Knowing When To Look Back: Bidirectional Rollouts in Dyna-style Planning

Yat Long Lo, [1, 3] Jia Pan, [1] Albert Y.S. Lam [2, 3]
[1]Department of Computer Science, University of Hong Kong
[2]Department of Electrical and Electronic Engineering, University of Hong Kong
[3] Fano Labs, Hong Kong

Abstract

In model-based reinforcement learning (MBRL), a model of the world is used to generate transitional data for an agent to learn from, in order to reduce sample complexity. Dyna is one of the most widely adopted MBRL frameworks that perform planning, acting, and learning in an online manner. Most of the previous works on Dyna perform one-step rollouts from sampled states to generate data based on the agent's history. Recently, it has been shown that planning shape and directionality (forward or backward planning) of the rollouts can impose a significant impact on the performance given a fixed planning budget. In this work, we conduct a systematic study on how these two factors affect the performance of model-based agents. We hypothesize that forward planning and backward planning serve complementary purposes, i.e. exploration and value propagation, in which careful state-dependent allocation of planning budget can improve learning efficiency. We further provide an online method to automate the decision between forward planning and backward planning using error-based epistemic uncertainty. We examine our proposed method in the tabular and linear function approximation settings for both perfect and learned models on GridWorld and Cartpole environments and propose the use of an ensemble of world models to counter compounding errors of long rollouts in the learned models. Our results show that both planning shape and directionality have a profound impact on Dyna methods' efficacy and bidirectional rollouts can improve learning efficiency using the same number of planning steps.

1 Introduction

With numerous successes of model-free reinforcement learning (RL) in various tasks from video game playing (Volodymyr et al. 2015) to robotics control (Gu et al. 2017), model-based RL has gained wide interest in the research community as a means to reduce the number of interactions with the environment, i.e., sample complexity, which can be expensive when being applied to real-world problems. Model-based RL commonly contains a model of the world that can be either hand-crafted or learned. Akin to how humans imagine scenarios in our heads, an agent learns from the data generated by the world model without actually taking actions within the real environment. The learning process from these generated data is known as planning.

Dyna (Sutton 1991) is a general framework of reinforcement learning that combines both model-free and model-based RL into a single coherent architecture. This framework has the appealing property of asynchronous processing of planning and learning in which an agent's decision-making and learning processes can operate at the same time as the world model is learning and planning. The world model is learned from an agent's real experience while both real and simulated experiences generated by the world model are used to update the value function or policy. We refer to the use of simulated experience to learn here as Dyna-style planning. Given the general and learning-algorithm-independent nature of Dyna, we study different planning properties following this framework.

Dyna offers flexible control over the planning process by how much computational budget is devoted to planning and how is the budget distributed (Holland, Talvitie, and Bowling 2018). Most of the previous works on Dyna, including the original Dyna-Q algorithm (Sutton 1991), performed one-step rollouts from sampled states to perform planning. The potential benefits of using such flexible control are less investigated. Two major factors over such planning control are planning shape (Holland, Talvitie, and Bowling 2018) and directionality. Holland, Talvitie, and Bowling (2018) studied the impact of planning shape on performance, demonstrating that longer or mid-length rollouts are often more beneficial than one-step rollouts for the same amount of planning steps. In terms of directionality, forward planning is the most common that the world model generates the reward and the next state given a sampled state and action. Moore and Atkeson (1993) used a form of backward planning by prioritizing states of high error and updating the values of their predecessors, which was shown to improve learning efficiency by propagating values along a trajectory faster. Edwards, Downs, and Davidson (2018) proposed forward-backward RL that uses a backward world model to generate data starting from goal states to handle sparse reward environments, but it assumed the knowledge of goal states and only performed rollouts from those goal

states. Overall, to our best of our knowledge, there is no systematic study on both planning shape and directionality and methods that take advantage of both forward and backward planning.

