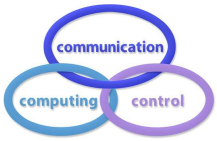
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 18, Issue: 5, Month: October, Year: 2023
Article Number: 5073, https://doi.org/10.15837/ijccc.2023.5.5073



CCC Publications

Dyna-Validator: A Model-based Reinforcement Learning Method with Validated Simulated Experiences

H. S. Zhang, J. C. Li, Z. M. He, J. H. Zhu, H. B. Shi

Hengsheng Zhang
1. School of Computer Science, Northwestern Polytechnical University
Xi'an, 710072, Shaanxi, China
2. Electronic Technology Group Corporation, Twentieth Research Institute, China
Xi'an, 710018, Shaanxi, China

Jingchen Li
School of Computer Science, Northwestern Polytechnical University
Xi'an, 710072, Shaanxi, China

Ziming He
School of Computer Science, Northwestern Polytechnical University
Xi'an, 710072, Shaanxi, China

Jinhui Zhu
School of Computer Science, Northwestern Polytechnical University
Xi'an, 710072, Shaanxi, China

Haobin Shi*
School of Computer Science, Northwestern Polytechnical University
Xi'an, 710072, Shaanxi, China
*Corresponding author: shihaobin@nwpu.edu.cn

Abstract

Dyna is a planning paradigm that naturally weaves learning and planning together through environmental models. Dyna-style reinforcement learning improves the sample efficiency using the simulation experience generated by the environment model to update the value function. However, the existing Dyna-style planning methods are usually based on tabular methods, only suitable for tasks with low-dimensional and small-scale space. In addition, the quality of the simulation experience generated by the existing methods cannot be guaranteed, which significantly limits its application in tasks such as continuous control of high-dimensional robots and autonomous driving. To this end, we propose a model-based approach that controls planning through a validator. The validator filters high-quality experiences for policy learning and decides whether to stop planning. To deal with the exploration and exploitation dilemma in reinforcement learning, a combination of ϵ-greedy strategy and simulated annealing (SA) cooling schedule control is designed as an action selection strategy. The excellent performance of the proposed method is demonstrated in a set of classical Atari games. Experimental results show that learning dynamic models in some games can

improve sample efficiency. This benefit is maximized by choosing the proper planning steps. In the optimization planning phase, our method maintains a smaller gap with the current state-of-the-art model-based reinforcement learning (MuZero). In order to achieve a good compromise between model accuracy and optimal programming step size, it is necessary to control the programming reasonably. The practical application of this method in a physical robot system helps reduce the influence of an imprecise depth prediction model on the task. Without human supervision, it is easier to collect training data and learn complex skills (such as grabbing and carrying items) while being more effective at scaling tasks that have never been seen before.

Keywords: Model-based reinforcement learning (MBRL), Dyna, Simulated annealing.

1 Introduction

Reinforcement Learning (RL) is a computational method about an agent interacting with the environment and learning an optimal strategy through trial and error, which is used for sequential decision problems in a wide range of fields such as natural science [1], social science [2] and engineering [3]. Thanks to the development of deep learning [4] [6] and neural networks [5] [7], deep reinforcement learning has flourished in areas such as robot control, autonomous driving, and gaming [8]. In RL, while a learning agent interacts with an environment over time steps, it executes an action. It affects the environment, correspondingly moving from the current state to the next and emitting a scalar reward signal. The agent's primary goal is to collect the largest accumulated reward in the long term. The agent must find an optimal policy in a deterministic or stochastic environment. In general, the RL problem is always formulated mathematically as a Markov Decision Process (MDP) [9] equipped with a transition model. A reward function is also associated with an MDP. Collectively, the transition and reward functions are often called the model of an environment. Suppose we have the transition and reward functions of MDP associated with the environment. In that case, we can exploit them and retrieve an optimal policy using model-based techniques, such as dynamic programming algorithms, Monte Carlo search, etc.

Meanwhile, in more and more complex real-world tasks, the RL methods for discrete low-dimensional state spaces can not meet the needs, deep reinforcement learning (DRL) [10] [11] can realize the direct control from the original input to the output through end-to-end learning. Model-based learning is significant to improve sample efficiency of DRL [12] [13] [14] [15], and an environment model can simply enable agents to solve complex tasks through fewer interactions with the real world than model-free methods.

Oh et al. [16] focused on using a predictive model to play Atari games well. In some cases, the predictions can be quite accurate for hundreds of steps. Ha et al. [17] presented a world model by composing a variational autoencoder with a recurrent neural network, which is successfully evaluated on a 2D racing game and VizDoom. The policy is trained by the world model can be transferred back into the actual environment. Similarly, Alaniz et al. [18] utilized a DNN-based transition model with Monte Carlo tree search to solve a block-placing task in Minecraft and the model is more training sample efficient than Deep Q-network. Dyna [19] was applied to several games from the Arcade Learning Environment (ALE) in [20], which measures the impact of planning shapes and compares the performance of Dyna-DQN on perfect and imperfect models. Model-based updates offer more benefits with unfamiliar experiences during planning. Azizzadenesheli et al. [21] proposed a sample-efficient algorithm called Generative Adversarial Tree Search (GATS) and trains a GAN-based [22] world model along with a Q-function. Simulated Policy Learning (SimPLe), a complete model-based deep RL algorithm based on video prediction models was proposed in [23] and outperforms state-of-the-art model-free algorithms on a range of Atari games. Inspired by AlphaZero [24], MuZero [25] learnd to plays games (Atari, chess, and Go) purely from the environment, with end-to-end learning and planning.

The above works are basically in video games from images. Outside of games, model-based RL (MBRL) has been widely used in other fields such as robotics [26] [27] [28] [29]. Kaiser et al. [23] and Hafner et al. [30] simulate robotic control by incorporating images into the real world.

The success of MBRL in high-dimensional problems depends on the accuracy of the dynamics model. However, despite the approach of learning dynamics models is relatively effective, the ap-

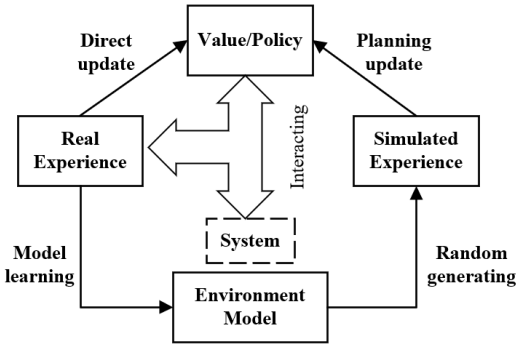


Figure 1: Illustration of Dyna architecture. In Dyna-style reinforcement learning, planning, acting, model learning, and direct reinforcement learning co-occur in parallel.

plication of model-based methods is still challenging in large-scale problems with high-dimensional observations [31]. Undoubtedly, it is extremely difficult to build a perfect dynamic model.

The Dyna architecture [19] yields policies that are both more effective than model-free learning, and more computationally efficient than the certainty-equivalence approach. The illustration of Dyna-style methods is shown in Figure 1. It simultaneously uses real experience to build a model and adjusts the policy by the model. Dyna operates in a loop of interaction with the environment. For direct RL, the agent directly interacts with the real environment and is not affected by the model error. The Dyna algorithm requires more learning cycles of indirect learning and model approximation, but it is tremendously less than the naive model-based method [32]. Incremental learning can often make full use of real limited experience to derive optimal policies and reduce interaction with the real environment. At each step, the acting, model learning, and direct RL require very little computation, and the remaining time is used for the planning process, which is inherently computation-intensive. So how to incorporate Dyna-style methods into DRL may be a research hotspot [33] [34] [35] [36].

