Should Models Be Accurate?

Esra'a Saleh
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
esraa1@ualberta.ca

John D. Martin
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
jmartin8@ualberta.ca

Anna Koop
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
akoop@ualberta.ca

Arash Pourzarabi*
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
pourzara@ualberta.ca

Michael Bowling
Department of Computing Science
University of Alberta / Amii
Edmonton, AB, Canada
mbowling@ualberta.ca

Abstract

Model-based Reinforcement Learning (MBRL) holds promise for data-efficiency by planning with model-generated experience in addition to learning with experience from the environment. However, in complex or changing environments, models in MBRL will inevitably be imperfect, and their detrimental effects on learning can be difficult to mitigate. In this work, we question whether the objective of these models should be the accurate simulation of environment dynamics at all. We focus our investigations on Dyna-style planning in a prediction setting. First, we highlight and support three motivating points: a perfectly accurate model of environment dynamics is not practically achievable, is not necessary, and is not always the most useful anyways. Second, we introduce a meta-learning algorithm for training models with a focus on their usefulness to the learner instead of their accuracy in modelling the environment. Our experiments show that in a simple non-stationary environment, our algorithm enables faster learning than even using an accurate model built with domain-specific knowledge of the non-stationarity.

Keywords: Model-based Reinforcement Learning, Planning, Model Learning, Meta Learning, Dyna, Sample Models

Acknowledgements

We are grateful for the generous support in funding this work. This work was funded by the Natural Sciences and Engineering Research Council, the Alberta Machine Intelligence Institute, and the Canadian Institute For Advanced Research. Computational resources were generously provided by Compute Canada.

*Arash Pourzarabi was killed when Flight PS752 was shot down by an Iranian surface-to-air missile on January 8, 2020. His unfinished research on training generative models to produce experience for model-based RL was a signficant influence on the direction taken in this work.

1 Introduction

A promising approach to sample efficiency in reinforcement learning is model-based RL, where an agent learns a model of the environment. The agent uses this model to generate its own *internal experience* with which it can plan, reducing the need for more expensive *veridical experience*, i.e., environment interactions. Learning effective models for planning remains a challenge in complex or changing environments [6]. Models for planning are traditionally trained to accurately simulate environment dynamics. In complex or changing environments, models will necessarily be imperfect, which can lead to detrimental effects on the learner.

Past work has used several strategies to grapple with imperfect models. The first is to compensate in the planning process. An imperfect model can be used selectively by estimating predictive uncertainty to avoid planning in states where it could be detrimental [1]. Similarly, while longer sequences of internal experience can often be beneficial in planning, their length needs to be limited due to increasing compounding model error [5]. Another strategy is to relax the requirement for models to accurately simulate environment transitions by focusing only on accuracy in the resulting returns. This has been formally articulated through the value equivalence principle [4, 3] and manifested in MuZero [7].

Our work questions the need for focusing on models that are concerned with accurately simulating the environment. We specifically study Dyna-style planning [9] in a prediction setting to provide a building block towards the control setting. We design a simple non-stationary environment that cannot be modelled perfectly without assumptions on how it changes. Then, we use meta-learning to train a model that generates synthetic transitions for fast adaptive learning. The resulting learning speed upon change in the environment makes it competitive even with a stable accurate model of environment dynamics for an agent with privileged access to regions of stability.

2 Should Models Be Accurate?

An important way RL systems can learn about their environment is by anticipating the outcomes of different behaviors using both veridical and internal experience. In the prediction setting, an RL system experiences a stream of observation vectors $\mathbf{o}_t \in \mathbb{R}^d$ and scalar rewards r_t. From this single stream of data, $\mathbf{h}_t \triangleq \mathbf{o}_0, r_1, \mathbf{o}_2, r_2, \mathbf{o}_3, \cdots r_{t-1}, \mathbf{o}_t$, the learner forms an approximate value function (i.e., a prediction) to estimate the expected sum of future discounted rewards. With linear function approximation, predictions are made by projecting a feature vector $\boldsymbol{\phi}_t = \mathbf{f}(\mathbf{h}_t) \in \mathbb{R}^n$ with weights $\boldsymbol{\theta}$:

$$\hat{v}(\boldsymbol{\phi}_t; \boldsymbol{\theta}_t) \triangleq \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t, \qquad \hat{v}(\boldsymbol{\phi}_t; \boldsymbol{\theta}) \approx v(\mathbf{h}_t) \triangleq \mathbf{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | H_t = \mathbf{h}_t].$$

Model-based learners update their prediction weights $\boldsymbol{\theta}$ by planning with internal experience produced by a model $m \in \mathcal{M} \subseteq \mathcal{P}(\mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n)$. In this work, models combine the selection of initial states[1] with the dynamics of a forward transition. At each planning step, the model generates an internal starting state $\tilde{\boldsymbol{\phi}}$, a next state $\tilde{\boldsymbol{\phi}}'$, and a reward $\tilde{r}$. Given this transition sample $(\tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}') \sim m$, an update rule can be applied to the prediction weights $\boldsymbol{\theta}$.

Dyna-style learning systems interleave planning steps with model-free updates on *veridical experience* $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$, which come from the environment's dynamics $m^* \in \mathcal{M}$. In our work, Dyna learners apply semi-gradient updates to reduce one-step temporal difference (TD) errors $\delta \triangleq r + \gamma \boldsymbol{\theta}^\top \boldsymbol{\phi}' - \boldsymbol{\theta}^\top \boldsymbol{\phi}$, using a fixed step size $\alpha \in \mathbb{R}^+$ at each time step: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \boldsymbol{\phi}$. This update is used with both veridical and internal experience.

In addition to learning predictions, model-based systems must also learn the planning model m. These systems are often designed with an *accuracy* objective—to approximate the environment dynamics so that eventually $m \approx m^*$. Using an accurate model, the learner stands to benefit from additional transition samples that simulate the veridical experience. Still, this raises an important question: should a learner strive for its model to be accurate? While accurate models might represent a subset of models that can be useful for learning, they do not represent the entirety of the set. Insisting on model accuracy, rather than model utility, can be a hindrance to fast adaptive learning. We expound on this with three observations.

First, **perfect accuracy in a model is not practically achievable**. Although accuracy is a well-defined objective for model learning, it represents a condition that cannot be achieved. Accuracy is only achieved when the planning model and environment agree—specifically when $m = m^*$. In realistic environments, a learning system can never achieve this condition, because the real world is too big and complex (including other learning agents such as itself) to be discovered and represented with a finite amount of compute. Therefore, fully-accurate models can never be learned.

Second, **even when possible, an accurate model is not necessary**. In spite of the barrier to perfect accuracy, prior work shows that other models can work just as well. For instance, the class of value-equivalent models are those which produce all the same value updates as m^*, but may transition under different dynamics [4]. Since they predict the same values, planning with value-equivalent models will lead to the same solution as an accurate model. Systems such as Predictron

[1] In Dyna-style planning the selection of an initial state (and action) for planning is sometimes called search control [9].

[8] and MuZero [7] are successful examples which also show the benefit of connecting the model learning process to the way it is used for planning and value improvement. More importantly, this line of work shows that even if accuracy could be achieved, fully-accurate models are not necessary.

Third, **an accurate model might not be the most useful**. In this work, planning models are used to approximate the true value function $v \in \mathcal{V}$ by reducing squared TD error with semi-gradient updates. Solutions to this objective come from the set of TD fixed points $\boldsymbol{\vartheta} \subset \mathcal{V}$. Fixed points can be indexed with the model whose experience produced them: $\boldsymbol{\theta}^{(m)} \in \mathcal{V}$.

