

Many of the above knowledge-based IM methods are implemented in a one-phase way, i.e., the intrinsic reward is computed when encountered, but there is not explicit planning towards it. We can of course also combine knowledge-based IM with two-phase exploration

(Sekar et al., 2020), i.e. 'plan to explore'. As mentioned before, nearly all knowledge-based IM approaches use task-conflated propagation, while Shyam et al. (2019) do learn separate value functions for the intrinsic and extrinsic rewards. The back-up depth of knowledge-based IM varies from shallow to deep, depending on the type of value back-up.

Competence-based intrinsic motivation Competence-based intrinsic motivation builds on the same curiosity principles as knowledge-based IM. However, competence-based IM selects new exploration targets based on *learning progress*, which focuses on the competence of the agent, rather than the knowledge about the MDP. The goal is to generate a *curriculum* of tasks for learning progress (Bengio et al., 2009). A formalization of these ideas are Intrinsically Motivated Goal Exploration Processes (IMGEP) (Baranes and Oudeyer, 2009). They consist of three steps: 1) learn a goal space, 2) sample a goal, and 3) plan towards the goal.

Goal space learning was already discussed in Sec. 4.7 and 4.8. The general aim is to capture the salient directions of variation in a task in a representation. For competence-based IM, it may be useful to learn a *disentangled* representation, where each controllable object is captured by a separate dimension in the representation. Then, we can create a better curriculum by sampling new subgoals that alter only one controllable object at a time (Laversanne-Finot et al., 2018).

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

In the third step, we actually attempt to reach the sampled goal. The key idea is that we should already know how to get close to the new goal, since we sampled it close to a previously reached state. Goal-conditioned value functions (discussed in Sec. 4.8) can be one way to achieve this, but we may also attempt to learn a mapping from current state and goal to policy parameters (Laversanne-Finot et al., 2018). The latter approach attempts to generalize in policy parameter space, and is thereby a deep propagation example.

Competence-based IM often uses deep exploration and task-separated, deep back-ups. However, the true difference between knowledge and competence-based approaches is the type of information that makes a state/goal salient (knowledge, e.g. novelty, or competence, i.e., learning progress). All other distinctions mentioned in the beginning of this section are applicable to both. The vanilla approach in each category is illustrated in Figure 11.

Finally, this section on model-based exploration has not discussed any hierarchical RL methods, since these were already covered in Sec. 4.8. Hierarchy can be used in a model-free or model-based way. In both cases, good higher level actions can strongly reduce the exploration complexity, like the depth of a tree during planning. As discussed in Sec. 4.8, good end-points for hierarchical actions can for example be obtained from global coverage of state space, which reminds of the goal spaces used in competence-based intrinsic motivation. In any cases, hierarchy will like be a crucial component of (model-based) exploration as well.

7.3 Stability

Another benefit of model-based RL, in the context of a known model, seems better asymptotic performance. For model-based RL with a learned model, the common knowledge is that we may improve data efficiency, but lose asymptotic performance in the long run. However, recent attempts of model-based RL with a known model, like AlphaGo Zero (Silver

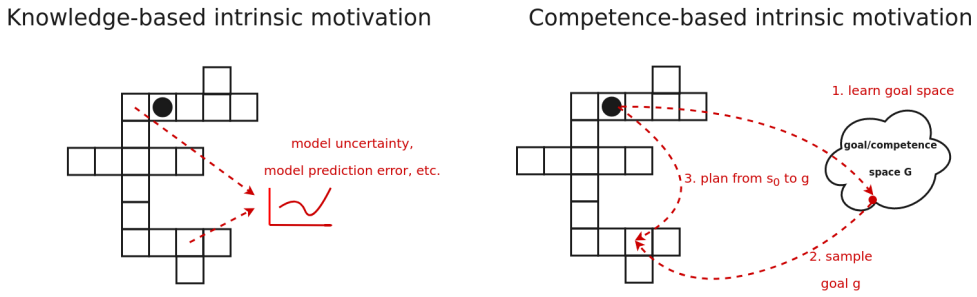

Figure 11: Knowledge-based versus competence-based intrinsic motivation. Solid circle identifies the current agent position. **Left**: In knowledge-based intrinsic motivation, every state (the arrows show two examples) in the domain gets associated with an intrinsic reward based on local characteristics, like visitation frequency, uncertainty of the model, prediction error of the model, etc. **Right**: In competence-based intrinsic motivation, we learn some form of a goal-space that captures (and compresses) the directions of variation in the domain. We then sample a new goal, for example at the edge of our current knowledge base, and explicitly try to reach it, re-using the way we previously got close to that state.