Dyna-style imagination plays a significant role in deep model-based algorithms. Initially, the linear-quadratic-Gaussian method was used to improve model learning. Gu et al. [37] combined the back-propagation iLQG method with Dyna-style comprehensive strategy promotion. In order to accelerate modelless continuous Q learning, Locally Linear Models (Local models) are combined with locally online strategy imagination promotion. Peng et al. [38] combined DynaQ [39] with the deep learning method (DQN) and proposed deep DynaQ (DDQ) to effectively train the task to complete the conversation agent at a low cost. The framework integrates world models (similar to user simulators) into strategy learning, models real users, and simulates user behavior to complement the limited real experience in planning. Holland et al. [40] used the Dyna variant learning environment model to generate experience for strategy training in Atari games and evaluated the influence of different planning shapes on the performance of the Dyna-DQN algorithm by using the perfect model and learning model, respectively. Azizzadenesheli et al. [41] proposed a Generative Adversarial Tree Search (GATS) algorithm, which combines the success of generative adversarial tree networks with robot motion planning. Robot arm operation based on video input is an essential application of artificial intelligence. Based on the pix2pix architecture [42], a generation model was introduced to model the transfer dynamics. The model-based approach suits low-dimensional tasks with relatively simple transformation and reward mechanisms [9]. While efficient methods (such as Gaussian processes) can quickly learn these models with few samples, they are difficult to represent complex and discontinuous systems [43].

Most existing model-based reinforcement learning and Dyna-style reinforcement learning methods focus on establishing models, but few focus on the quality of experience generated by models. There is no doubt that the quality of the simulation experience is crucial when planning longer steps, especially if the state space is very complex. Take autonomous driving as an example. The single-step error generated by the model is often within the acceptable range. However, in the multi-step planning process, the accumulated errors will lead to the deviation of the later experience from the actual

track, the waste of computing resources, and the reverse help of strategy learning. Therefore, (1) it is necessary to balance the benefits of multi-step programming and the increased model errors at each step. In addition, (2) the experience generated by the model needs to be ensured to be of high quality, and the generation process needs to be controlled. These two motivations drive our work. To this end, we designed a new action selection strategy and the Dyna-Validator architecture to accomplish both improvements.

This study proposes a model-based RL, called Dyna-Validator, which achieves outstanding performance in the ALE. The score outperforms the previous quite successful model-based approach (SimPLe) in some games and is closer to the best one (MuZero). It makes Dyna-style methods may be a viable approach to MBRL in high-dimensional problems with an imperfect environment model. The model is typically used by planning methods for multiple sequential predictions, and errors in predictions accumulate quickly with each step. Inspired by the generative adversarial network (GAN) [22], a validator is proposed to differentiate high-quality simulated experiences generated by the environment model during multi-step planning. Only the predicted frame is recognized as fake (low-quality), the planning process will stop, otherwise, it will run to the specified number of planning steps. With the validator for controlled planning, the learning efficiency is greatly improved and saving the computational resources for planning.

In addition, the balance between exploration and exploitation is one of the key challenges to the performance of RL algorithms. Pure exploitation may lead to locally optimal policies. To reach the global one, exploration is necessary. Based on the above knowledge, the ε-greedy strategy is proposed to deal with this problem. However, suboptimal actions will be necessarily picked by the unmodified strategy. Excessive exploration unavoidably degrades the performance of RL algorithms even if it may accelerate the learning process and allow to neglect the locally optimal policies. Naturally, the Metropolis criterion from the simulated annealing (SA) algorithm is adopted [44]. The exploration will gradually decay, leading to convergence towards the optimum.

The main contributions of this work can be summarized as follows:

- To cope with the exploration-exploitation dilemma, the ε-greedy policy with cooling schedule control of simulated annealing (SA) is used as the action-selection strategy.
- This work proposes a model-based method, called Dyna-Validator. By employing the validator for controlled planning, the method can alleviate the effect of model errors and obtain high-quality experiences for policy learning with an imperfect environment model, especially in high-dimensional domains.

The paper is organized as follows. The second section introduces the proposed model-based method, Dyna-Validator. The third section summarizes the experimental scenarios and analysis results. The conclusions and future work are drawn in the final section.

2 Multi-step Planning with Validated Simulated Experiences

In this paper, we focus on model-based approaches to find optimal policies. This study tends to establish the Dyna-style method as an effective solution in high-dimensional domains. Dyna is a fundamental method to model-based reinforcement learning (MBRL) that interleaves planning, acting, and learning. The Dyna-style method is a potentially powerful way in large-scale, high-dimensional domains.

The challenge for model-based methods is to learn accurate models in such problems. We consider the environment model that uses a state and action as input and output one possible next state and reward. For Dyna to take full advantage of a model, it must apply the model to generate valuable and unfamiliar experiences (replace exploration in the real environment). To explore efficiently, it is necessary to generate multi-step high-quality experiences from the start state during planning. Due to the increased number of planning steps, the convergence speed and learning efficiency may be improved, but the computational resources are likely to be wasted. Undoubtedly, the more planning steps, the more simulated experience will be generated. However, the environment model cannot

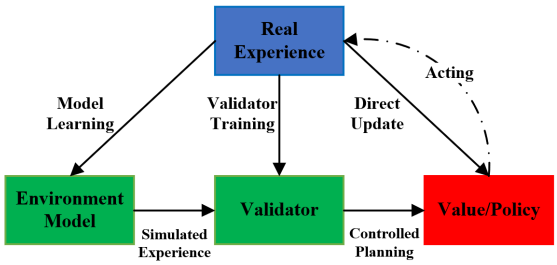


Figure 2: The proposed Dyna-style method for policy learning. The main difference between our architecture and the traditional Dyna-style architecture is the addition of a validator to mitigate the impact of model errors during longer planning steps.

perfectly reproduce the real agent's behaviors, therefore, the quality of model experience is crucial. To obtain a good compromise between model accuracy and the optimal planning step length, it is necessary to control planning reasonably.

The validator (refer Subsection 2.2) is designed to weaken the impact of model error with longer planning steps, which identifies the low-quality experience by outputting the probability of predicted frames are real and decides whether to stop the planning process, so as not to waste computational resources on each planning step and further improve the policy.

As illustrated in Figure 2, the proposed Dyna-style approach consists of three stages: (1) direct reinforcement learning: the agent interacts with the real environment, collects real experiences (transitions: frames, actions, next frames, and rewards); (2) environment model learning: the environment model is learned and refined using real experience; (3) validator modeling for controlled planning: the validator is learned and refined to differentiate the quality of simulated experience. Then the agent improves the policy using real experience and high-quality simulated experience generated by the environment model and the validator. The validator identifies the low-quality experience by outputting the probability that the prediction frame is true and decides whether to stop the planning process to avoid wasting computing resources on each planning step and further refine the strategy.