With a focus on planning shape (i.e., rollout lengths), we encounter the issue of compounding of errors (Asadi, Misra, and Littman 2018) in longer rollouts. The issue originates from the inaccuracy of generated data by learned models that can detrimentally affect performance. This is exacerbated when the rollouts are long, outweighing any benefits the rollout lengths can bring. To mitigate this problem, Talvitie (2014) and Venkatraman, Hebert, and Bagnell (2015) proposed hallucination to prepare the model to handle the fake and inaccurate inputs generated by itself. Asadi et al. (2019) proposed using a multi-step model directly to avoid feeding generated outputs back to the world model.

In this paper, we first conduct a systematic study on how both planning shape and directionality affect performance. Then, we hypothesize that forward planning and backward planning serve complementary roles of exploration and value propagation. We propose an automatic method for bidirectional rollouts that takes advantage of their hypothesized roles using error-based epistemic uncertainty. For the same amount of planning budget, we show that our proposed method for bidirectional rollouts produces the best overall performance among different planning shapes when compared with only-forward rollouts and only-backward rollouts, offering supporting evidence to our hypothesis for roles of forward planning and backward planning. We test our method in the tabular and linear function approximation settings for both perfect and learned models on GridWorld and Cartpole environments. Additionally, we propose the use of an ensemble of world models to counter compounding errors of long rollouts in learned models that drop inaccurate data based on the degree of disagreement among them. The simple and straightforward approach is found to be effective in avoiding catastrophic failures in long rollouts.

2 Background

2.1 Reinforcement Learning

In reinforcement learning, an agent interacts with its environment by taking actions at discrete time steps $t = 0, 1, 2, \cdots$. The environment is commonly formulated as a Markov Decision Process (MDP) with states $\mathcal{S}$, actions $\mathcal{A}$, transition probabilities $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, rewards $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ and discount function $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ (White 2017). At each time step t, the agent is in a state S_t, and takes an action A_t. In response, the environment emits a reward R_{t+1} and takes the agent to a state S_{t+1}. The goal of the agent is to maximize the return, defined as the discounted sum of the cumulative rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1}$$

In this paper, as we focus on the rollout mechanisms in Dyna-style planning, the methods used should be independent of the choice of learning algorithms. Thus, we use Q-learning (Watkins and Dayan 1992) as our default learning algorithm to isolate the effect of the planning parameters that we are interested in. In Q-learning, the agent learns to approximate the state-action value function and acts near-greedily according to those state-action values. The state-action values for a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ are the expected return for that policy beginning from state s and action a:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a], \tag{2}$$

where $\mathbb{E}_\pi[\cdot]$ denotes taking expectation under policy π.

For linear function approximation, We parameterize the state-action value functions, denoted as $\hat{q}(S_t, A_t, \boldsymbol{\theta}_t)$, where $\boldsymbol{\theta}_t$ refers to the weight vectors. State-action value functions are commonly learned by bootstrapping the state-action value of the next state, minimizing the temporal difference error. The update rule for Q-learning to learn state-action value functions is as follows:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma \max_{A_{t+1}} \hat{q}(S_{t+1}, A_{t+1}) \\ - \hat{q}(S_t, A_t)) \nabla_{\boldsymbol{\theta}} \hat{q}(S_t, A_t), \end{aligned} \tag{3}$$

where α is the learning rate. Each linear function approximator takes in a state feature vector as an input and produces state-action value for each possible action.

2.2 Model-Based Reinforcement Learning

Dyna Learning the value function often requires a huge number of samples. Model-based approaches aim to lower such sample complexity using world models to generate imagined experiences. Dyna-Q (Sutton 1991) learns a value function from both real and imagined experiences using the same update rule as Q-learning. Specifically, a world model is used to generate a complete transition to be used by a learning algorithm. The world models are either handcrafted or learned. Here, we parameterize learned world models with neural networks.

