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

G. Zacharias Holland [1] **Erik Talvitie** [2] **Michael Bowling** [1]

Abstract

Dyna is an architecture for reinforcement learning agents that interleaves planning, acting, and learning in an online setting. Dyna aims to make fuller use of limited experience to achieve better performance with fewer environmental interactions. In Dyna, the environment model is typically used to generate one-step transitions from selected start states. We applied one-step Dyna to several games from the Arcade Learning Environment and found that the model-based updates offered little benefit, even with a perfect model. However, when the model was used to generate longer trajectories of simulated experience, performance improved dramatically. This observation also holds when using a model that is learned from experience; even though the learned model is flawed, it can still be used to accelerate learning.

1. Introduction

Reinforcement learning (RL) has seen much recent success on a variety of complex problems with high-dimensional state spaces due to its integration with deep-neural networks — a field of research known as deep RL. An important testbed for deep RL algorithms is 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. A multitude of methods exist for playing games from the ALE (e.g., Mnih et al., 2015; Mnih et al., 2016; Hessel et al., 2018), and 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 make use of an explicit environment model to perform planning, which in this sense is any process that takes a model and uses it to produce or improve a policy. Although they are generally more complex than model-free methods, model-based methods have been shown to learn a good policy with fewer environmental interactions, making better use of limited experience (e.g., Sutton, 1990; Moore & Atkeson, 1993). This is especially important for domains where interacting with the environment and collecting real experience is expensive.

Although most demonstrations of the benefits of model-based methods have been on simple problems, there have been several recent approaches applied to higher-dimensional problems (Tamar et al., 2016; Oh et al., 2017; Weber et al., 2017). However, all of these approaches use the learned model solely to provide additional input when selecting actions. Part of the allure of model-based RL is that a model could also be used as a substitute for gathering more experience in the real world.

Dyna is an architecture for RL agents that combines aspects of both model-free and model-based RL in a flexible way. The model is used to generate simulated experience alongside the real experience observed from the environment. The real and simulated experience are both used in the same way — to update the agent's value function and/or policy.

To investigate Dyna in a domain with a high-dimensional state space we evaluated its performance on several games from the ALE. Surprisingly, we found that Dyna-style planning provides almost no benefit over simply doing more updates with the real data already collected by the algorithm. It is only when the model is used to produce multi-step rollouts — sequences of more than one state — does the additional computation required for planning become beneficial. Our empirical results show that rollout length is a key factor in the effectiveness of Dyna-style planning.

2. Dyna and The Ineffectiveness of One-step Planning

One of the first instantiations of the Dyna architecture was Dyna-Q (Sutton, 1990). Dyna-Q combines one-step tabular Q-learning (Watkins, 1989; Watkins & Dayan, 1992), with a model that, from a given start state, can predict the next state and reward. After taking a step in the environment and

[1]Department of Computing Science, University of Alberta, Edmonton, AB, Canada [2]Department of Computer Science, Franklin & Marshall College, Lancaster, PA, USA. Correspondence to: Zacharias Holland <gholland@ualberta.ca>.

Accepted at the FAIM workshop "Prediction and Generative Modeling in Reinforcement Learning", Stockholm, Sweden, 2018. Copyright 2018 by the author(s).

updating the value function with the conventional model-free Q-learning update, the following planning procedure is repeated n times: sample a start state, S, uniformly at random from the set of previously seen states; sample an action, A, uniformly from the the set of actions previously chosen in S; use the model to predict the next state, S' and reward, R; and update the value function using S, A, S', and R with the same Q-learning update. To explore the effectiveness of Dyna in problems with high-dimensional state spaces, we extend DQN to incorporate Dyna-style planning. We then evaluate it in several Atari games.

2.1. Deep Q-networks and Dyna-DQN

Deep Q-networks (DQN) (Mnih et al., 2015) is a model-free deep RL method, based on Q-learning, that uses a deep convolutional neural network to approximate the value function. Unlike Q-learning, DQN does not update the value function after every step using a single transition; instead, DQN uses experience replay (Lin, 1992), and places each observed transition into an experience replay buffer. Then, for a single training step, the algorithm selects a mini-batch of transitions from the experience buffer uniformly at random to update the parameters. Training steps are performed after every f observed transitions.

It is straightforward to extend DQN to use the Dyna architecture. After every step taken in the environment, simulated experiences are generated starting from a state sampled from a planning buffer containing the agent's recent real experience. Keeping a separate buffer ensures that start states are always from the agent's actual experience. The simulated experiences are placed into the experience replay buffer alongside the transitions observed from the real environment, and training continues to happen after every f observed transitions — real or simulated. As a result, mini-batches sampled at training time will contain a mix of real and simulated experience.

