The Effect of Planning Shape on Dyna-style planning in High-dimensional State Spaces

G. Zacharias Holland
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
gholland@ualberta.ca

Erin J. Talvitie
Department of Computer Science
Franklin & Marshall College
Lancaster, PA, USA
erin.talvitie@fandm.edu

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

Abstract

Dyna is a fundamental approach to model-based reinforcement learning (MBRL) that interleaves planning, acting, and learning in an online setting. In the most typical application of Dyna, the dynamics model is used to generate one-step transitions from selected start states from the agent's history, which are used to update the agent's value function or policy as if they were real experiences. In this work, one-step Dyna was applied to several games from the Arcade Learning Environment (ALE). We found that the model-based updates offered surprisingly little benefit over simply performing more updates with the agent's existing experience, even when using a perfect model. We hypothesize that to get the most from planning, the model must be used to generate unfamiliar experience. To test this, we experimented with the "shape" of planning in multiple different concrete instantiations of Dyna, performing fewer, longer rollouts, rather than many short rollouts. We found that planning shape has a profound impact on the efficacy of Dyna for both perfect and learned models. In addition to these findings regarding Dyna in general, our results represent, to our knowledge, the first time that a learned dynamics model has been successfully used for planning in the ALE, suggesting that Dyna may be a viable approach to MBRL in the ALE and other high-dimensional problems.

Keywords: Reinforcement learning, planning, model-based, Dyna

1 Introduction

Dyna (Sutton, 1990) is a general architecture for reinforcement learning (RL) agents that flexibly combines aspects of both model-free and model-based RL. In Dyna, the agent learns a predictive model of its environment, which is used to generate simulated experience alongside the agent's real experience. Both types of experience are used in the same way: to update the agent's value function and/or policy. Planning in this way is referred to as *Dyna-style planning*.

Dyna-style planning has many appealing properties that make it a potentially powerful approach in large-scale problems. Because the model indirectly influences the agent's behavior via the value function or policy, the computationally expensive process of planning can be asynchronous with the agent's decision-making loop, retaining a model-free agent's ability to operate on a fine-grained timescale. Furthermore, Dyna permits flexible control over how much computational effort is devoted to planning and how that planning effort is distributed, potentially focusing on updates that will have the most impact, as in Prioritized Sweeping (Moore & Atkeson, 1993). That said, there have been relatively few examples of the application of Dyna-style planning in large-scale problems with high-dimensional observations, such as images.

In this work we study the behavior of Dyna in the Arcade Learning Environment (ALE) (Bellemare, Naddaf, et al., 2013), where agents learn to play games from the Atari 2600 system with raw images as input. Though many successful methods exist for playing games from the ALE and other image based domains (e.g. (Mnih et al., 2015, 2016; Hessel et al., 2018)), the majority of these approaches can be considered *model-free*, in that they learn and improve a policy without using a predictive model of the environment. Model-based methods have the potential to offer dramatic sample complexity benefits and are thus of significant interest. However, despite the introduction of increasingly effective approaches for learning models in Atari games (Bellemare, Veness, & Bowling, 2013; Bellemare et al., 2014; Oh et al., 2015), the application of model-based methods to the ALE is still an open problem, with many challenges to overcome (Machado et al., 2017).

Though some recent approaches to image-based problems incorporate model-like components (e.g. (Tamar et al., 2016; Oh et al., 2017; Weber et al., 2017)), these methods arguably discard many of the anticipated benefits of model-learning by making the model essentially a part of the parameterization of the value function or policy. Part of the allure of model-based RL is that a dynamics model could be learned even when the agent has little guidance on how to behave and that the model can be used as a substitute for gathering expensive experience in the real world. In this work we focus on learning and directly planning with a dynamics model.

In the original Dyna-Q algorithm (Sutton, 1990), planning was performed by selecting several states from the agent's recent experience and using the model to roll forward one step from each of those states and thus generate value function updates. This remains the most common form of Dyna-style planning. In our experiments in the ALE we found that using the model in this way provided surprisingly little benefit, even with a perfect model — similar sample efficiency gains could be obtained simply by performing more model-free updates on the data the agent had already gathered. We note, however, that Dyna-style planning can take on a variety of shapes. For a fixed budget of model interactions, one might generate many one-step rollouts or fewer multi-step rollouts. Empirically we find that planning with longer rollouts yields dramatic improvement when using a perfect model. We also find that planning shape impacts performance when the model is learned (and is therefore imperfect), though model flaws can make rollouts unreliable.