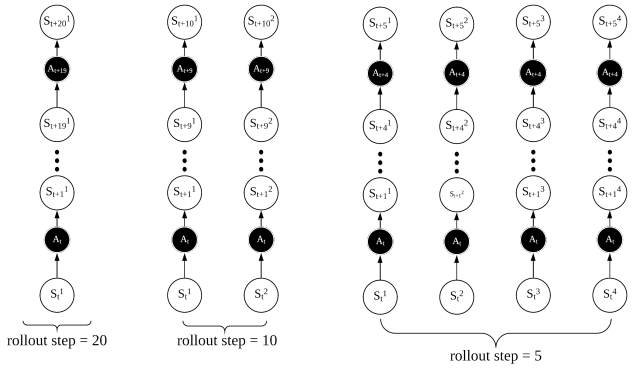

Figure 1: Different planning shapes

Planning Shape Planning shape is a term coined by Holland, Talvitie, and Bowling (2018) to formalize how a planning budget is distributed. The budget refers to the number of planning steps per real step taken in the environment. The

distribution can take many shapes. For instance, as illustrated in Figure 1, if the planning budget is 20 steps, an agent can sample one state for 20 rollout steps, sample two states for 10 rollout steps, sample four states for 5 rollout steps, and so on. It is found that the distribution can have a profound impact on the performance of a Dyna agent. Specifically, 1-step rollouts do not seem to benefit performance when compared with longer rollouts.

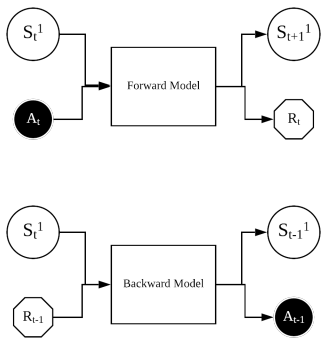

Figure 2: Forward and backward models

Directionality As shown in Figure 2, forward and backward model have different input-output relationships. Same as the way defined in Sutton and Barto (2018), a forward model takes in a state and an action to produce the corresponding next state and reward. Rollouts can be performed iteratively by feeding the next state and a sampled action into the forward model. On the other hand, a backward model takes in a state and a reward to produce a corresponding previous state and the action that led to the current state. Using either the forward or backward model gives the agent a complete transition to learn but the value of the state that is being updated is different. Note that another possible variant of the backward model is to also predict the reward that an agent gets to reach a particular state. This removes the assumption of the knowledge of a reward function but increases the learning burden of the agent. In this work, we assume that we have a reward oracle that tells the agent the correct reward as it rolls out backward.

3 Bidirectional Rollouts Using Epistemic Uncertainty

Given a fixed planning budget, when should an agent perform forward and backward planning? If we crudely equate learning from data generated by world models with animals' imagination, the question would become when one should look ahead and back from a scenario (a state). We hypothesize that forward and backward plannings serve complementary purposes of exploration and value propagation, respectively. Specifically, forward planning helps improve the value estimation of a state by generating unfamiliar experiences of possible futures. On the other hand, backward planning facilitates propagating the value learned for a particular state back to its predecessors. Hence, as the value estimation of a state-action pair improves through actual environment learning and forward planning, backward planning should be performed to a greater extent to inform value estimations of the state's predecessors. We hypothesize realizing this intuition can improve learning efficiency, especially in sparse reward settings. This is because what is learned would be propagated faster through backward planning at the right time when the value estimation of a state becomes accurate enough.

To know when the estimation is accurate enough, we propose using the simple quantity of the learning error which requires no extra computation to obtain. The learning error has a positive relationship with epistemic uncertainty, which is induced during the learning process. As the error goes down, an agent should be more certain about the value estimation of that state. In other words, simply based on the learning error, we can control how much backward planning to perform. We show how to apply such intuition in both the tabular and function approximation settings.

3.1 Tabular Setting

In the tabular setting, using Q-learning as an example, a simple extension can be done by having a maximum error (ME) table of the same size as the Q-value table to store the maximum error of each state-action pair. With the maximum error known, we have a reference point to the assess learning progress for each pair. To translate such information into resource allocation decisions in terms of how much to roll out forward and backward, we use the maximum error as a normalizing factor to obtain a value between 0 and 1. The allocation equations are as follows:

$$n\_forward = (e_t / ME(S_t, A_t)) \times max\_plan\_steps, \quad (4)$$

$$n\_backward = max\_plan\_steps - n\_forward, \quad (5)$$

where max_plan_steps is the number of planning steps an agent can take, $n\_forward$ and $n\_backward$ are the number of forward rollout steps and the number of backward rollout steps, e_t is the last temporal difference error of the state-action pair, and $ME(S_t, A_t)$ refers to the maximum error of the pair obtained from the maximum error table. We can see that as the error goes down, fewer resources will be allocated to perform forward rollouts with more resources left for backward rollouts to propagate the learned values backward. We hypothesize such a way of allocation would improve learning efficiency.