et al., 2017a) and Guided Policy Search (Levine and Koltun, 2013), manage to outperform model-free attempts on long-run empirical performance. This suggests that with a perfect (or good) model, model-free RL may actually lead to better (empirical) asymptotic performance. Moreover, MuZero (Schrittwieser et al., 2019) uses a (value-equivalent) learned model and actually outperforms the asymptotic performance of AlphaGo Zero.

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

There is some initial work that supports these ideas. Silver et al. (2008) already described the use of *transient* and *permanent* memory, where the transient memory is the local plan that fine-tunes the value estimates. Both Moerland et al. (2020b) and Wang et al. (2019) recently studied the trade-off between planning and learning (already mentioned in Sec. 5.2), finding that optimal performance requires an intermediate planning budget per real step, and not a very high budget (exhaustive search), or no planning budget per timestep at all (model-free RL). Since model-free RL is notoriously unstable in the context of function approximation (Henderson et al., 2018), we may hypothesize that the combination of global function approximation (learning) and local atomic/tabular representation (planning) helps stabilize learning and achieve better asymptotic performance (see Hamrick et al. (2020) as well).

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

reasoning and 'thinking slow'.

7.4 Transfer

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

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

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

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

One way to achieve good transfer is by learning representations that generalize well. The object-oriented and physics-based approaches, already introduced in Sec. 4.7, have shown success in achieving this. For example, Schema Networks (Kansky et al., 2017) learn object interactions in Atari games, and manage to generalize well to several variations of Atari Breakout, like adding a new wall or slightly changing the dynamics (while still complying with the overall physics rules).

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

initialization that can quickly adapt to new tasks sampled from the task space (Clavera et al., 2018), which implicitly transfers knowledge about the dynamics of related tasks.

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

7.5 Safety

Safety is an important issue, especially when learning on real-world systems (Amodei et al., 2016). For example, with random exploration it is easy to break a robot before any learning has taken place. Berkenkamp et al. (2017) studies a model-based safe exploration approach based on the notion of asymptotic stability. Given a 'safe region' of the current policy, we want to explore while ensuring that we can always get back to the safe region. As an alternative, Aswani et al. (2013) keep two models: the first one is used to decide on an exploration policy, while the second model has uncertainty bounds and is used for verification of the safety of the proposed policy. Ostafew et al. (2016) ensure constraints by propagating uncertainty information in a Gaussian Process model. Safety is a vital aspect of real-world learning, and it may well become an important motivation for model-based RL in forthcoming years.

7.6 Explainability

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

7.7 Disbenefits

Model-based RL has disbenefits as well. First, model-based RL typically requires additional computation, both for training the model, and for the planning operations themselves. Second, model-based RL methods with a learned model can be very unstable due to uncertainty and approximation errors in the model. Therefore, although these approaches can be more data efficient, they also tend to have lower asymptotic performance. We already extensively discussed how to deal with model uncertainty. Third, model-based RL methods require additional memory, for example to store the model. However, with function approximation this is typically not a large limitation. Finally, model-based RL algorithms typically have more tunable hyperparameters than model-free algorithms, including hyperparameters to estimate uncertainty, and hyperparameters to balance planning and real data collection. Most of these disbenefits are inevitable, and we are essentially trading extra computation, memory and potential instability (for a learned model) against better data efficiency, targeted exploration, transfer, safety and explainability.

8 Related Work

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

Moerland et al. (2020a) present a framework for reinforcement learning and planning that disentangles their common underlying dimensions, but does not focus on their integration. In some sense, Moerland et al. (2020a) look 'inside' each planning or reinforcement learning cycle, strapping their shared algorithmic space down into its underlying dimensions. Instead, our work looks 'over' the planning cycle, focusing on how we may integrate planning, learning and acting to provide mutual benefit.