The Arcade Learning Environment is a classic benchmark in RL. The high-dimensional frames have long been problematic for model-based reinforcement learning. Actions in such games are discrete and the environments are deterministic. An agent's interaction with the environment can be formalized as a discrete-time Markov Decision Process (MDP) in Atari games. At each time step t, the agent observes the current state $\mathbf{S}_t \in \mathbb{S}$ and selects an action $\mathbf{A}_t \in \mathbb{A}$. After executing the action, the agent observes the next state $\mathbf{S}_{t+1} \in \mathbb{S}$. Besides, a reward $\mathbf{r}_t \in \mathbb{R}$ can be obtained. The agent's objective is to find an optimal policy π, which maximizes the expected return $\mathrm{Q}_\pi(\mathrm{s},\mathrm{a})$ for all s, a, where S is a finite set of states, A is a set of actions, R is a reward function. The discounted return $\mathrm{G_t} = \mathrm{r_t} + \gamma \mathrm{G_{t+1}}$, and $\mathrm{Q}_\pi(\mathrm{s},\mathrm{a}) = \mathrm{E}_\pi\left(\mathrm{G_t} \mid \mathrm{S_t} = \mathrm{s}, \mathrm{A_t} = \mathrm{a}\right) = \mathrm{E}\left[\mathrm{r_t} + \gamma \mathrm{Q}_\pi\left(\mathrm{S_{t+1}}, \mathrm{a_{t+1}}\right)\right]$.

Deep Q-network. The policy is learned by the vanilla deep Q-network (DQN) method [45]. DQN can obtain valid representations of the environment from high-dimensional sensory inputs, and learn successful policies using an end-to-end way. The optimal action-value $\mathrm{Q}'(\mathrm{s}',\mathrm{a}')$ of the sequence s' at the next time-step was known for all possible actions a'. We refer to a neural network function approximator with weights θ as a Q-network, which can be trained by Eq(1)

$$L\left(\theta\right) = E\left[\left(r + \gamma \max_{a'} Q^{'}\left(s', a'; \theta'\right) - Q\left(s, a; \theta\right)\right)^2\right] \tag{1}$$

where $\gamma \in [0,1]$ is the discount rate, which determines the present value of future rewards, $\mathrm{Q}(\cdot)$ is the approximated value function, and $Q'(\cdot)$ is the target value function, its parameters only periodically are updated by copying the Q-network's parameters.

Finding the proper balance between exploration and exploitation in RL requiring further attention. The Metropolis criterion from the SA algorithm is introduced into the action-selection strategy to deal with this problem. The cooling schedule controls the annealing rate and is critical to the performance.

In RL, the ε-greedy policy is usually employed to tackle the exploration-exploitation dilemma, with larger ε corresponding to a larger probability of exploration. During the learning process, the agent needs to extensively search the state space in the early stage, and adequately utilize the optimal policy in the later stage. As the agent's knowledge about the environment increases, the proportion of exploration should decrease. Toward this effect, the cooling schedule of simulated annealing (SA) is incorporated into the ε-greedy policy. So the probability ε for the action-selection strategy can change adaptively with episodes.

$$\varepsilon_n = \varepsilon_f + \frac{\varepsilon_0 - \varepsilon_f}{1 + e^{\beta(n-N/5)}} \tag{2}$$

where ε_n, ε_0, and ε_f denote the n-th episode, initial, and final greedy factors, respectively. β represents the scaling factor and N is the total number of episodes.

Accordingly, the action-selection policy in the n-th episode as

$$\begin{cases} \text{random}, & p < \varepsilon_n \\ \text{argmax}_a\, Q(s,a), & \text{otherwise} \end{cases} \tag{3}$$

where $p \in (0,1)$ is a random number.

Obviously, the action-selection policy does not greedily reject all the suboptimal solutions, and can eventually reach the optimal state.

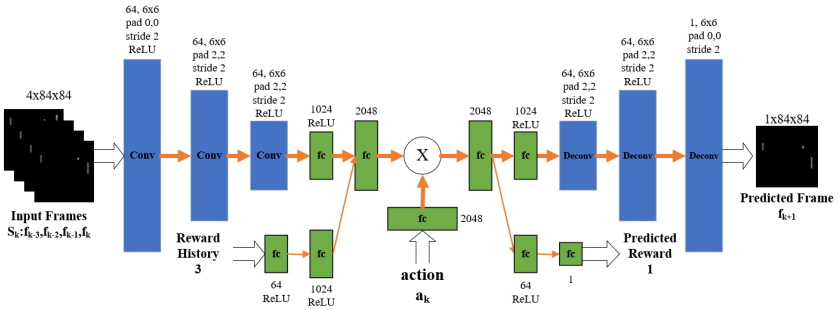
During the direct reinforcement learning process, to balance exploitation and exploration in an unknown environment, the improved ε-greedy policy, denoted by SA-greedy, is adopted to select the action. The performance of DQN with two action selection strategies is compared in Section 3.1.

2.1 Environment Model Learning

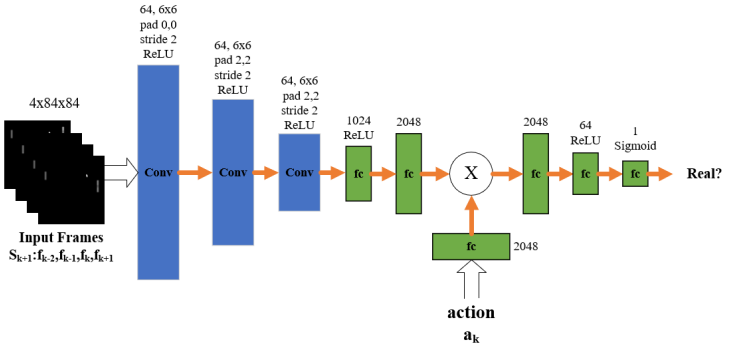
To enable planning, a suitable environmental model is needed. The environment model should at least meet the following desiderata. First, the model learning is best to be incremental and adaptive, since the agent interleaves learning and planning. Second, the model is best to be data-efficient, in order to improve the data-efficiency of learning value functions. Third, sampling is best to be computationally efficient.

However, the perfect model is almost non-existent in a large number of tasks, planning with the imperfect model is often the focus of research. So the basic model architecture is selected from [16], which can make visually accurate predictions for hundreds of steps in some Atari games. The model is extended to make reward prediction in [46]. The environment model can achieve more precise spatial transformations because temporal correlations can be obtained directly from pixels in the concatenated frames by convolutional filters, whereas it is not suitable for modeling arbitrarily long-term dependencies because it requires more memory and parameters as more frames are concatenated into the input. It is almost inevitable for the model to make noisy predictions of high-dimensional images during multi-step planning.

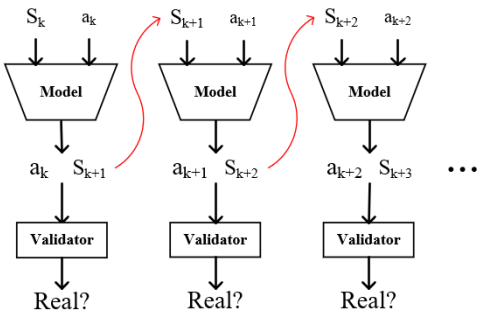
Environment Model Architecture. The basic architecture, presented in Figure 3(a), consists of a convolutional encoder and decoder. At each planning step k, the input to the model is four stacked grayscale frames S_k (as well as the action a_k) while the output is the next frame f_{k+1} and expected reward, where S_k consists of four stacked frames (f_{k-1}, f_{k-2}, f_{k-1}, f_k). The encoder maps four consecutive histories of previous frames into a feature vector using a series of convolution layers and fully-connected layers. An action is represented using the one-hot vector, which is integrated into the feature vector through element-wise vector multiplication. The advantages of multiplicative interactions have been explored in image processing [47]. After the action transformation, the single next frame and the current reward are reconstructed by a deconvolution architecture. Notably, the reward history for the three transitions associated with the input frames is provided as input. By concatenating the predicted frame with the last three history frames, and running the model forward another step, the model can make K-step predictions until the validator shows the termination signal (the predicted frame is considered low-quality) or it reaches the maximal steps (K), where K is the pre-defined planning step size. The process is shown in Figure 3(c).