3.2 Function Approximation Setting

Our simple extension in the tabular setting is not scalable in the function approximation setting, especially in environments with large and continuous state spaces. Specifically, three problems arise, namely the memory issue of the ME table in continuous state spaces, the unreliability of the error measure under the function approximation setting, and the poor quality of learned world models. We propose three techniques to handle these issues

State Discretization: Having a ME table is intractable and inefficient in continuous state spaces. To handle this issue, we propose the use of state discretization to aggregate similar states together. This is done by binning each dimension of the state features separately, denoted as a function $discretize$. The unique combination of bins across all state dimensions would be one entry in the ME table. This method is highly scalable as the number of state dimensions grows linearly with the number of bins (Ghiassian et al. 2020). Hence, when we encounter a large and continuous state space, we no longer need a huge or possibly infinite-sized table but a reasonably sized one after state discretization. With state discretization, the planning budget allocation equation is as follows:

$$\begin{aligned} n\_forward \ &= (e_t / ME(discretize(S_t, A_t))) \\ &\times max\_plan\_steps. \end{aligned} \tag{6}$$

Here, we have an additional $discretize$ function that discretizes the state and returns the corresponding index to access the maximum error value for a subset of states in the ME table.

Exponential Moving Error: Another issue comes from the unreliability of using the learning error as a pseudo-measure of epistemic uncertainty. In the function approximation setting, learning no longer strictly linearly reduces error. An improvement in the value estimation of a state-action pair may worsen the estimation of another pair. Additionally, in some unseen or rarely seen states, or some catastrophic states to the value function, the error values can become exceptionally large. As a result, the relationship between the error and epistemic uncertainty is no longer as linear as we may expect in the function approximation setting. To overcome this, we propose the use of exponential moving error in the replacement of maximum error. By doing so, the error measure used for normalization is less likely to be skewed by large values and is more adaptive to recent learning progress. The update of ME table entries is as follows:

$$\begin{aligned} &ME(discretize(S_t, A_t)) = \\ &\beta \times ME(discretize(S_t, A_t)) + (1 - \beta) \times e_t, \end{aligned} \tag{7}$$

where β is the decaying constant, a hyperparameter to be tuned.

An Ensemeble of World Models: The last issue is the poor quality of world models. As we move towards harder problems beyond the tabular setting, we can no longer have perfect world models. In simple environments like tabular GridWorld, perfect world models can be created, given that we have a deterministic environment with well-defined action space. On the other hand, in harder problems like physics simulators and robotics, it is almost impossible to have a perfect world model, letting alone two perfect world models for forward and backward rollouts. The issue is significantly worse when such a world model is learned. When we use an inaccurate model to perform rollouts, the longer a rollout, the more inaccurate the predictions, caused by the issue of compounding errors (Asadi, Misra, and Littman 2018). To alleviate this issue, we propose the use of an ensemble of world models. The ensemble approach has been a commonly used approach in machine learning. The high-level idea is to improve prediction accuracy by training a set of models that differ by factors like different random initialization to achieve better performance. Osband et al. (2016) proposed the approach of learning an ensemble of value functions that are initialized differently. As learning progresses, the difference in predictions across these functions serves as an indicator of how certain the agent is with the predictions. Taking inspiration from this idea, we propose having an ensemble of learned world models. Their disagreements in the generated rollouts can serve as an indicator of how certain the models are with their predictions. If it is highly uncertain, the agent drops those transitions and does not learn from them. As a result, the agent can avoid learning from data that are highly erroneous with the potential of negatively affecting the learning progress. Specifically, we use the standard deviation of predictions across all the world models as the numerical measure. If the measure is greater than a threshold κ, the rollout is dropped to avoid the compounding of errors. We denote this threshold as κ. When the rollout is not dropped, we use the averaged prediction across the world models as the generated data. More options other than simple averaging like weighted averaging can be further explored.