Hamrick (2019) presents a recent coverage of mental simulation (planning) in deep learning. While technically a model-based RL survey, the focus of Hamrick (2019) lies with the relation of these approaches to cognitive science. Our survey is more extensive on the model learning and integration side, presenting a broader categorization and more literature. Nevertheless, the survey by Hamrick (2019) is an interesting companion to the present work, for deeper insight from the cognitive science perspective. Plaat et al. (2020) also provide a recent description of model-based RL in high-dimensional state spaces, and puts additional emphasis on implicit and end-to-end model-based RL (see Sec. 6 as well).

Finally, several authors (Nguyen-Tuong and Peters, 2011; Polydoros and Nalpantidis, 2017; Sigaud et al., 2011) have specifically surveyed structured model estimation in robotics and control tasks. In these cases, the models are structured according to the known laws of physics, and we want to estimate a number of free parameters in these models from data. This is conceptually similar to Sec. 4, but our work discusses the broader supervised learning literature, when applicable to dynamics model learning. Thereby, the methods we discuss do not need any prior physics knowledge, and can deal with much larger problems. Moreover, we also include discussion of a variety of other model learning challenges, like state and temporal abstraction.

9 Discussion

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

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

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

A second part of benchmarking is actually running and comparing algorithms. Although many benchmarking environments for RL have been published in recent years (Bellemare et al., 2013; Brockman et al., 2016), and benchmarking of model-free RL has become quite

popular, there is relatively little work on benchmarking model-based RL algorithms. Wang et al. (2019) recently made an important first step in this direction by benchmarking several model-based RL algorithms, and the field would profit from more efforts like these.

For reporting results, an important remaining challenge for the entire RL community is standardization of learning curves and results. The horizontal axis of a learning curve would ideally show the number of unique flops (computational complexity) or the number of real world or model samples. However, many papers report 'training time in hours/days' on the horizontal axis, which is of course heavily hardware dependent. Other papers report 'episodes' on the horizontal axis, while a model-based RL algorithm uses much more samples than a model-free algorithm per episode. When comparing algorithms, we should always aim to keep either the total computational budget or the total sample budget equal.

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

- Asymptotic performance: Model-based RL with a learned model tends to have better sample complexity, but inferior asymptotic performance, compared to model-free RL. This is an important limitation. AlphaGo Zero recently illustrated that model-based RL with a known model should be able to surpass model-free RL performance. However, in the context of a learned model, a major challenge is to achieve the same optimal asymptotic performance as model free RL, which probably requires better ways of estimating and dealing with model uncertainty.
- Hierarchy: A central challenge, which has already received much attention, is temporal abstraction (hierarchical RL). We still lack consistent methods to identify useful subroutines, which compress, respect reward relevancy, identify bottleneck states and/or focus on interaction with objects and salient domain aspects. The availability of good temporal abstraction can strongly reduce the depth of a tree search, and is likely a key aspect of model-based learning.
- Exploration & Competence-based intrinsic motivation: A promising direction within exploration research could be competence-based intrinsic motivation (Oudeyer et al., 2007), which has received less attention than its brother knowledge-based intrinsic motivation (see Sec. 7.2). By sampling goals close to the border of our currently known set, we generate an automated curriculum, which may make exploration more structured and targeted.
- Transfer: We believe model-based RL could also put more emphasis on the transfer setting, especially when it comes to evaluating data efficiency. It can be very hard to squeeze out all information on a single, completely new task. Humans mostly use forward planning on reasonably certain models that generalize well from previous tasks. Shifting RL and machine learning from single task optimization to more general artificial intelligence, operating on a variety of tasks, is an important challenge, in which model-based RL may definitely play an important role.
- Balancing: Another important future question in model-based RL is balancing planning, learning and real data collection. These trade-offs are typically tuned as hyperparameters, which seem to be crucial for algorithm performance (Wang et al., 2019; Moerland et al., 2020b). Humans naturally decide when to start planning, and for how long (Kahneman, 2011). Likely, the trade-off between planning and learning should be a function of the collected data, instead of a fixed hyperparameter.
- Prioritized sweeping: Prioritized sweeping has been very successful in tabular settings, when the model is trivial to revert. As mentioned throughout the survey, it has also been applied to high-dimensional approximate settings, but this creates a much larger challenge. Nevertheless, exploration in the forward direction may actually be just

as important as propagation in the backwards direction, and prioritized sweeping in high-dimensional problems is definitely a topic that deserves attention.