(a) environment model



(b) validator



(c) K-step planning

Figure 3: The illustration of the environment model and the validator for controlled planning: (a) environment model, (b) validator, and (c) K-step planning. The network uses three different types of neuron layers ('Conv' for convolutional layer, 'Deconv' for deconvolutional layer, and 'fc' for fully-connected layer) in combination with two types of activation functions ('ReLU' and 'Sigmoid'). The current action is represented as a one-hot vector, then integrated into the compressed feature vector through element-wise vector multiplication denoted by "X".

Model Training. The model is trained to minimize the mean square error (L_2 loss) between the predicted and target frames(denoted $\widehat{\mathbf{f}}_\kappa$ and $\mathbf{f}_\kappa$ respectively) over K-steps. After the action transformation, the reward predictor outputs a single scalar reward, so the model should also be trained to minimize the K-step mean square error between the predicted and target rewards (denoted $\widehat{r}_\kappa$ and r_κ respectively). Finally, the objective function as

$$\mathcal{L}_K = \frac{1}{2K}\sum_{\kappa=1}^{K}\left(\left\|\hat{\mathbf{f}}_\kappa - \mathbf{f}_\kappa\right\|^2 + \left\|\hat{r}_\kappa - r_\kappa\right\|^2\right) \tag{4}$$

Note that the input rewards and target rewards are clipped to the same interval [-1,1] like DQN. Besides, the clipped loss max (Loss, C) for a constant C is used, which is crucial for improving the model and enables it to concentrate on small but important areas (e.g. the ball in Pong) during the optimization process. In our experiments, we set C = 15 for L_2 loss on predicted frames. Finally, the model is first trained to make 1-step, 3-step, then 5-step predictions, which is a curriculum approach to enable stability during training.

2.2 Validator Modeling for Controlled Planning

To make model-based updates more valuable, it is necessary to increase the planning step during the planning stage, but the model is usually flawed and easily makes unreliable predictions with multi-step planning. The prediction error at earlier time-steps can severely affect predictions at later time-steps, so that the model should be highly accurate short-term in order to perform reasonably longer-term. Achieving high model accuracy has been one of the major issues in high-dimensional problems. In contrast, it is easier to take some measures to stop the planning process when the prediction is wrong. The validator is applied to weaken the impact of model error as the number of planning steps increases on Dyna-style planning.

Validator. The validator, denoted by D, is used to differentiate high-quality simulated experiences and determine whether the prediction process stops during planning. D consists of a series of convolutional layers and fully-connected layers with a single output, a probability. Its architecture is illustrated in Figure 3(b). At each planning step k, the validator D predicts the probability whether a frame f_{k+1} given frames S_k and actions a is real or made by the environment model, denoted by G, where S_k consists of four stacked frames (f_{k-1}, f_{k-2}, f_{k-1}, f_k). Notably, sticky actions are used to inject stochasticity into the environment model (repeat_action_probability = 0.25) during planning [31]. In this paper, the activation function for the validator's output layer is Sigmoid, only the probability is greater than the threshold T (we set T = 0.8 in experiments), the predicted frame is considered real (high-quality). Only high-quality simulated experience can be placed into the experience buffer, then used for policy learning.

Validator Training. The validator D is trained using the predicted frames produced by the environment model, and real frames. The mini-batch training is used and the following is the validator gradient updates: for a given set of 5 consecutive frames and an action, sampled from the replay buffer, (f_1, f_2, f_3, f_4, a, f_5).

$$\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(f_5\right) + \log\left(1 - D\left(G\left(f_1, f_2, f_3, f_4, a\right)\right)\right)\right] \tag{5}$$

Practically, the predicted frames f_5 are not only simply reasonable (e.g. positions of the ball and paddles in Pong), but also fit spatial transformations based on the most recent three history frames, (f_2, f_3, f_4) and action, a. In summary, to make the Validator classify more precisely and the objective function can be rewritten as

$$\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(f_5, f_2, f_3, f_4, a\right) + \log\left(1 - D\left(G\left(f_1, f_2, f_3, f_4, a\right), f_2, f_3, f_4, a\right)\right)\right] \tag{6}$$

where m represents the batch size.

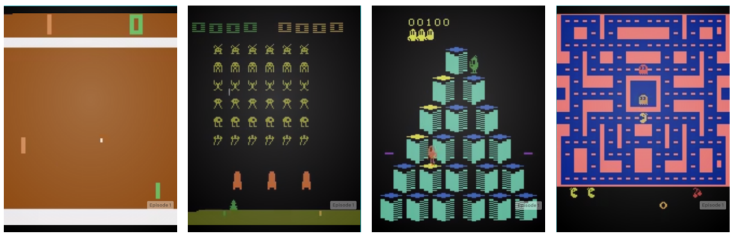


Figure 4: Screenshots from six Atari 2600 Games: (Left-to-right) Pong, Space Invaders, Ms Pacman, Qbert

In the planning process, the environment model G and the validator D are applied to generate high-quality simulated samples to improve policy. The process is repeated the maximal (K) planning steps until the predicted frame is considered fake.

In addition, two buffers are used in our method: one for storing real experience and high-quality model experience selected by the validator, which are used to improve policy, and the other contains some recent real states, which for all games was multiple episodes of experience. During planning, the second buffer has two main functions: one is used to train the environment model and the validator, the other is used to provide start states.

The above describes how the high-quality simulated experience is generated. In theory, it may take a lot of time to generate valuable samples if the environment model is not well-trained. However, this never happened in experiments because D is updated whenever G is refined.

3 Experiments

Some sample screenshots from four games from the ALE used for training are shown in Figure 4. Below we discuss the main reason for selecting each game.

Pong. Pong is chosen for our study due to its simple environment and fast convergence.

Space Invaders. Space Invaders is very difficult to model. The environment model can only achieve accurate predictions for a few time-steps into the future.

Ms Pacman. Ms Pacman is very difficult to model. The environment model can only achieve accurate predictions for a few time-steps into the future. The position of Ms Pacman regulates the movement of the ghosts according to complex rules. Furthermore, the agent is hard to fully explore certain regions of the state space.

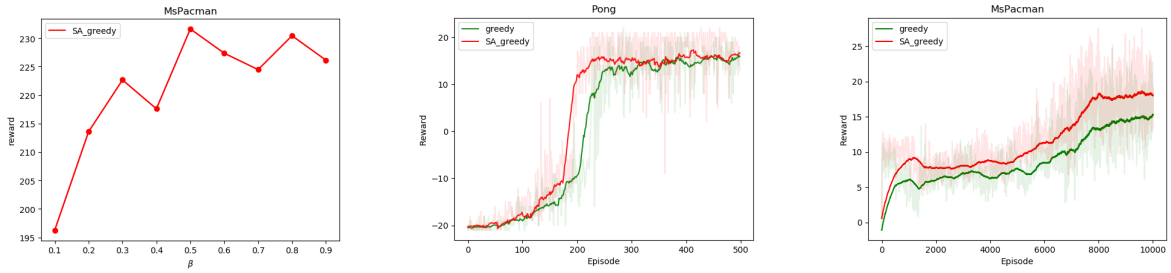
Qbert. Qbert is also a difficult game to model. The environment model is unable to predict accurately beyond very short-term, because the background is complicated.