Given that fully-accurate models cannot be achieved, nor are they needed to reach a fixed point, one may wonder whether there exists models that are *more useful* than others, or even more useful than the environment model! A natural measure of a model's utility could be the number of calls required to approach a TD fixed point. With this, consider a subset of models $\mathcal{M} \subset \mathbb{M}$ with the same TD fixed point as the environment model m^*: $\mathcal{M} \triangleq \{m \in \mathbb{M} | \boldsymbol{\theta}^{(m)} \in \boldsymbol{\vartheta}\}$. The following result shows that there can be models that find solutions faster than the environment model when starting from the same point.

Theorem 1. *There are domains where, for some initial $\boldsymbol{\theta} \in \mathcal{V}$, a TD fixed point $\boldsymbol{\theta}^* \in \boldsymbol{\vartheta}$ can be approached with fewer calls to $m \in \mathcal{M}$ than with the domain dynamics $m^* \neq m$.*

Proof Sketch. Consider a domain that contains k states, which share the same outgoing transitions and value. Suppose the agent is employing a linear function approximator with states represented with a one-hot encoding. Consider a model that produces internal experience from the k-hot vector containing these states, thus treating them as a single state. Clearly $m \neq m^*$, and existing results from the state aggregation literature show such aggregations exist and do not change the TD fixed point. Therefore, $m \in \mathcal{M}$. However, it would take k calls to m^* to produce the update produced by one call to m. □

In short, a perfectly-accurate model is not practically achievable depending on the complexity of an environment, is not necessary by the value equivalence principle, and it might not be the most useful for a learner's objectives. A useful model in the prediction setting is one that helps the learning system obtain an accurate value function.

3 A Demonstration of Useful Models

In this section, we demonstrate how useful models can be trained to aid fast learning. We focus on a prediction problem in a non-stationary environment where Dyna-style planning is used to rapidly adapt to changes in the environment. The environment is constructed so that, despite being simple, an agent's state representation is not sufficient to build an accurate model. We first describe the environment. Then, we present baselines that represent both model-free RL and model-based RL with models aimed at accuracy. After that, we introduce our algorithm that uses meta-learning to train a model aimed at usefulness. We seek to answer the question: *can an algorithm that learns a useful, but not necessarily accurate model, result in faster learning than either an accurate model or an equivalent model-free learning approach?*

Non-Stationary Windy Hallway. To empirically compare the learning performance of planning models, we use a non-stationary windy hallway: a gridworld of 6 columns and 3 rows. The last column contains 3 episode-terminating grid cells. The starting location is in row 1 and column 0. In every timestep, an agent either moves one grid cell North-East or South-East, each with a probability of 0.5 (or possibly just East if already in a northern or southern-most cell). Every 300 episodes, the environment switches between two different reward regimes, where the location of non-zero reward changes. In one regime, reward is +1 for terminating in row 0, and 0 otherwise. In the other, reward is -1 for terminating in row 2 and 0 otherwise. All algorithms use a discount factor of $\gamma = 0.9$, and planning algorithms perform 5 planning updates per environment step. Experiments run for 75,000 steps (15,000 episodes) and each algorithm's value-function estimates are compared against the true expected return.

Baselines. Based on the environment, we select baselines to highlight learning speed when no planning is done and when Dyna-style planning is done with models aiming for accuracy. For our representative algorithm without planning, we use TD(0), which employs value updates from veridical experience only. We call this algorithm *Model-free*. To keep things simple, our baseline representatives of algorithms with Dyna-style planning do not explicitly train a model at all, but replay back veridical experience as the extreme of optimizing for accuracy. We examine two alternatives for how past experience transition tuples are sampled. The simplest option is to store every experienced transition tuple. A model will simply sample transition tuples proportional to how often they were experienced in the past. We call the Dyna algorithm using this model *AllExperienceDyna*. If the environment were Markov in the agent's state representation, this model would approach a perfectly accurate model. As our environment is not Markov, this approach will result in obvious problems as internal experience from different reward regimes can distract it from the true expected return. Our second baseline uses domain-specific knowledge of the environment to only store and resample veridical transitions tuples that are Markov in the agent's state representation (i.e., the non-terminating state transitions). This experience is stable across changes in the reward regime and we call the Dyna algorithm with this model *StableExperienceDyna*. This approach, in many ways, is an

unfair comparison, as it is exploiting domain knowledge to construct a stable model. We could, though, see this as an idealized representative of Abbas and colleagues' approach that does selective planning based on "model inadequacy" [1], which is analogous to using a stable model.