By combining all these three techniques, we can extend our intuition to the function approximation setting. We will showcase the results in the experimental results section.

4 Experimental Setup

We conduct extensive experiments on the GridWorld environment for the tabular setting and the Cartpole environment for the linear function approximation setting.

For the GridWorld environment, an agent attempts to reach the goal state at the upper-right corner, starting from the lower-left corner. The agent only receives a reward of +1 when reaching the goal state. The state representation is the coordinate of the agent on the grid. GridWorld of various sizes are used to vary reward sparsity: 5x5, 10x10, 20x20, 40x40 and 80x80. For model-based agents, we used a perfect forward and backward world model with a fixed planning budget of 20 steps. To look at the effect of rollout mechanisms, we examine three types of model-based reinforcement learning agents, namely one that only performs forward rollouts (Dyna_Q_Forward), one that only performs backward rollouts (Dyna_Q_Backward) and one that dynamically performs forward and backward rollouts based on the epistemic uncertainty as discussed above (Dyna_Q_Forward_Backward).

For the Cartpole environment, a pole is attached by an unactuated joint to a cart, moving along a frictionless track. An episode starts with the pole upright and the goal of the agent is to prevent the pole from falling by increasing or decreasing the cart's velocity along the horizontal direction. The agent receives a reward of +1 as long as the pole is up and a terminating reward of -1 if the pole falls. The state representation has four features, namely the cart's position, the cart's velocity, the pole's angle, and the pole's velocity