In these environments, observation is a high-dimensional visual input (RGB image of the screen). The original observation (210x160x3) in games is more than we need, so image preprocessing is necessary. First, the image is cropped and downsampled to 84x84x3, and then the color is converted to grayscale, which means the observation vector is 84x84x1. At each planning step, the model takes a stack of the last 4 frames, which means the observation vector is 84x84x4, and action (represented as a one-hot vector) as input, then outputs a single predicted next frame and reward. The frame skips equal to 4 before a new action is selected. There are at least 3 and at most 18 actions in each game.

3.1 Action selection strategy

Our implementation of DQN used the same hyper-parameters as Mnih et al. [45] with some small changes used by Machado et al. [31]. At each step, DQN selects an action by the action-selection strategy. In view of this, a set of experiments are designed to verify the performance of SA-greedy, compared to ε-greedy ($\varepsilon = 0.3$).

The SA greedy strategy has many advantages over the ε-greedy strategy, but the nonlinear cooling plan for the exploration rate ε introduces a new hyperparameter, the scale factor β. Finding the optimal hyperparameter (hyperparameter tuning) for the learning algorithm is often one of the most



(a) Ms Pacman (with different scaling factors β)

(b) Pong

(c) Ms Pacman

Figure 5: The performance differences of SA-greedy on Ms Pacman using different scale factors β are shown in (a). The results of DQN with ε-greedy and SA-greedy in (b) Pong and (c) Ms Pacman.

challenging but necessary, so the following work will investigate the effect of scale factor β on the performance of the SA greedy strategy. Since Pong state space is relatively simple and the performance difference of different scale factors β is small, Ms Pacman is chosen as the experimental scene. In order to see the differences more clearly and accurately, only the highest original score was compared; that is, Formula (7) was not used to reshape the reward. The experiment was repeated three times for each setting, and the average was calculated. The influence of different proportion factors (β) of SA greedy strategy on the highest score of Ms Pacman is shown in Figure 5(a). When the scale factor value is 0.5, the score is highest, so it is used in other experiments. From the experimental data, different scale factors (β) did not significantly impact the experimental results.

Experimental settings. The experience replay buffer size is set to 500k. The discount factor γ is set to 0.95. The SA-initial greedy factor (ε_0) and SA-final (ε_f) are set to 1.0 and 0.01, respectively. The scaling factor (β) is 0.5. The episode number for Pong and Ms Pacman are 500 and 10000, respectively. The learning rate is 1e-4.

The rewards are normalized to adapt to different games according to [45]. The following rules are used to formulate rewards: where r_s and r are the shaped and original reward and r, respectively. The shaped reward is used during training.

$$r_s = \begin{cases} 1 & r > 0 \\ 0 & r = 0 \\ -1 & r < 0 \end{cases} \tag{7}$$

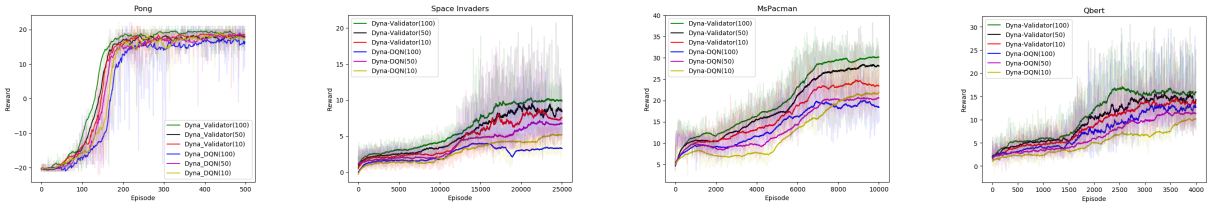
Figure 5 shows the performance of DQN with different action-selection strategies (ε-greedy and SA-greedy) in Pong and Ms Pacman.

It is observable that the SA-greedy policy has more advantages with respect to convergence speed and scores. Pong is relatively simple, and the state space is easy to explore. The SA-greedy strategy makes the agent explore the environment faster and achieve faster convergence speed than the ε-greedy. In Pong, the SA-greedy achieves convergence around the 220-th episode and the score is 16, however, the ε-greedy achieves convergence around the 250-th episode and the score is 15. With much more complex state space compared with Pong, Ms Pacman is hard to fully explore. However, we get the same result as Pong. In Ms Pacman, the SA-greedy achieves convergence around the 8000-th episode and the score is 18, however, the ε-greedy achieves convergence around the 9000-th episode and the score is 14.

Certainly, the agent explores the environment quickly in the early episode, then utilizes the optimal policy to accelerate the learning process, and obtains high scores in the later episode. In the subsequent experiments, we uniformly use the SA-greedy policy as the action selection strategy.

3.2 Environment model with various planning steps

Compared to the perfect copy of the emulator for Atari games, the environment model shown in Figure 3(a) is relatively flawed. The experiment aims to study the impact of model error and verify



(a) Pong (b) Space Invaders (c) Ms Pacman (d) Qbert

Figure 6: The results of Dyna-Validator with different planning steps (10, 50, and 100) on four games compared to Dyna-DQN: (a) Pong (b) Space Invaders (c) Ms Pacman (d) Qbert. The green, black, and red curves describe the performance of the Dyna-Validator in the three Settings, which outperformed the Dyna-DQN in almost all episodes.

the performance of the proposed method. The impact of various planning steps is investigated and the focus is on the performance of the model with multi-step planning.

Learning the model online is often very difficult. Initial model errors may cause the agent behavior that fails to visit states where the errors occur, and this situation will never be corrected. Besides, both the model and policy are constantly changing, they may uncertainly influence each other. To address these issues, we attempt to incorporate a validator for controlled planning.

For planning, start states were selected from a separate buffer containing the 10,000 most recent real states. Dyna-DQN draws m (m =100 in experiments) start states from the buffer and rolls out K steps from each, produces a sequence of K states and rewards. These samples are all placed in the experience replay buffer. DQN is incorporated into the Dyna architecture, called Dyna-DQN, which is the attempt to combine DQN with planning [20]. Dyna-DQN was trained for 100k real frames, or equivalently mxKx100k (e.g. the value is 10M when m = 100 and K=1) combined model and real frames. The training frequency was every 4 steps of real and model experience.

Experimental settings. To train the model online, batches data are sampled from real experience in the experience replay buffer. The model is trained with a 1-step prediction (batch size 32, learning rate 1e-4) for 125k updates (500k steps, with training every 4 steps), then followed with a 3-step prediction (batch size is 32, learning rate 1e-5), finally using a 5-step prediction (batch size is 8, learning rate 1e-5). Other experimental settings are the same as Subsection 3.1.

The experiments measure the performance of Dyna-Validator with different planning steps (10, 50, and 100) on four games (Pong, Space Invaders, Ms Pacman, and Qbert) from the ALE. To better evaluate the benefit of our method, we compare several methods.

DQN: the agent was trained only for 100k real frames.