SynthDyna. In Algorithm 2, we propose a proof-of-concept algorithm that we call *SynthDyna*. SynthDyna's model, unlike our planning baselines, is (i) learned, and (ii) generates synthetic internal planning experience that does not aim to accurately simulate environment dynamics. In designing SynthDyna, we seek to demonstrate that giving the model control over the representation of a transition for planning while having a model learning objective informed by utility to the learning system, can result in faster adaptation in our reward switching non-stationary environment. Like a typical Dyna architecture, SynthDyna updates its value function parameters using both veridical experience transitions and internal planning transitions from its model. Unlike our other Dyna representatives, SynthDyna's model is a generative model, using a 2-layer fully connected neural network with parameters $\boldsymbol{\eta}$ that takes in a Gaussian noise vector as input and produces a transition tuple $(\tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}')$. SynthDyna trains this generative model with a meta-learning procedure that uses a meta-loss tied to the learner's primary objective. As seen in line 6 of Algorithm 2, SynthDyna collects every experienced veridical transition $(\boldsymbol{\phi}_{t-1}, r_t, \boldsymbol{\phi}_t)$ and it also collects the value function parameters, $\boldsymbol{\theta}_p$ from the previous step before planning updates were performed. At a given frequency of steps in the environment, SynthDyna samples a batch of tuples where each is $(\boldsymbol{\theta}_p, \boldsymbol{\phi}, r, \boldsymbol{\phi}')$. It then uses that batch's samples as inputs to the metaloss; see Algorithm 1. The metaloss carries out the process of planning with SynthDyna's model for k steps updating $\boldsymbol{\theta}_p$ from the batch. It then evaluates the squared TD-error, on the stored veridical transition $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$ from the batch. To prevent the TD-target in the error from being manipulated before this evaluation step, the value function parameters in the target are set as the initial parameters $\boldsymbol{\theta}$. The gradient of the resulting meta-loss is then used to update the model parameters $\boldsymbol{\eta}$ to improve the model.

Algorithm 1 $\mathcal{L}$: SynthDyna Meta Loss

1: **input:** $\boldsymbol{\theta}$, $\boldsymbol{\phi}$, r,$\boldsymbol{\phi}'$
2: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
3: **for** $1, \cdots, k$ **do**
4: $\quad \tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}' \sim m(\boldsymbol{\eta})$
5: $\quad \tilde{\delta} \leftarrow \tilde{r} + \gamma \boldsymbol{\theta}'^{\top} \tilde{\boldsymbol{\phi}}' - \boldsymbol{\theta}'^{\top} \tilde{\boldsymbol{\phi}}$
6: $\quad \boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}' + \zeta \tilde{\delta} \tilde{\boldsymbol{\phi}}$
7: **return:** $(r + \gamma \boldsymbol{\theta}^{\top} \boldsymbol{\phi}' - \boldsymbol{\theta}'^{\top} \boldsymbol{\phi})^2$