The primary contribution of this work is an empirical exploration of the impact of planning shape on Dyna-style planning. The results offer guidance to future practitioners wishing to apply Dyna. In particular, we find that:

1. For Dyna to take full advantage of a model, it must use the model to generate unfamiliar experience (in essence to replace exploration in the real environment).

2. Performing longer rollouts in the model seems to be a simple, effective way to generate unfamiliar experience.

3. Planning shape seems to be an important consideration even when the model is imperfect.

4. Even if there were dramatic improvements in model-learning, where highly accurate models could be learned very quickly, there would still be little benefit from planning with one-step rollouts. Longer rollouts are necessary to allow planning to exploit more accurate models.

Furthermore, though this paper is primarily focused on the above scientific observations and not on generating new state of the art results in the ALE, the experimental results demonstrate for the first time (as far as we are aware) a sample-complexity benefit from learning a dynamics model in some games. This benefit is at its greatest with proper choice of planning shape. Model-based reinforcement learning in the ALE has proven to be such a challenge that even a modest benefit from a learned model is a significant and unprecedented development (Machado et al., 2017). This finding may point the way toward yet more effective model-based approaches in this domain and establishes Dyna as a viable planning method in large-scale, high-dimensional domains.

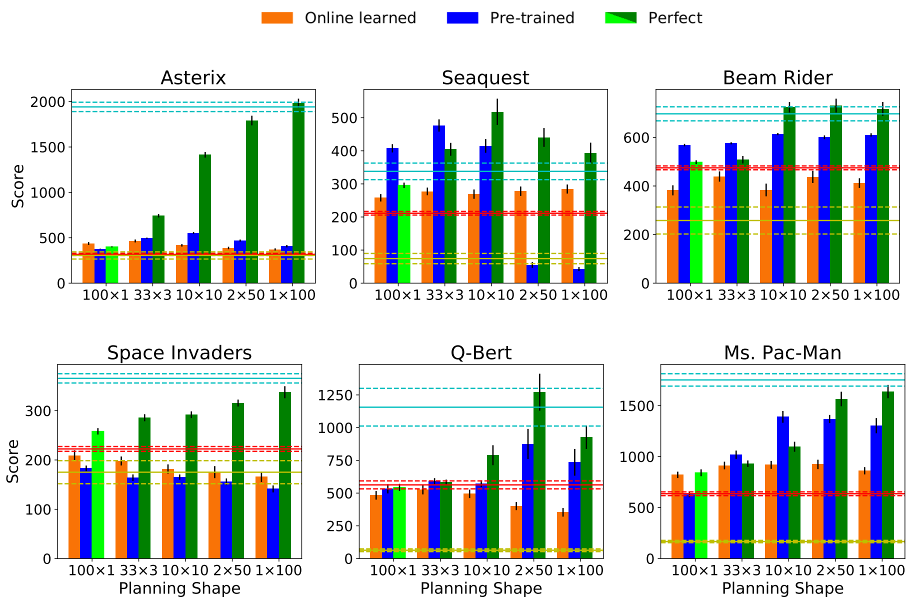

Figure 1: The results of Dyna-DQN with the perfect and imperfect models on six games from the ALE.

2 The Ineffectiveness of One-step Rollouts

Dyna-style planning uses a model to make predictions of future states based on a start state and action. When the state is high dimensional, like an image, it is not clear how to generate reasonable start states. One solution is to sample the start state from the previously observed states, as in Dyna-Q (Sutton, 1990). The advantage of selecting states in this way is that the distribution or structure of the state space does not need to be known, but planning may not be guaranteed to cover the state space entirely. To explore this strategy we experiment with Dyna in several Atari games.

Dyna is a meta-algorithm. To instantiate it, one must make specific choices for the model and the value function/policy learner. In the experiments in this section, the agent uses a perfect model. This allows us to investigate best case planning performance and focus on the question of how best to use the model. Later we incorporate imperfect, learned models.

We experimented with agents based on two different value function learners. One was based on the Sarsa algorithm (Rummery & Niranjan, 1994; Sutton, 1996), using linear value function approximation with Blob-PROST features (Liang et al., 2016), which have been shown to perform well in the ALE. The other was based on Deep Q-Networks (DQN) (Mnih et al., 2015), which approximates the value function using a deep convolutional neural network. Qualitatively, the results were very similar for both agents, suggesting that our conclusions are robust to different concrete instantiations of the Dyna architecture. Here we focus on the results from the DQN-based agent, that we call Dyna-DQN, on six games.