Dyna-DQN: the agents were online-learned with the same environment model in Figure 3(a). The learning process is described in Subsection 2.1. The planning steps are also set to 10, 50, 100, respectively, which means that model and real frames remain the same. The detail of Dyna-DQN (the original name is Rollout-Dyna-DQN) is in [20], which is for reference only.

Dyna-Validator(our method): the learning process was described in Subsection 2.1 and Subsection 2.2, which is the same as Dyna-DQN, but our method is learned with a validator. The number of simulated experience may be less than Dyna-DQN in the replay buffer, but the quality is guaranteed.

The proposed method aims to weaken the impact of model error during multi-step planning. The results for the four games are shown in Figure 6. The average reward was recorded for ten independent runs.

Pong is relatively simple and exploration for the state space is very sufficient. The experience buffer can easily obtain all the transition information, so there is not much difference with respect to the mean scores between these methods. However, Dyna-Validator converges more quickly than Dyna-DQN on the same planning step. The more planning step for Dyna-Validator, the faster the convergence speed. Dyna-Validator achieves the best performance when the planning step is 100. Interestingly, the performance of Dyna-Validator on 10-step planning is superior to Dyna-DQN on 100-step planning. The examples of four consecutive predicted images reconstructed by our model in Pong at various stages during 100-step planning are shown in Figure 7. The model can extract

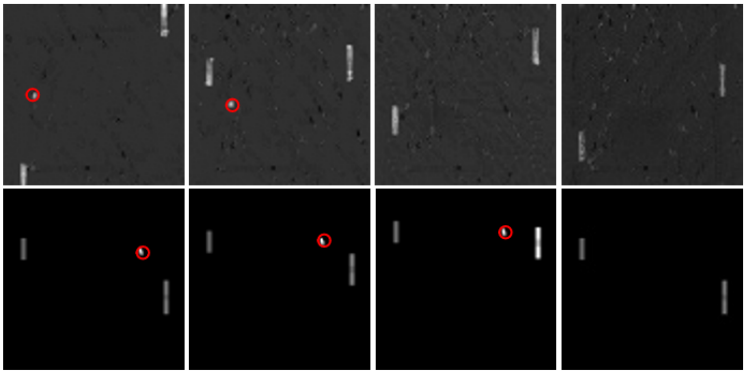


Figure 7: Pong: the above and below are the example of four consecutive predicted frames at the early stage and late stage during 100-step planning, respectively. The ball is in the red circle, otherwise, it is disappeared.

features of images, including some important information such as the ball, and simulate dynamic characteristics for the operating environment. The environment model is often hard to predict the direction of the ball when hit by the agent or the opponent. Quite rarely, the ball may disappear when hit by the agent (e.g. the fourth frame from left to right at the late stage). According to the results, these low-quality frames are discarded by the validator, and high-quality simulated experiences can greatly boost performance.

In the other three games, the performance of Dyna-Validator is similar and cleaner than Pong, which all show planning with longer steps is more beneficial with respect to the mean scores and the convergence speed. However, Dyna-DQN's performance related to the final score is quite different with various planning steps.

In Space Invaders, the optimal planning step for Dyna-DQN is 50, and the worst one is 100. The model predicts the ghost's movement very accurately but almost always fails to the bullet. At the late stage of 100-step planning, the mean score decreased slightly. Because of the model error, the bullet may predict difficultly, which affects the result. Small but useful details are easily overlooked due to the model limitations. The examples of four consecutive predicted images reconstructed by our model in SpaceInvaders at various stages during 100-step planning are shown in Figure 8. The model can simulate the entire environment, but the small detail (the bullet) is lost in the fourth predicted frame, and it is not sure whether the bullet trajectory is accurate. Even at the late stage, the bullet may disappear occasionally.

In Ms Pacman, the optimal planning step for Dyna-DQN is 10, and the worst one is 100. The environment model can predict well the movement of Ms Pacman, but fail to predict long-term the movement of the ghosts in the episodes. In Q-bert, the performance with 100-step planning for Dyna-DQN is the best, and with 10-step planning is the worst. After the training of a few frames, only the background is predicted. The agent can explore the environment well to establish the environment model in the later episode, so the quality of model experience is high and the optimal planning step is 100.

For Dyna-DQN, some results demonstrate that the trade-off between long-step planning and the model error is not negligible. For example, the best performance in Space Invader is on 50-step planning and dropped off in shorter and longer steps. The same finding also appears in other games and the best performance is often not the longer planning step. However, there is no effective method to determine the optimal planning steps, which also limits the application of Dyna in high-dimensional state spaces.

Dyna-Validator with learning the model and validator online achieves a great improvement in all games. The validator can select high-quality simulated experiences for policy learning. Judging from the results, it is indeed successful to avoid many low-quality predicted experiences during various planning steps. During longer steps, because unfamiliar but valuable experience is easier to generate. the improvement is generally the most obvious.

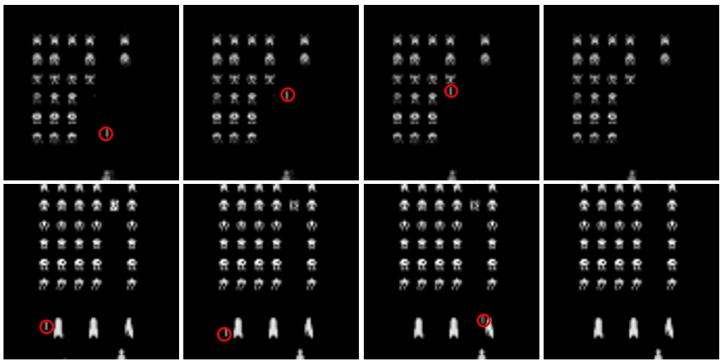
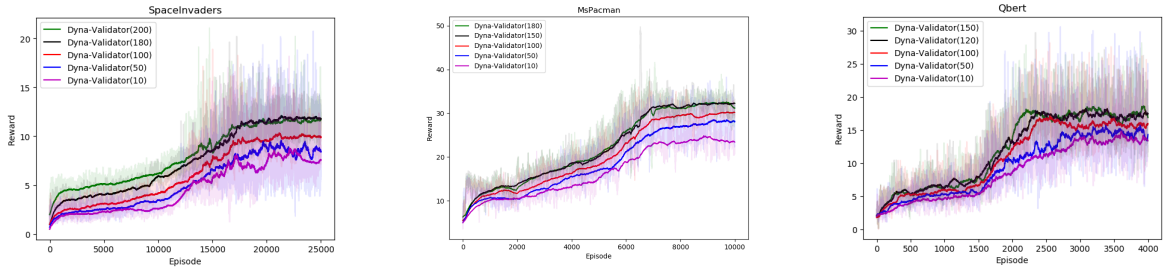


Figure 8: SpaceInvaders: the above and below are the example of four consecutive predicted frames at the early stage and late stage during 100-step training, respectively. The bullet is in the red circle, otherwise, it is disappeared.



(a) Space Invaders (b) Ms Pacman (c) Qbert

Figure 9: The results of Dyna-Validator on three games with the optimal planning step: (a) Space Invaders, (b) Ms Pacman, (c) Qbert. We compare performance differences to determine optimal planning steps.