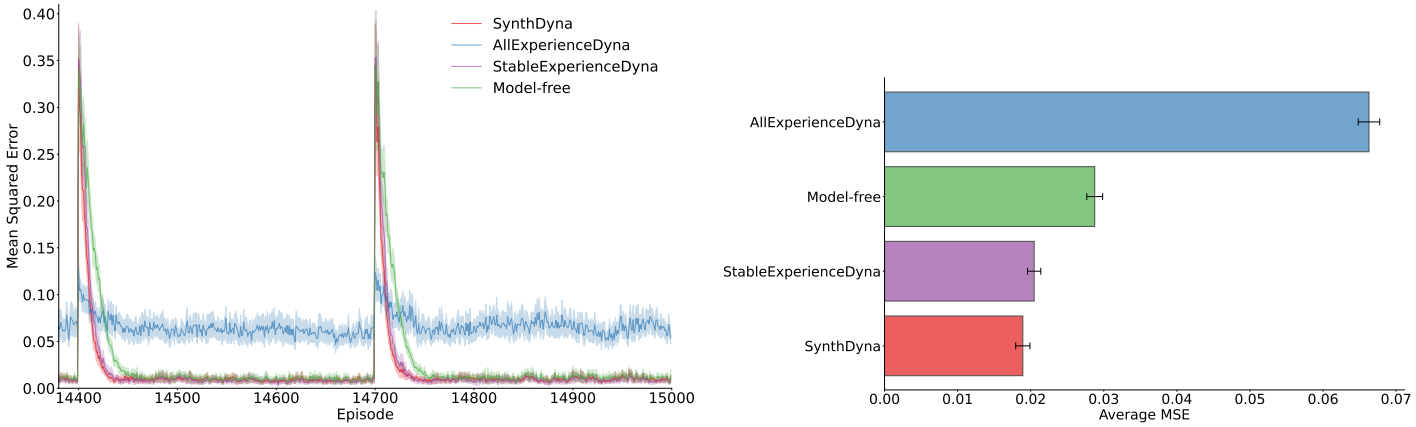
Algorithm 2 SynthDyna for Prediction

1: **input:** feature transform f
2: **initialize:** $\boldsymbol{\theta}_0$, $\boldsymbol{\eta}$, $\mathcal{D} \leftarrow \{\}$, $\boldsymbol{\phi}_0 \leftarrow f(\mathbf{h}_0)$
3: **for** $t = 1, 2, \cdots$ **do**
4: $\quad$ Observe $\mathbf{o}_t$ and r_t from the environment.
5: $\quad \boldsymbol{\phi}_t \leftarrow f(\mathbf{h}_t)$
6: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{\theta}_p, \boldsymbol{\phi}_{t-1}, r_t, \boldsymbol{\phi}_t\}$
7: $\quad$ // Update value with veridical experience.
8: $\quad \delta_t \leftarrow r_t + \gamma \boldsymbol{\theta}_{t-1}^{\top} \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^{\top} \boldsymbol{\phi}_{t-1}$
9: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \alpha \delta_t \boldsymbol{\phi}_{t-1}$
10: $\quad \boldsymbol{\theta}_p \leftarrow \boldsymbol{\theta}_t$ // Save $\boldsymbol{\theta}_t$ for model training
11: $\quad$ // Update value with internal experience.
12: $\quad$ **for** $1, \cdots, k$ **do**
13: $\quad\quad \tilde{\boldsymbol{\phi}}, \tilde{r}, \tilde{\boldsymbol{\phi}}' \sim m(\boldsymbol{\eta})$
14: $\quad\quad \tilde{\delta} \leftarrow \tilde{r} + \gamma \boldsymbol{\theta}_t^{\top} \tilde{\boldsymbol{\phi}}' - \boldsymbol{\theta}_t^{\top} \tilde{\boldsymbol{\phi}}$
15: $\quad\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \beta \tilde{\delta} \tilde{\boldsymbol{\phi}}$
16: $\quad$ // Update SynthDyna model with metaloss $\mathcal{L}$ (Alg 1).
17: $\quad$ Sample minibatch $\mathcal{B}$ from $\mathcal{D}$.
18: $\quad \boldsymbol{\eta} \leftarrow \text{Adam}(\boldsymbol{\eta}, \mathcal{B}, \mathcal{L})$

We hypothesize that the combination of SynthDyna's meta-learned model that aims to be useful rather than accurate, and giving the model full control over the representation of the internal planning transitions will yield a model that can surpass the learning speed of all baselines including our strongest baseline, StableExperienceDyna, that is unfairly informed of stable transitions in the environment.