In these experiments, the agent used a perfect copy of the emulator for its model. Start states for planning were selected from the planning buffer containing the 10,000 most recent real states observed by the agent, which for all games was multiple episodes of experience. For each real step, Dyna-DQN drew 100 start states from the buffer and simulated a single transition from each. Dyna-DQN was trained for 100k real frames, or equivalently 10M combined model and real frames. The training frequency was every 4 steps of real and model experience. After training, the mean score in 100 evaluation episodes using a fixed $\epsilon = 0.01$ was recorded. This training and evaluation procedure was repeated for thirty independent runs. The mean scores and standard errors for the six games are shown in Figure 1. The bright green bars labelled 100×1 represent Dyna-DQN.

To better evaluate the benefit of model-based updates, we also compared to the following model-free DQN baselines (pictured as horizontal lines in Figure 1).

DQN 100k: DQN trained only for 100k real frames (yellow lines). This allows us to compare DQN and Dyna-DQN with an equivalent amount of real experience. This benchmark serves as a sanity check to show that using the perfect model to gather additional data does improve sample complexity.

DQN Extra Updates: DQN trained for 100k real frames, but with the same number of updates to the value function as Dyna-DQN (red lines). For each time DQN would normally perform a single training step, DQN Extra Updates performs 100 training steps. This way DQN Extra Updates is like Dyna-DQN, but it uses only experience gathered from the environment, while Dyna-DQN also generates experience from the model. DQN Extra Updates allows us to isolate the advantage of using the model to generate new experience compared to doing more updates with the real experience.

DQN 10M: DQN trained for 10M frames (cyan lines). This allows us to compare DQN and Dyna-DQN with an equivalent amount of total experience. We might hope the experience generated by a perfect model would allow Dyna-DQN to perform comparably to this baseline.

In every game excluding SEAQUEST, Dyna-DQN (100×1) provided little benefit over DQN Extra Updates, even with a perfect model. This indicates that most of the benefit of planning was from simply updating the value function more often, which does not require a model. In other words, the extra computation required by Dyna to utilize the model was not worth the effort. It seems that planning in this way — taking a single step from a previously visited state — does not provide data that is much different than what is already contained in the experience replay buffer. If true, a strategy is needed to make the data generated by the model different from what was already experienced.

3 Planning with Longer Rollouts

We hypothesize that it may be possible to generate more diverse experience, and thus improve learning, by rolling out more than a single step from the start state during planning. Since the current policy will be used for the rollout, the model may generate a different trajectory than what was originally observed. Longer rollouts would also allow the agent to see the longer-term consequences of exploratory actions or alternative stochastic outcomes.

It is straightforward to modify Dyna-DQN so that instead of rolling out a single step, the model is used to roll out k steps from each start state, producing a sequence of k states and rewards, which are all placed in the experience replay buffer. Let this algorithm be called Rollout-Dyna-DQN. When $k = 1$ we recover exactly Dyna-DQN.

Given a budget of planning time in terms of a fixed number of model prediction steps, planning could take on a variety of shapes. Let the planning shape be described by the notation $n \times k$, where n is the number of start states. For example: 100 rollouts of 1 step (100×1); 10 rollouts of 10 steps (10×10); or 1 rollout of 100 steps (1×100), each require the same amount of computation from the model. The experience generated during a multi-step rollout is still recorded as single transitions in the replay buffer, to be sampled independently during a DQN training step. Furthermore we hold the total number of transitions drawn from the model constant. Thus the only difference between two planning shapes is the distribution of transitions in the replay buffer.

To investigate the effects of planning shape we varied the planning shape for Dyna-DQN. We trained and evaluated 100×1, 33×3, 10×10, 2×50, and 1×100 planning using the same experimental setup. The results for the six games are shown in dark green in Figure 1. Note that the ratio of real to simulated transitions remains the same in each case.

In every game, 100×1 planning achieved the worst performance whereas longer rollout lengths allowed Rollout-Dyna-DQN to significantly outperform DQN Extra Updates and approach the performance of DQN 10M. The only difference between two planning shapes is the distribution of experience generated by the model. Thus, our results suggest that for Dyna to make the most of the model, it is critical that the model be used to generate sufficiently novel experience, and generating multi-step rollouts appears to be a simple and effective strategy for accomplishing this. Doing longer rollouts during planning makes using the model worth the effort whereas the 100×1 planning is often no better than doing extra updates with only real experience.

4 Planning with a Learned Model

Next we repeated the experiment with an imperfect model, trained using the deep convolutional neural network architecture introduced by Oh et al. (2015), but extended to predict the next reward as well as the next frame. For the first set of experiments with a learned model, we pre-trained the model with expert data. Then, holding the model fixed, we repeated the experiment above, measuring Rollout-Dyna-DQN's performance with various planning shapes. The results of applying Rollout-Dyna-DQN with the imperfect, pre-trained model are shown in Figure 1 (blue bars).