In summary, the environment model has difficulty in accurately predicting small objects. The reason is that the squared error is small when the model fails to predict small objects during training. At the same time, this model is difficult to establish long-term dependencies. Although the models do generate new objects with reasonable shapes and movements, the predicted frames do not necessarily match the ground-truth. Dyna-Validator takes advantage of the validator and chooses high-quality experiences for policy learning during multi-step planning. To a certain extent, it has made up for some shortcomings of the model.

3.3 Environment model with the optimal planning step

The environment model with various planning steps is evaluated in previous experiments. As the above result show, the score for Dyna-Validator increases with the increase of planning steps on four games. Now we try to find the optimal planning step for Dyna-Validator. The challenge is how to balance the benefits of long steps for planning and the model's error increasing with planning step size. Fortunately, the validator has effectively weakened the impact of model error during longer planning steps, so the optimal planning step is easier to determine.

The experiment allows us to investigate whether a Dyna-style approach with an imperfect model can achieve competitive performance in comparison with some successful model-based RL algorithms (e.g. SimPLe and MuZero). We only compare game scores, which is the most common way to evaluate the performance of RL algorithms.

The experimental setup of Dyna-Validator is the same as Subsection 3.2, but adds various planning steps for different games to find the optimal one. The number of planning steps is gradually increased by 10 on the basis of 100 (e.g. 110, 120, 130...) until the best one is found. For each planning

step, we repeated the experiment 10 times and recorded the mean score. In principle, this process is finite due to the limitations of the model. The results of the experiment confirmed this hypothesis. Of course, this is only a rough estimate, we just provide one way to maximize the advantages of the proposed method.

Note that Dyna-Validator with 100-step planning had got a relatively high score (the highest score is 21) in Pong. So the results of Dyna-Validator on the other three games except Pong are shown in Figure 9. In order to make the results more convincing, there are 5 curves for each game: three kinds of curves (10-step, 50-step, 100-step) are the same as Figure 6, and the other two are the most critical for determining the optimal planning step.

In Figure 9(a), compared with 180-step planning, the score of 200-step planning behaves rather similarly in Space Invaders. The optimal planning step is about 180. In Figure 9(b), the score of 150-step planning is closer to 180-step in Ms Pacman. The optimal planning step is about 150. In Figure 9(c), compared with 120-step planning, the score of 150-step did not increase in Q-bert. So the optimal planning step is about 120. Clearly, Dyna-Validator with the optimal planning step achieved maximal scores.

To demonstrate the competitiveness of our approach, maximal scores (original reward) over five training runs is compared with SimPLe and MuZero in Table 1. In addition to the above four games, the other six games from the ALE are reported. Compared with SimPLe and Dyna-DQN, Dyna-Validator shows substantial improvements in all games except Pong. Of course, there is still a big gap with MuZero. Some policy-based RL algorithms (e.g. Asynchronous Advantage Actor-critic [48]) are integrated into the Dyna architecture may better narrow the gap.

Table 1: The maximal score of Dyna-Validator on ten games from the ALE compared to DQN, Dyna-DQN, SimPLe, and MuZero baselines.

Game	DQN	Dyna-DQN	SimPLe	**Dyna-Validator**	MuZero
Alien	165.31	325.48	616.90	**6923.42**	741,812.63
Asterix	341.53	423.46	1,128.30	**8635.81**	998,425.00
BeamRider	312.75	383.24	621.6	**5867.56**	454,993.53
Boxing	1.6	3.5	7.8	**48.4**	100.0
Hero	778.34	1534.21	2,656.60	**9621.74**	49,244.11
MsPacman	231.62	896.30	1480.0	**7563.21**	243,401.10
Pong	17	19	13	**20**	21
Qbert	123.2	455.24	1288.8	**5705.63**	72,276.00
Seaquest	134.57	277.38	683.3	**2894.47**	999,976.52
SpaceInvaders	178.25	190.26	367	**1783.82**	74,335.30

Dyna-Validator benefits from longer-step planning and the performance is less affected by the model error. Anyway, the imperfect model with learning a validator online narrows the gap with baselines. With the idea of this proposed method, researchers can focus on improving the model architecture. This experiment can prove that the validator can improve the performance of Dyna-style methods with an imperfect model in complex environments. After all, a perfect model is complicated to obtain.

In this experiment, Dyna-style methods show some advantages, which may not be the best way to solve such problems, but it provides a feasible solution. As a long-term challenge, Dyna-style methods may represent a promising and highly efficient alternative to model-free RL.

4 Conclusion

This paper proposed a model-based method, Dyna-Validator, incorporating a validator for controlled planning. Through the environment model with various planning steps, the excellent performance of the Dyna-Validator is verified in a series of games from the Arcade Learning Environment (ALE). The experimental results prove that learning dynamic models in some games can benefit sam-

ple complexity. By correctly choosing the planning step, this benefit will be maximized. During the optimal planning step, the proposed method keeps a smaller gap with the current state-of-the-art MBRL (MuZero). With an imperfect model, the optimal planning step length is difficult to determine, but the validator selects valuable model experience and enables the model's reliability in longer planning. So researchers can focus more energy on improvements in model architecture.

Collecting complete experience on state space for modeling is very time-consuming and challenging, especially in large-scale, high-dimensional domains. With the idea of this paper, the Dyna architecture can be further applied in high-dimensional state spaces. Dyna-style planning may be a promising approach to training more successful model-based agents. It is even more expensive to train an agent in the real world, so dynamics models that are trained incrementally may be useful to transfer policies back to the real world in future work. Meanwhile, such approaches can enable highly efficient policy learning from raw sensory inputs in domains such as robotics and autonomous driving. There are still limitations to the proposed approach that need to be overcome. First, the Dyna-Validator method proposed in this paper can only select the optimal planning step length by experience, which is not efficient enough. Therefore, it is of great value to study how to choose the optimal planning step size adaptively, perhaps by monitoring the model's accuracy in some way. In addition, the performance of the proposed planning method is limited by model errors. It would be interesting in the future to explore how to learn accurate, high-dimensional dynamical models from limited data.

Funding

This work is supported in part by Major Research Project of National Natural Science Foundation of China under Grant 92267110, National Natural Science Foundation of China under Grant 62076202 and 61976178, Open Research Projects of Zhejiang Lab (NO.2022NB0AB07), Shaanxi Province Key Research and Development Program of China under Grant 2023-YBGY-354, 2022GY-090, CAAI-Huawei MindSpore Open Fund (NO.CAAIXSJLJJ-2021-041A) and Innovation Foundation of Northwestern Polytechnical University (CX2022016).

Author contributions

Hengsheng Zhang: Methodology, Software, Validation, Investigation, Data curation, Writing - original draft. **Jingchen Li:** Methodology, Supervision, Writing - review & editing. **Ziming He:** Conceptualization, Writing - review & editing. **Jinhui Zhu:** Software, Validation, Resources. **Haobin Shi:** Formal analysis, Resources, Visualization, Supervision, Project administration, Funding acquisition.

Conflict of interest

The authors declare no conflict of interest.

References

[1] Kim, H.J., Madhavi, S. A Reinforcement Learning Model for Quantum Network Data Aggregation and Analysis[J]. Journal of System and Management Sciences, 2022, 12(1): 283-293. https://doi.org/10.33168/JSMS.2022.0120.

[2] Park, H.J., Kim, S.C. An Efficient Packet Transmission Protocol Using Reinforcing Learning in Wireless Sensor Networks[J]. Journal of System and Management Sciences, 2021, 11(2): 65-76. https://doi.org/10.33168/JSMS.2021.0205.