Results. We evaluate SynthDyna against our three baselines in the the non-stationary hallway environment described previously. Every algorithm's hyper-parameters are selected after a comprehensive grid search. The primary metric we focus on is the mean squared value error (MSE) per episode, which measures the average squared error of the predicted return compared to the expected return over the timesteps in an episode. Let $\boldsymbol{\theta}$ be the value function's parameters before an update based on a veridical transition, $(\boldsymbol{\phi}, r, \boldsymbol{\phi}')$, (i.e., line 9 of Algorithm 2). For an episode of length n, starting at timestep e, the mean squared value errors per episode is computed with: $\text{MSE} = \frac{1}{n} \sum_{t=e}^{e+n} (v(\mathbf{h}_t) - \boldsymbol{\phi}_t^{\top} \boldsymbol{\theta}_t)^2$.

Figure 1a shows the MSE per episode for the last two reward-regime switches. Figure 1b highlights the average MSEs for each algorithm for the last 600 episodes. AllExperienceDyna has the poorest performance across baselines due to its planning process that updates the value function with old experience that could be in direct opposition to the current reward regime. In an environment like this one, the much simpler Model-free algorithm performs substantially better as each update to the value function is exclusively done with the present moment's veridical experience. StableExperienceDyna is able to surpass the performance of AllExperienceDyna and Model-free demonstrating the value of model-based RL for fast adaptation in this domain. It can reap the benefits of planning by updating based on past transition tuples that are stable across switching reward regimes. SynthDyna outperforms even StableExperienceDyna with a statistically significant difference between the average MSEs over the last 600 episodes (two sample t-test, $p < 0.05$). This shows that a model like SynthDyna's can be built in a reward switching non-stationary environment to produce



(a) Mean squared error per episode for each algorithm

(b) Average MSE per algorithm

Figure 1: We ran each algorithm in the non-stationary windy hallway environment for 15,000 episodes, with 30 trials. Fig. 1a, shows the mean squared value error per episode for the last 600 episodes. Fig. 1b, shows the average MSE over the last 600 episodes. Shaded regions and error bars show 95% confidence intervals.

completely synthetic transition tuples and yet update the value function more effectively than if a stable and accurate model of the world was used.

4 Conclusion

"All models are wrong but some are useful" [2]. In this work, we questioned whether models in MBRL should aim to accurately simulate environment dynamics. We posited an alternative approach that models should aim to be useful to the learning system, aiding in faster adaptation. As a proof of concept, we introduced SynthDyna, which uses meta-learning to train a model for Dyna-style planning such that it directly reduces prediction error. In our experiments, SynthDyna outperforms model-based baselines focused on accuracy. Focusing on model usefulness and the approach taken by SynthDyna are promising directions for MBRL to realize its potential for general, sample efficient RL.

References

[1] Z. Abbas, S. Sokota, E. Talvitie, and M. White. Selective Dyna-style planning under limited model capacity. In *International Conference on Machine Learning*, pages 1–10. PMLR, 2020.

[2] G. E. Box. Robustness in the strategy of scientific model building. In *Robustness in statistics*, pages 201–236. Elsevier, 1979.

[3] A.-m. Farahmand, A. Barreto, and D. Nikovski. Value-aware loss function for model-based reinforcement learning. In *Artificial Intelligence and Statistics*, pages 1486–1494. PMLR, 2017.

[4] C. Grimm, A. Barreto, S. Singh, and D. Silver. The value equivalence principle for model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:5541–5552, 2020.

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

[6] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the Arcade Learning Environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[8] D. Silver, H. Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, et al. The predictron: End-to-end learning and planning. In *International conference on machine learning*, pages 3191–3199. PMLR, 2017.

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