In the following experiments we explore idealized performance of Dyna-DQN by assuming a perfect model is available. This isolates the effects of planning from the accuracy of the model and provides an estimate of the maximum benefit that the model can provide in Dyna-DQN.

2.2. Experiments

We ran experiments on six games from the ALE and chose to study the games from the original training set outlined by Bellemare, Naddaf, et al. (2013), supplemented with two additional games, Q-BERT and MS. PAC-MAN, that Oh et al. (2015) used to evaluate their model learning approach, which we employ in Section 4. We have omitted results in FREEWAY since our implementations of DQN and Dyna-DQN almost always score zero points at the number of training frames we used.

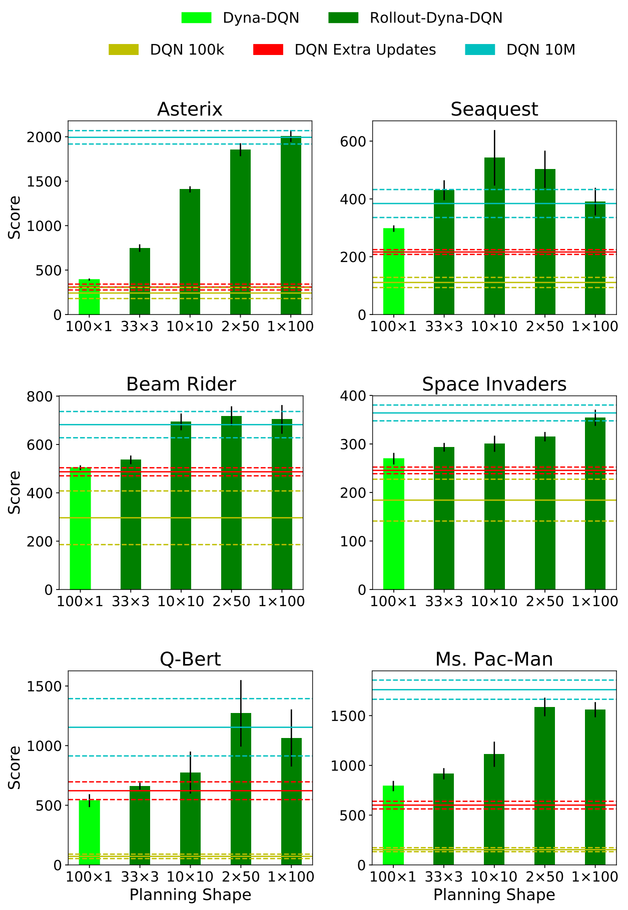

Figure 1. The results of running Dyna-DQN and Rollout-Dyna-DQN on six games from the ALE compared to the DQN baselines. Dyna-DQN provides almost no benefit over simply doing more updates with the same amount of data from the environment. The performance of Rollout-Dyna-DQN tends to increase as the rollout length increases across all the games.

In these experiments, Dyna-DQN made use of an environment model that was a perfect copy of the emulator. 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 did 100 iterations of planning. Dyna-DQN was trained for 100k real frames, or equivalently 10M combined model and real frames. After training, the mean score in 100 evaluation episodes was recorded. This training and evaluation procedure was repeated for ten independent runs. The mean scores and standard errors for the six games are shown in Figure 1 (bright green bars are Dyna-DQN)[1]. To better understand the benefit of model-based updates, we also compared to the following

[1]Additional details for all the algorithms and experiments described in this paper are available in a longer version (https://arxiv.org/abs/1806.01825).

model-free DQN baselines.

DQN 100k: DQN trained for 100k real frames (yellow lines). This allows us to compare DQN and Dyna-DQN with an equivalent amount of real experience. As one might expect, Dyna-DQN outperformed DQN 100k; it uses the model to generate more experience and does more updates.

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 evaluate the advantage of using the model to generate new experience compared to simply doing more updates with the real experience. Surprisingly, Dyna-DQN provided little benefit over DQN Extra Updates, even with a perfect model.

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; however, in most games the performance did not approach that of DQN 10M.

Overall, we find that the extra computation required by Dyna-DQN to utilize the model does not appear to be worth the effort. A possible explanation for these results is 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 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

One possible strategy to generate more diverse experience is to roll out more than a single step from the start state during planning like Gu et al. (2016) and Kalweit and Boedecker (2017). Since the current policy will be used for the rollout, the model may generate a different trajectory than what was originally observed.