[3] Jie, W.J., Connie, T., Goh, M.K.O. Forward collision warning for autonomous driving[J]. Journal of Logistics, Informatics and Service Science, 2022, 9(3): 208-225. https://doi.org/10.33168/LISS.2022.0315.

[4] Hussein, A.K. Feature weighting based food recognition system[J]. Journal of Logistics, Informatics and Service Science, 2022, 9(3):191-207. https://doi.org/10.33168/LISS.2022.0314.

[5] Anđelić N, Car Z, Šercer M. Neural Network-Based Model for Classification of Faults During Operation of a Robotic Manipulator[J]. Tehnički vjesnik, 2021, 28(4): 1380-1387.

[6] Anđelić N, Car Z, Šercer M. Prediction of Robot Grasp Robustness using Artificial Intelligence Algorithms[J]. Tehnički vjesnik, 2022, 29(1): 101-107.

[7] Li Y, He Z, Gu X, et al. AFedAvg: communication-efficient federated learning aggregation with adaptive communication frequency and gradient sparse[J]. Journal of Experimental & Theoretical Artificial Intelligence, 2022: 1-23.

[8] Shi H, Li J, Mao J, et al. Lateral transfer learning for multiagent reinforcement learning[J]. IEEE Transactions on Cybernetics, 2021.

[9] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[10] Chen L, Deng Y, Cheong K H. Probability transformation of mass function: A weighted network method based on the ordered visibility graph[J]. Engineering Applications of Artificial Intelligence, 2021, 105: 104438.

[11] Li J, Shi H, Hwang K S. An explainable ensemble feedforward method with Gaussian convolutional filter[J]. Knowledge-Based Systems, 2021, 225: 107103.

[12] Agarwal A, Kakade S, Yang L F. Model-based reinforcement learning with a generative model is minimax optimal[C]//Conference on Learning Theory. PMLR, 2020: 67-83.

[13] Plaat A, Kosters W, Preuss M. High-Accuracy Model-Based Reinforcement Learning, a Survey[J]. arXiv preprint arXiv:2107.08241, 2021.

[14] Zhang M, Vikram S, Smith L, et al. Solar: Deep structured representations for model-based reinforcement learning[C]//International Conference on Machine Learning. PMLR, 2019: 7444-7453.

[15] Janner M, Fu J, Zhang M, et al. When to trust your model: Model-based policy optimization[J]. arXiv preprint arXiv:1906.08253, 2019.

[16] Oh J, Guo X, Lee H, et al. Action-conditional video prediction using deep networks in atari games[J]. arXiv preprint arXiv:1507.08750, 2015.

[17] Ha D, Schmidhuber J. Recurrent world models facilitate policy evolution[J]. arXiv preprint arXiv:1809.01999, 2018.

[18] Alaniz S. Deep reinforcement learning with model learning and monte carlo tree search in minecraft[J]. arXiv preprint arXiv:1803.08456, 2018.

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

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

[21] Azizzadenesheli K, Yang B, Liu W, et al. Sample-efficient deep RL with generative adversarial tree search[J]. arXiv preprint arXiv:1806.05780, 2018: 25.

[22] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial nets[J]. Advances in neural information processing systems, 2014, 27.

[23] Kaiser L, Babaeizadeh M, Milos P, et al. Model-based reinforcement learning for atari[J]. arXiv preprint arXiv:1903.00374, 2019.

[24] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge[J]. nature, 2017, 550(7676): 354-359.

[25] Schrittwieser J, Antonoglou I, Hubert T, et al. Mastering atari, go, chess and shogi by planning with a learned model[J]. Nature, 2020, 588(7839): 604-609.

[26] Deisenroth M P, Neumann G, Peters J. A survey on policy search for robotics[J]. Foundations and trends in Robotics, 2013, 2(1-2): 388-403.

[27] Veerapaneni R, Co-Reyes J D, Chang M, et al. Entity abstraction in visual model-based reinforcement learning[C]//Conference on Robot Learning. PMLR, 2020: 1439-1456.

[28] Paxton C, Barnoy Y, Katyal K, et al. Visual robot task planning[C]//2019 international conference on robotics and automation (ICRA). IEEE, 2019: 8832-8838.

[29] Deng Z, Guan H, Huang R, et al. Combining model-based q-learning with structural knowledge transfer for robot skill learning[J]. IEEE Transactions on Cognitive and Developmental Systems, 2017, 11(1): 26-35.

[30] Hafner D, Lillicrap T, Fischer I, et al. Learning latent dynamics for planning from pixels[C]//International Conference on Machine Learning. PMLR, 2019: 2555-2565.

[31] Machado M C, Bellemare M G, Talvitie E, et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents[J]. Journal of Artificial Intelligence Research, 2018, 61: 523-562.

[32] Pan Y, Zaheer M, White A, et al. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains[J]. arXiv preprint arXiv:1806.04624, 2018.

[33] Heess N, Wayne G, Silver D, et al. Learning continuous control policies by stochastic value gradients[J]. arXiv preprint arXiv:1510.09142, 2015.

[34] Feinberg V, Wan A, Stoica I, et al. Model-based value estimation for efficient model-free reinforcement learning[J]. arXiv preprint arXiv:1803.00101, 2018.

[35] Kalweit G, Boedecker J. Uncertainty-driven imagination for continuous deep reinforcement learning[C]//Conference on Robot Learning. PMLR, 2017: 195-206.

[36] Kurutach T, Clavera I, Duan Y, et al. Model-ensemble trust-region policy optimization[J]. arXiv preprint arXiv:1802.10592, 2018.

[37] Gu S, Lillicrap T, Sutskever I, et al. Continuous deep q-learning with model-based acceleration[C]//International conference on machine learning. PMLR, 2016: 2829-2838.

[38] Peng B, Li X, Gao J, et al. Deep dyna-q: Integrating planning for task-completion dialogue policy learning[J]. arXiv preprint arXiv:1801.06176, 2018.

[39] Sutton R S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming[M]//Machine learning proceedings 1990. Morgan Kaufmann, 1990: 216-224.

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

[41] Azizzadenesheli K, Yang B, Liu W, et al. Surprising negative results for generative adversarial tree search[J]. arXiv preprint arXiv:1806.05780, 2018.

[42] Isola P, Zhu J Y, Zhou T, et al. Image-to-image translation with conditional adversarial networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1125-1134.

[43] Wang T, Bao X, Clavera I, et al. Benchmarking model-based reinforcement learning[J]. arXiv preprint arXiv:1907.02057, 2019.

[44] Guo M, Liu Y, Malec J. A new Q-learning algorithm based on the metropolis criterion[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2004, 34(5): 2140-2143.

[45] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[46] Leibfried F, Kushman N, Hofmann K. A deep learning approach for joint video frame and reward prediction in atari games[J]. arXiv preprint arXiv:1611.07078, 2016.

[47] Memisevic R. Learning to relate images[J]. IEEE transactions on pattern analysis and machine intelligence, 2013, 35(8): 1829-1846.

[48] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1928-1937.

Cite this paper as:

Zhang, H.S.; Li, J.C.; He, Z.M.; Zhu, J.H.; Shi, H. B. (2023). Dyna-Validator: A Model-based Reinforcement Learning Method with Validated Simulated Experiences, *International Journal of Computers Communications & Control*, 18(5), 5073, 2023.
https://doi.org/10.15837/ijccc.2023.5.5073