In every game except SPACE INVADERS, we see evidence that Rollout-Dyna-DQN with a learned model can (modestly) outperform DQN Extra Updates and furthermore that rollouts longer than one step provided the most benefit. The reason the performance was especially poor in SPACE INVADERS is that the model had trouble predicting bullets, which is fundamental to scoring points. Oh et al. (2015) attribute this flaw to the low error signal produced by small objects, which can make it difficult to learn about small details in the image. In SEAQUEST the learned model curiously seems to sometimes outperform the perfect model. Recall that the learned model is trained on expert data — perhaps it is overfitting in a way that happens to be beneficial to planning.

Finally, we study Rollout-Dyna-DQN in the case where the model is learned alongside the value function. The results are shown in Figure 1 (orange bars). Learning the model and value function together adheres to the original conception of Dyna, and also allows us to investigate whether learning a model can improve sample efficiency in this context.

As with the perfect and pre-trained models, the best performance was achieved using rollouts longer than one step (except in SPACE INVADERS). Furthermore, though performance with the online-learned model is generally worse than with the pre-trained model, in three games (ASTERIX, SEAQUEST, and MS. PAC-MAN), it was consistently above DQN Extra Updates. This demonstrates that in some cases there is a modest sample efficiency advantage to learning and planning with a dynamics model online over re-using the agent's prior experience. To our knowledge, this is the first time that this has been demonstrated in the ALE, which has proven to be very challenging for model-based approaches (Machado et al., 2017). While this specific instantiation of Dyna is not a plausible competitor to successful model-free agents in this domain, our results suggest that Dyna-style planning is a viable approach that, with further study, may lead to more successful model-based agents. Importantly, the key finding of this paper is that, even if model quality is significantly improved, planning shape must be a consideration in order to get the most out of the learned model.

5 Conclusions

We found that Dyna is a promising approach for model-based RL in high-dimensional state spaces (achieving, for the first time, a sample efficiency benefit in the ALE from planning with a learned dynamics model). However, we also found that the benefit of Dyna-style planning is limited when model-generated experience takes only one step away from the agent's stored experience, as in the original Dyna-Q. To get the most value from model-based updates, the model must be used to general novel experience. We found that using the model to generate fewer, longer rollouts was an effective way to achieve this. This finding seems to be robust across different concrete instantiations of Dyna — our results indicate that planning shape has a notable impact on the benefit of planning with different value learners, multiple pre-trained models, and a model learned online along with the value function.

References

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

Bellemare, M. G., Veness, J., & Bowling, M. (2013). Bayesian learning of recursively factored environments. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 1211–1219). Atlanta, Georgia, USA: PMLR.

Bellemare, M. G., Veness, J., & Talvitie, E. (2014). Skip context tree switching. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st international conference on machine learning* (Vol. 32, pp. 1458–1466). Bejing, China: PMLR.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *AAAI conference on artificial intelligence.*

Liang, Y., Machado, M. C., Talvitie, E., & Bowling, M. (2016). State of the art control of Atari games using shallow reinforcement learning. In J. Thangarajah, K. Tuyls, S. Marsella, & C. Jonker (Eds.), *Proceedings of the 15th international conference on autonomous agents and multiagent systems (AAMAS 2016).*

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

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., & Bowling, M. (2017). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, *abs/1709.06009*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

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

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. (2015). Action-Conditional Video Prediction using Deep Networks in Atari Games. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 2845–2853). Curran Associates, Inc.

Oh, J., Singh, S., & Lee, H. (2017). Value prediction network. In I. Guyon et al. (Eds.), *Advances in neural information processing systems 30* (pp. 6118–6128). Curran Associates, Inc.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Tech. Rep.). CUED/F-INFENG/TR 166, Engineering Department, Cambridge University.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In B. Porter & R. Mooney (Eds.), *Machine learning proceedings 1990* (pp. 216 – 224). San Francisco (CA): Morgan Kaufmann.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky & M. E. Hasselmo (Eds.), *Advances in neural information processing systems 8* (pp. 1038–1044). MIT Press.

Tamar, A., Wu, Y., Thomas, G., Levine, S., & Abbeel, P. (2016). Value iteration networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29* (pp. 2154–2162). Curran Associates, Inc.

Weber, T., Racanière, S., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., ... Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In I. Guyon et al. (Eds.), *Advances in neural information processing systems 30* (pp. 5690–5701). Curran Associates, Inc.