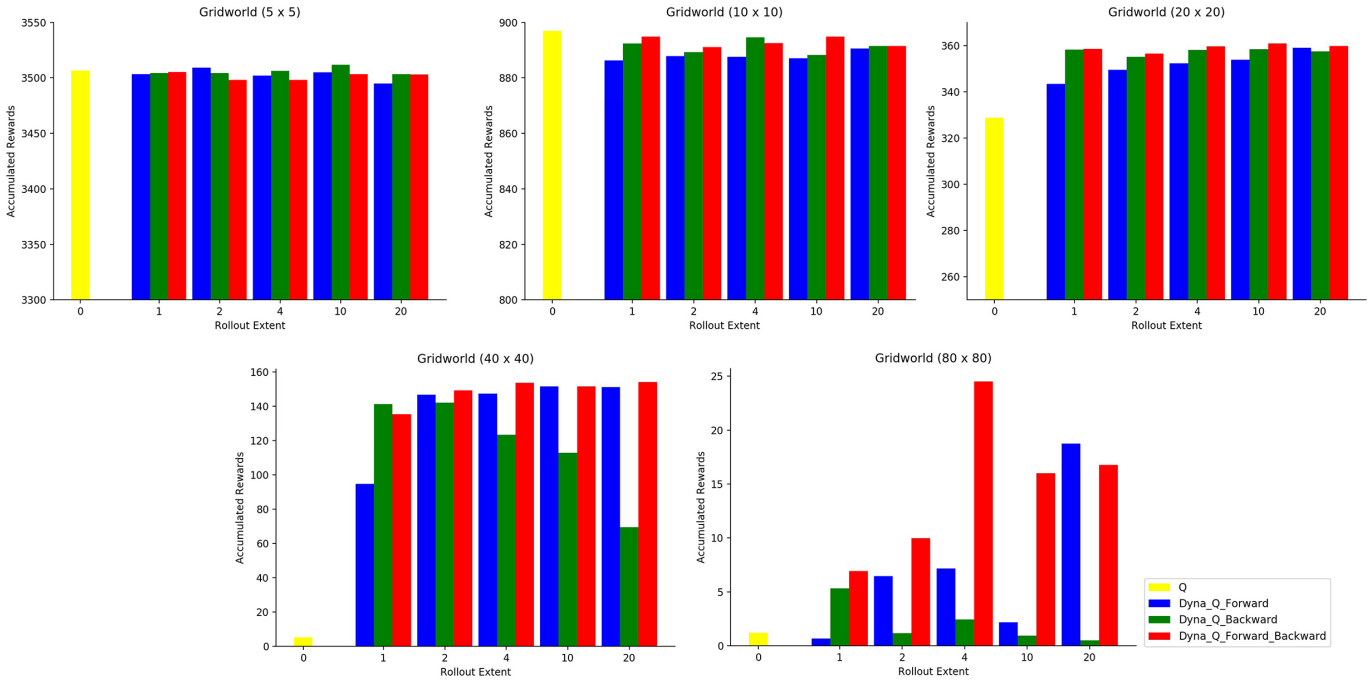

Figure 3: Performance of forward, backward and bidirectional rollouts for GridWorld environments

at the tip. We used the implementation provided by OpenAI (Brockman et al. 2016). To acquire a better feature representation with a higher number of features, we pretrain a feature extractor using Deep Q-learning (Mnih et al. 2015) and adopt the output of the last layer as the inputs to the learning algorithm. We assume imperfect world models by pretraining fully connected neural networks with a fixed planning budget of 20 steps. For the ensemble method, we use an ensemble of six world models for forward models and backward models, respectively. Details on the pretraining of the world models and feature extractor are included in the appendix (7.1 and 7.2) . To assess the performance of our proposed method, we compare with multiple Dyna-Q variants. We consider two algorithms that use only-forward rollouts (Dyna_Q_F) and only-backward rollouts (Dyna_Q_B), respectively, and two other algorithms with the same respective settings but with an ensemble of world model (Dyna_Q_F_DE and Dyna_Q_B_DE). Our proposed method is denoted as Dyna_Q_FB_DE, which uses an ensemble of world model and performs bidirectional rollouts dynamically.

For both sets of experiments, we use model-free Q-learning as our baseline. Parameters sweeps are done on the learning rate, β and κ (see appendix 7.3). Results reported are averaged over 10 runs of different random seeds.

5 Results

5.1 Tabular Setting

In Figure 3, we compare the performance of forward-only (blue), backward-only (green), and bidirectional rollouts (red) in the GridWorld environments of various sizes. To begin with, as problem difficulty increases, we can see model-based agents outperform the baseline Q-learning agent (yellow). This is obvious and expected as model-based agents have additional learning experiences received from their corresponding world models. As the GridWorld size increases, the model-free agent fails to acquire rewards due to the high reward sparsity.

Comparing the two agents that only perform forward and backward rollouts, we can see that as the problem difficulty increases, the agent with only forward rollouts outperforms the agent with only backward rollouts. One possible explanation is the relatively lower capability of exploration using backward rollouts, which prompts the question of whether backward rollouts can be applied conditionally in combination with forward rollouts to achieve even more superior results.

Among the three types of model-based agents, our proposed method with bidirectional rollouts has the best overall performance across GridWorld sizes. More importantly, our proposed method has consistently better performance in GridWorld sizes with larger reward sparsity (20x20, 40x40, and 80x80). This provides supporting evidence of the significant impact of directionality on performance. Specifically, learning efficiency can be improved by utilizing forward and backward planning dynamically based on error-based epistemic uncertainty.

Additionally, looking at the planning shape, our tabular results are in agreement with the conclusions made in Holland, Talvitie, and Bowling (2018) under the nonlinear function approximation setting. We can see that medium-length rollouts are much better than one-step rollouts.