It is straightforward to modify each planning iteration of Dyna-DQN so that instead of rolling out a single step, the model is used to rollout k steps from the start state, producing a sequence of k states and rewards. Let this algorithm be called Rollout-Dyna-DQN. When $k = 1$ we recover exactly Dyna-DQN described in Section 2.1.

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 planning iterations. 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, but the way that the resulting predictions are distributed in the state space are different. In the next section we investigate the effects of planning shape on the performance of Rollout-Dyna-DQN compared to the DQN baselines.

3.1. Experiments

Our experimental setup is the same as in Section 2.2, but now the planning shape for Dyna-DQN is varied. As discussed above, the bright green bars in Figure 1 correspond to planning shape 100×1. The dark green bars in Figure 1 show the performance for planning shapes 33×3, 10×10, 2×50, and 1×100. Note that the ratio of real transitions to simulated transitions remains the same in each case.

In each game there was a longer rollout length that resulted in a dramatic improvement over 100×1 planning, significantly outperforming DQN Extra Updates. Further, in every game, there was a planning shape that approached the performance of DQN 10M.

Our results suggest that with the Dyna architecture it is critical for the model to generate sufficiently novel experience, and using multi-step rollouts appears to be an effective strategy. Doing longer rollouts during planning makes using the model worth the effort whereas the 100×1 planning is no better than doing extra updates with only real experience.

4. Planning with an Imperfect Model

In the previous sections we have drawn conclusions from an ideal setting, but if agent has an imperfect model do the same conclusions hold? To investigate this question we replaced the perfect copy of the emulator with a learned model pre-trained on data from expert play.

4.1. Experiments

For this section, the experimental setup is the same as in Sections 2.2 and 3.1, but the perfect model has been replaced with an imperfect model. For the imperfect model, we made use of the deep-neural network architecture introduced by Oh et al. (2015). This model was shown to make visually accurate predictions for hundreds of steps on video input from Atari games conditioned on actions. In its original formulation the model predicts only the next state, but an environment model for reinforcement learning needs to predict both the next state and the next reward. Therefore, we extended the model to make reward predictions similarly to Leibfried, Kushman, and Hofmann (2017).

We trained and evaluated three different models with Rollout-Dyna-DQN to ensure that any trends that were ob-

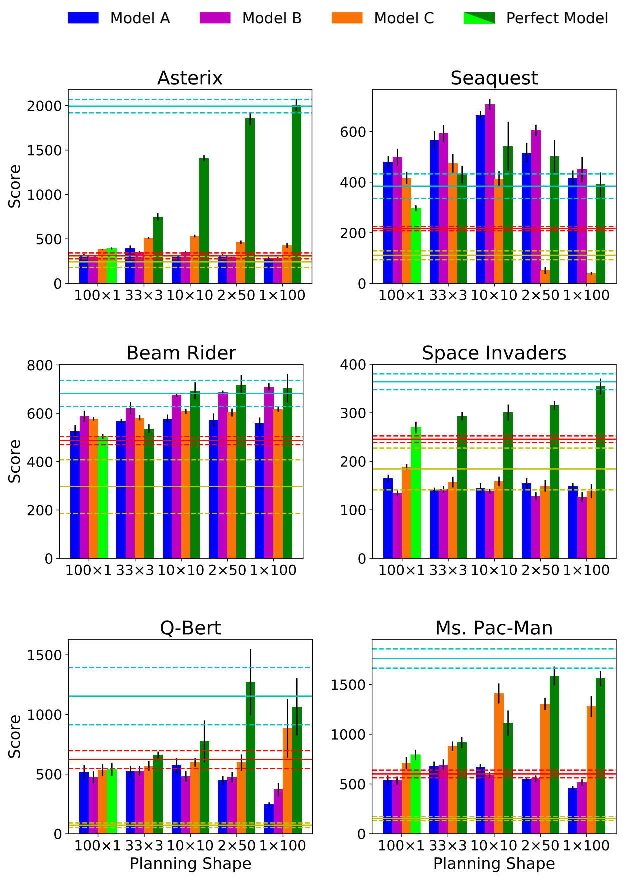

Figure 2. The results of running Rollout-Dyna-DQN with the perfect and imperfect models on six games from the ALE. Like the perfect model, using an imperfect model with a rollout length greater than one provides the most benefit. The horizontal lines show the baseline scores described in Section 2.2.

served were not specific to a particular model. Note that because each model is the result of a single training run on a single dataset, our results cannot be used to draw reliable conclusions about the comparative effectiveness of the different model training regimes. Our aim in this experiment is only to study the impact of model error on Rollout-Dyna-DQN. As such, we refer to the models merely as Models A, B, and C. The results of applying Rollout-Dyna-DQN with the three imperfect models are shown in Figure 2. The perfect model results and baselines are the same as in Figure 1.

