Pattern Recognition 128 (2022) 108668

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Erlang planning network: An iterative model-based reinforcement learning with multi-perspective

Jiao Wang*, Lemin Zhang, Zhiqiang He, Can Zhu, Zihui Zhao

College of Information Science and Engineering, Northeastern University, China

ARTICLE INFO

Article history:
Received 31 July 2021
Revised 1 December 2021
Accepted 23 March 2022
Available online 25 March 2022

Keywords:
Model-based reinforcement learning
Multi-perspective
Bi-level
Planning
Trajectory imagination

ABSTRACT

For model-based reinforcement learning (MBRL), one of the key challenges is modeling error, which cripples the effectiveness of model planning and causes poor robustness during training. In this paper, we propose a bi-level Erlang Planning Network (EPN) architecture, which is composed of an upper-level agent and several multi-scale parallel sub-agents, trained in an iterative way. The proposed method focuses upon the expansion of representation by environment: a multi-perspective over the world model, which presents a varied way to represent an agent's knowledge about the world that alleviates the problem of falling into local optimal points and enhances robustness during the progress of model planning. Moreover, our experiments evaluate EPN on a range of continuous-control tasks in MuJoCo, the evaluation results show that the proposed framework finds exemplar solutions faster and consistently reaches the state-of-the-art performance.

1. Introduction

Reinforcement learning has been successfully applied to various applications such as games [1,2], computer vision [3], virtual machine scheduling [4], etc. Reinforcement learning (RL) algorithms are most commonly classified into two categories: model-free RL (MFRL) and model-based RL (MBRL). MFRL directly learns a value function or a policy with environment, while MBRL learns a world model to facilitate learning complex behaviors. Even though working well in a few artificial intelligence tasks [5–7], this model-free reinforcement learning (MFRL) starts to touch its performance limitation in some complex scenarios because (1) too much interactions with environment can lead to low sampling efficiency; (2) it is hard to be applied in real world, especially when agents have trouble in interacting with dynamics. On the contrary, by learning a model of the environment, model-based RL algorithms learn with significantly lower sample complexity [8], and also provide a promising direction to overcome these problems.

Normally, model-based reinforcement learning (MBRL) approaches try to learn a model that predicts future observations conditioned on actions, which can be used to simulate the real environment and plan multi-step actions. Especially, the model is represented by a Markov reward process (MRP) or a Markov decision process (MDP), and the sequential decision problem can be solved by planning, which uses this model to evaluate and select among possible policies. Given a perfect simulator is available, tree-based planning methods have enjoyed huge success in challenging domains, such as Go, Chess, and Shogi. However, in real-world problems, the dynamics governing the environment are often complex and unknown. Learning an accurate model of the real world has proven to be a challenging problem in certain fields [9], which inevitably brings about modeling errors or model-bias [8]. The modeling errors cripple the effectiveness of these methods, resulting in policies that exploit the deficiencies of the models. Moreover, model-bias will lead to the accumulating errors, which may compound during the planning and inevitably decrease the performance of planning. Clearly,the approach of model-based reinforcement learning can be improved by reducing this error, but unfortunately, this error cannot be completely eliminated, so policy learning has to deal with inaccurate environmental models. Indeed, the ultimate goal of model-based reinforcement learning is to obtain good policies. Therefore, it is worth investigating that how to obtain good policies by using models with bias.

In this paper, we take an internal multi-perspective approach to reduce the impact of model errors on policy decisions and thus improve the final performance effects of model-based reinforcement learning. The same environmental model can be abstracted as different MDPs, or different algorithms can be used to learn policies, or both, and in any case, the impact on decisions caused by biases in the model will differ. By fusing the different decisions given by these differentiated policies, the impact of errors on the final de-

* Corresponding author.
E-mail addresses: wangjiao@ise.neu.edu.cn (J. Wang), zhanglemin97@gmail.com (L. Zhang), 1900792@stu.neu.edu.cn (Z. He), 1900954@stu.neu.edu.cn (C. Zhu), zhaozihui0525@163.com (Z. Zhao).

https://doi.org/10.1016/j.patcog.2022.108668
0031-3203/

cision can be reduced. In detail, we present a bi-level *model planning* algorithm for MBRL in a parametric latent model, which bases on a long-term latent state prediction architecture recurrent state-space model (RSSM [10]). The Erlang Planning Network (EPN) architecture is composed of several multi-perspective parallel sub-agents and an upper-level agent. At every latent state, sub-agents apply the learned model to train and produce a policy distribution for the current state, which temporally extends courses of repetition action over a multi-perspective way. Moreover, its only goal is to provide independently various perspectives and rich knowledge about the world but does not change the definition of original MDP. Besides, the multi-perspective policy guidance from various sub-agents is considered by the upper-level agent, in order to execute the eventual action. Specifically, to solve long-horizon behaviors on latent imagination by fixed scale action repeat, we present EPN based on the state-of-the-art MBRL algorithm [11] which contains multiple multi-perspective sub-agents, instead of one.