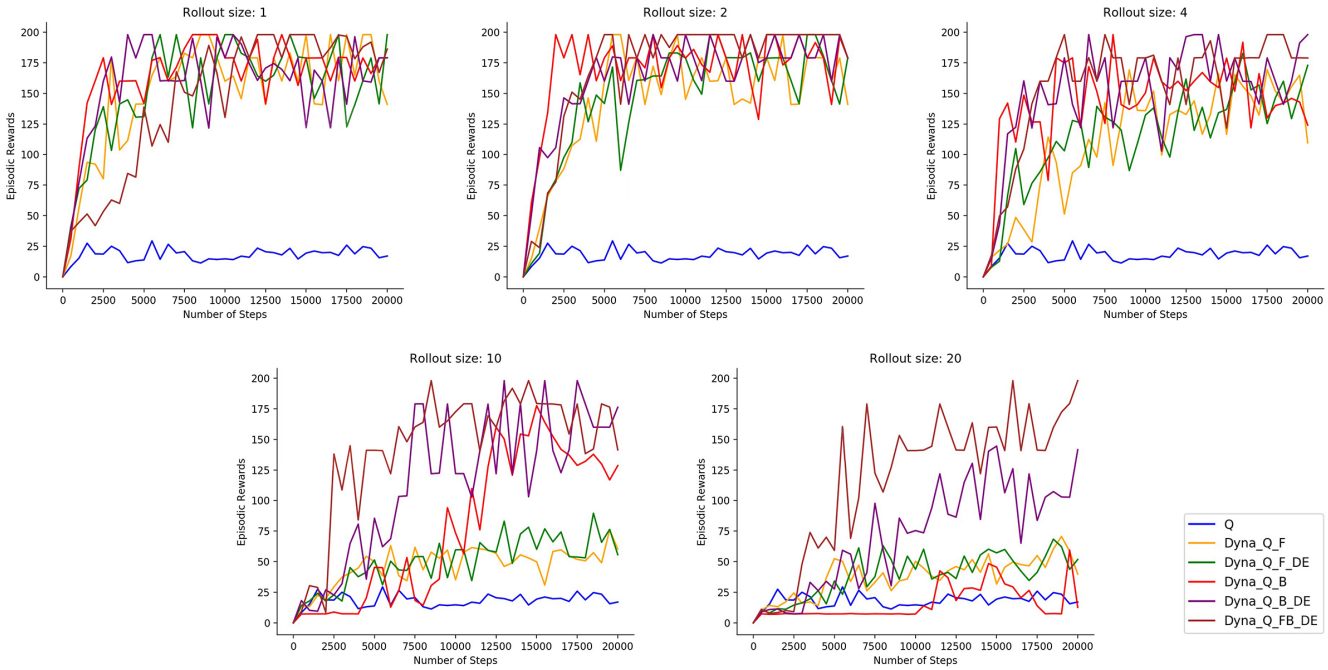

Figure 4: Learning curves of forward, backward and bidirectional rollouts for Cartpole environment

5.2 Function Approximation Setting

In Figure 4, we show the performance of all the mentioned agents across different planning shapes. Being consistent with the results in the GridWorld Experiments, medium-length planning shapes have the best performance for the model-based agents.

We can see that our proposed approach (Dyna_Q_FB_DE) has the best overall results. As the rollout length increases, we observe how an ensemble of world models alleviates the problem of compounding errors by dropping harmful transitions. In other words, the approach is robust when we have imperfect world models and need longer rollouts. The same effect can also be observed in agents that only perform forward or backward rollouts exclusively but with an ensemble of world models (Dyna_Q_F_DE and Dyna_Q_B_DE). By comparing with these two agents, we can see how bidirectional rollouts based on epistemic uncertainty helps with improving learning efficiency. These results in the function approximation setting provide further support to our hypothesis on the differing roles of forward planning and backward planning, and how careful and state-dependent allocation can improve performance.

6 Conclusion and Future Work

In this work, we investigate how the planning shape and directionality of the rollout mechanism affect the performance of a model-based reinforcement learning agent. We hypothesize that if an agent can perform forward and backward plannings dynamically, it can achieve better performance and learning efficiency. We postulate that forward and backward plannings have complementary roles of exploration and value propagation respectively. Once the quality of value estimations improves, more backward planning should be performed to propagate the values backward to previous states for quicker credit assignment. By doing so, a better value function can be obtained at a faster pace.