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

10 Summary

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

- Nomenclature in model-based RL is somewhat vague. We define model-based RL as 'any MDP approach that uses i) a model (known or learned) and ii) learning to approximate a global value or policy function'. We distinguish three categories of planning-learning integration: 'model-based RL with a learned model', 'model-based RL with a known model', and 'planning over a learned model' (Table 1).
- Model-based reinforcement learning may first require approximation of the dynamics model. Key challenges of model learning include dealing with: environment stochasticity, uncertainty due to limited data, partial observability, non-stationarity, multi-step prediction, and representation learning methods for state and temporal abstraction (Sec. 4).
- Integration of planning and learning involves a few key aspects: i) where to start planning, ii) how much budget to allocate to planning and acting, iii) how to plan, and iv) how to integrate the plan in the overall learning and acting loop. Planning-learning methods widely vary in their approach to these questions (Sec. 5).
- Explicit model-based RL manually designs model learning, planning algorithms and the integration of these. In contrast, implicit model-based RL optimizes elements of this process, or the entire model-based RL computation, against the ability to predict an outer objective, like a value or optimal action (Sec. 6).
- Model-based RL can have various benefits, including aspects like data efficiency, targeted exploration, transfer, safety and explainability (Sec. 7). Recent evidence indicates that the combination of planning and learning may also provide more stable learning, possibly due to the mutual benefit of global function approximation and local tabular representation.

In short, both planning and learning are large research fields in MDP optimization that depart from a crucially different assumption: the type of access to the environment. Crossbreeding of both fields has been studied for many decades, but a systematic categorization of the approaches and challenges to model learning and planning-learning integration lacked so far. Recent examples of model-based RL with a known model (Silver et al., 2017a; Levine and Koltun, 2013) have shown impressive results, and suggest much potential for future planning-learning integrations. This survey conceptualized the advancements in model-based RL, thereby: 1) providing a common language to discuss model-based RL algorithms, 2) structuring literature for readers that want to catch up on a certain subtopic, for example for readers from either a pure planning or pure RL background, and 3) pointing to future research directions in planning-learning integration.

References

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

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

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

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

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

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

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

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

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

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

Asmuth, J., Li, L., Littman, M. L., Nouri, A., and Wingate, D. (2009). A Bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press.

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

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

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

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

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

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

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

Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE.

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

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

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

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

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

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

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

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

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

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

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

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

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

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Deisenroth, M. and Rasmussen, C. E. (2011). 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fu, J., Levine, S., and Abbeel, P. (2016). One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4019–4026. IEEE.

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

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

Ghahramani, Z. and Hinton, G. E. (1996). Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science.

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

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

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

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

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

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

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

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

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

Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. (2003). Generalizing plans to new environments in relational MDPs. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1003–1010. Morgan Kaufmann Publishers Inc.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Hester, T. and Stone, P. (2012a). Intrinsically motivated model learning for a developing curious agent. In *2012 IEEE international conference on development and learning and epigenetic robotics (ICDL)*, pages 1–6. IEEE.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., and George, D. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1809–1818. JMLR. org.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-SGD: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

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

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

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

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

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

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

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

Machado, M. C., Bellemare, M. G., and Bowling, M. (2017). A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2295–2304. JMLR. org.

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

Mahadevan, S. (2009). Learning Representation and Control in Markov Decision Processes: New Frontiers. *Foundations and Trends® in Machine Learning*, 1(4):403–565.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 752–759. ACM.

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

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

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

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

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

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

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

Pong, V., Gu, S., Dalal, M., and Levine, S. (2018). Temporal Difference Models: Model-Free Deep RL for Model-Based Control. In *International Conference on Learning Representations (ICLR 2018)*. OpenReview. net.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. (2017b). The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3191–3199. JMLR. org.

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

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

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

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

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

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

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

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

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

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

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

Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. (2008). Dyna-style Planning with Linear Function Approximation and Prioritized Sweeping. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 528–536, Arlington, Virginia, United States. AUAI Press.

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

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

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

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

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

Tesauro, G. and Galperin, G. R. (1997). On-line Policy Improvement using Monte-Carlo Search. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 1068–1074. MIT Press.

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

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

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

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

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE.

Todorov, E. and Li, W. (2005). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE.

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

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

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

Van Hoof, H., Chen, N., Karl, M., van der Smagt, P., and Peters, J. (2016). Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE.

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

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

Vanschoren, J. (2019). Meta-learning. In *Automated Machine Learning*, pages 35–61. Springer, Cham.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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