As with the perfect model, rollouts longer than one step provided the most benefit. For every game, except for SPACE INVADERS, there was a model and planning shape that performed better than DQN Extra Updates, which demonstrates that even when the model has flaws, planning with rollouts can provide some benefit.

Now that the model is imperfect, it is interesting to observe that in most game and model combinations the best-performance was achieved at medium rollout length. There is a limit on how far the model can roll out before small errors compound and make the predictions unreliable (e.g. Talvitie, 2014). Thus, there is a trade-off between improved planning and increased model error as rollout length is increased. For example, in ASTERIX using Model C, the performance peaked at 10×10 planning and dropped off as rollouts became shorter or longer.

5. Discussion

Despite the introduction of increasingly effective approaches for learning predictive models in Atari Games (Bellemare, Veness, & Bowling, 2013; Bellemare, Veness, & Talvitie, 2014; Oh et al., 2015), to our knowledge this is the first time that a learned dynamics model has been successfully used for planning in this challenging domain. Our results show that, combined with deep RL methods, Dyna is a promising approach for model-based RL in high-dimensional state spaces and that planning shape is a critical consideration in extracting the most benefit from the model. In every game from the ALE we tested, there was a planning shape with a rollout length greater than one that outperformed DQN Extra Updates, and 100×1 planning. Longer planning rollouts appears to be an effective strategy for generating novel experience, which seems to be necessary to use the model to its full potential.

Our findings suggest multiple next steps. In our experiments we pre-trained the models; clearly it would be interesting to study Rollout-Dyna-DQN in the case where the model is learned online alongside the value function. This would necessarily involve careful consideration of how to best use the model early in training, while its predictions may be highly unreliable. Finally, though we found longer rollouts to be an effective way to use the model to generate experience, there are other promising approaches. For instance Pan et al. (2018) and Goyal et al. (2018) use inverse dynamics models to effectively propagate value updates backwards in a manner similar to prioritized sweeping (Moore & Atkeson, 1993; Peng & Williams, 1993). It may be possible to combine these insights, exploiting a forward model's ability to reveal novel states and a backward model's ability to efficiently improve the value function.

Acknowledgements

The authors thank Marlos Machado and Jesse Farebrother for their help with the implementation of DQN. This research was funded by NSF grant IIS-1552533, NSERC, Alberta Innovates, and the Alberta Machine Intelligence Institute (amii), and was enabled in part by support provided by WestGrid (westgrid.ca), CalculQuébec (calculquebec.ca), and Compute Canada (computecanada.ca).

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.

Goyal, A., Brakel, P., Fedus, W., Lillicrap, T. P., Levine, S., Larochelle, H., & Bengio, Y. (2018). Recall traces: Backtracking models for efficient reinforcement learning. *CoRR*, *abs/1804.00379*. Retrieved from http://arxiv.org/abs/1804.00379

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous deep Q-learning with model-based acceleration. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (Vol. 48, pp. 2829–2838). New York, NY, USA: 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.*

Kalweit, G., & Boedecker, J. (2017). Uncertainty-driven imagination for continuous deep reinforcement learning. In S. Levine, V. Vanhoucke, & K. Goldberg (Eds.), *Proceedings of the 1st annual conference on robot learning* (Vol. 78, pp. 195–206). PMLR.

Leibfried, F., Kushman, N., & Hofmann, K. (2017). *A deep learning approach for joint video frame and reward prediction in atari games.* Presented at the ICML 2017 Workshop on Principled Approaches to Deep Learning. Retrieved from http://arxiv.org/abs/1611.07078

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

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.

Pan, Y., Zaheer, M., White, A., Patterson, A., & White, M. (2018). *Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains.* To appear at the 27th International Joint Conference on Artificial Intelligence. Stockholm, Sweden.

Peng, J., & Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, *1*(4), 437–454.

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.

Talvitie, E. (2014). Model regularization for stable sample rollouts. In *Proceedings of the 30th conference on uncertainty in artificial intelligence* (pp. 780–789).

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.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* (Unpublished doctoral dissertation). Cambridge University.

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

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.