Based on our hypotheses, we propose an online method to perform bidirectional rollouts using error-based epistemic uncertainty, as a numerical indicator for the quality of value estimations. Specifically, we keep track of the maximum learning error (or exponential moving error in the function approximation case) of each state-action pair to assess learning progress, which is then used to allocate the planning budget for forward and backward plannings. To further extend our method to large and continuous state space, we apply state discretization, an efficient method to overcome the need of keeping track of all possible state-action pairs. Additionally, to counter the problem of error compounding in long rollouts in imperfect world models, we propose using an ensemble of world models to drop harmful and erroneous rollouts from learning.

By conducting experiments in both the tabular and linear function approximation settings, we reaffirm the benefits of medium-length rollouts when compared with one-step rollouts for the same amount of planning steps. We also demonstrate how our proposed method of bidirectional rollouts can improve performance and learning efficiency when compared with our baseline Dyna agents of the same planning budget, particularly in the sparse reward settings. These provide supporting evidence to the hypothesized roles of forward and backward plannings.

For future work, we plan to conduct a larger scale of study on more complicated problems to further assess our hypotheses made in this work. We also plan to develop better and more efficient methods in performing bidirectional rollouts. For instance, we can look at more principled approaches like Gaussian processes to model uncertainty, which can also be used to assess a world model's uncertainty towards its predictions.

References

Asadi, K.; Misra, D.; Kim, S.; and Littman, M. L. 2019. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*.

Asadi, K.; Misra, D.; and Littman, M. L. 2018. Lipschitz continuity in model-based reinforcement learning. *arXiv preprint arXiv:1804.07193*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

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

Ghiassian, S.; Rafiee, B.; Lo, Y. L.; and White, A. 2020. Improving performance in reinforcement learning by breaking generalization in neural networks. *arXiv preprint arXiv:2003.07417*.

Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, 3389–3396. IEEE.

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*.

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):529–533.

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

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, 4026–4034.

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

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

Talvitie, E. 2014. Model regularization for stable sample rollouts. In *UAI*, 780–789.

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*.

Volodymyr, M.; Koray, K.; David, S.; Andrei, A. R.; and Joel, V. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

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

White, M. 2017. Unifying task specification in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML-17)-Volume 70*, 3742–3750.

7 Appendix

7.1 World Models Pretraining

To pretrain the forward and backward world models, we first collect one million transitions by following a random policy. Then, we perform supervised training using fully-connected neural networks with 1 hidden layer by minimizing the mean-squared errors. The parameters details are given:

Batch Size	256
Epoch	50000
Learning Rate	0.0001
Optimizer	Adam with default parameters
Number of hidden units	512

Table 1: Hyperparameter setting for world models pretraining

For an ensemble of world models, we train each of them with different random seeds.

7.2 Feature Extractor Pretraining

We pretrain a feature extractor using a fully-connected neural network in order to obtain a better feature representation for linear function approximation, used for the experiments with the Cartpole environment. To do so, we pretrain a Deep Q learning (Mnih et al. 2015) agent and use the output of the last layer as features. The parameters details are shown in table 2.

Batch Size	32
Epoch	500000
Learning Rate	0.0001
Optimizer	Adam with default parameters
Number of hidden units	32
Experience replay buffer size	10000
Target network update rate	100

Table 2: Hyperparameter setting for feature extractor pretraining

7.3 Hyperparameter Sweep

Table 3 presents the values of hyperparameters we sweep over to produce the experimental results. We used the same exploration policy for all the agents in this work.

GridWorld	
Number of steps	100000
Learning rate	$2^{-i}, i \in 1, 2, 3, 4, 5, 6, 7, 8$
Rollout sizes	1, 2, 4, 10, 20
World sizes	5, 10, 20, 40, 80
Discount rate	0.999
Planning buffer size	1000
Cartpole	
Number of steps	20000
Feature size	32
Learning rate	$10^{-i}, i \in 1, 2, 3, 4$
Discretization bin size	6, 8, 10, 15, 20
beta	0.99
kappa	$2^{-i}, i \in 4, 5, 6, 7, 8$
Discount rate	0.999
Planning buffer size	10000
ϵ**-greedy policy**	
ϵ starting value	1.0
ϵ minimum value	0.1
ϵ decay	0.9995

Table 3: Hyperparameter sweep for the experiments