A world model could be applied to multiple agents to maximize the cumulative reward for some continuous tasks. The multi-perspective actor is prone to local dilemmas and can be learned from rich action repetition scales to a better comprehension of the world model. We also focus upon the expansion of representation by environment: multiple scales views over the world model of sub-agents to assist planning. The sub-agents policy can be associated with the hidden state as a state information extension. Thus, we train the upper-level agent to plan knowledge obtained from multiple sub-agents and then perform the eventual action for the latent state in the real environment. Especially, the EPN is trained in an iterative way between two steps. First, using the current policy, data is gathered from interaction with the environment and then used to learn the latent dynamics model. Second, the policy is improved with imagined data generated by the learned model. We applied the policy repetition sub-agents individually to generally control tasks from pixels inputs, which reaches performance close to and sometimes higher than methods without repetition. The results show that EPN significantly outperforms previous model-based algorithms with much better robustness, finds exemplar solutions faster and reaches state-of-the-art performance consistently.

2. Related work

Model-based RL (MBRL) uses interactions with the environment to learn a model of it. Building an accurate observation prediction model is often very challenging when the observation space is large [12,13] (e.g., high-dimensional pixel-level image frames), and even more difficult when the environment is stochastic. Therefore, minimizing its harmful effects on planning is a promising direction for model-based RL. Normally, model-based reinforcement learning can be subdivided into two principal categories: modelling an abstract Markov Decision Process (MDP) model in an end-to-end way, and constructing a concrete state transition model and then planning in it. While the end-to-end method constructs a value function that estimates cumulative reward on different prediction tasks [1,14], which is a lack of interpretability. Unlike the second MBRL method, this learned abstract model can't represent the uncertainty of the dynamic model and doesn't offer an explicit way to represent an agent's knowledge about the world, because the internal state has no semantics of environment state attached to it, which is simple to hidden state of the overall model. Besides, it is difficult to be applied in a continuous state space because it only focuses on the value function.

The separation between model learning and planning can overcome the above problems, but faces the challenges of model-bias and compounding modeling errors during planning, as the agent is not able to optimize its model for effective planning. An ideal model requires that the realistic dynamic environment can be reconstructed as accuracy as possible, by reducing model bias, which improves the planning performance. Prior works [15–17] focus on learning local-linear models, or learning latent dynamics in a two-stage process to evolve linear controllers in imagination without probabilistic dynamics model to express model uncertainty [18]. PETS [19] builds probabilistic ensemble transition models, which improves the robustness of decisions by integrating the results of different transition models, but requires to define rewards manually for various tasks. RSSM [10] learns the environment dynamics from images using a partially stochastic sequential latent variable model that applies both deterministic and stochastic models, and accurately predicts the rewards ahead for multiple time steps. Thus, recent work has shown promising in learning the dynamics of simple low-dimensional environments [20–22]. A common method focuses on directly modelling the observation stream at the pixel level. Kaiser et al. [20] describes Simulated Policy Learning (SimPLe), a complete MBRL algorithm based on video prediction models. Effective as it is to mitigate the problems of compounding error, SimPLe is not computationally tractable in large-scale problems, which may not be available in practice. Other methods [23–25] have implemented their work in a real environment and solved some simple tasks, while we focus on the MuJoCo simulated environment. In high-dimensional environments, by learning a latent space rather than directly modelling the observation stream at the pixel level, an effective approach provides a much more compact state representation and enables fast planning.

When in high-dimensional environments, we would like to learn the dynamics in a compact latent space to enable fast planning, and on the other hand, we hope to construct a model that facilitates effective planning in visually complex domains such as the set of MuJoCo continuous-control tasks. To choose actions through fast online model planning, these recent advances have led to a great deal of excitement in the field of building a latent state-space model.The traditional planning method [15,26] has presented some advantages such as implementation simplicity, lower computational burden (no gradients) and no requirement to specify the task-horizon in advance, agents with policy gradient can learn it and have some estimation of the rewards beyond the planning horizon. I2A [27] hands imagined trajectories to a model-free policy, and SOLAR [25] solves robotic tasks via guided policy search in latent space. However, their algorithms deal with the visual complexity of the real world and solve tasks with a a simple gripper, rather than simulated environments.

Although the following work of PlaNet [10], Dreamer [11] uses actor-critic architecture as the analytic gradient to learn long-horizon behaviors for visual control purely by latent imagination, and obtains experimental results at the same level with model-free methods. For example, [28] builds a latent state-space model that is sufficient to predict its future latent states, however, it still focused on the majority of the model capacity on potentially irrelevant detail. The PlaNet [10] interprets the pixels of a state to learn a predictive model of an environment. While [11] solves long-horizon tasks from images purely by latent imagination and learns behaviors by propagating analytic gradients of learned state values back through trajectories imagined in the compact state space of a learned world model. Besides, [29] uses discrete representations and is trained separately from the policy. Even though these methods given above have arrived mean scores of the multi-round tests stabilized at a high level, the large variance of these test scores indicates that the robustness of planning in the world model with conventional CEM may be poor. Therefore, we learn a latent state with internal multi-perspectives way which avoids falling into the local optimal point and enhances the robustness during the planning.

What's more, Hierarchical reinforcement learning gives inspiration to fuse different multi-perspective agents (an upper-level agent and several multi-scale parallel sub-agents) in structural form. Sutton et al. [30] provides an in-depth explanation and discussion of Semi-Markov, using the word options to represent a set of meaningful actions corresponding to a certain scene.However, the sub-goals of the algorithm are artificially divided in advance. In order to realize the automatic division of sub-goals, the hierarchical structure is used to realize the automatic division of states [31–33]. Similarly, EPN uses a hierarchical structure to aggregate different sub-policies. The difference is that hierarchical reinforcement learning only chooses the different opinions of each sub-agents, while EPN gives new decisions with reference to the decisions of the sub-agents.

3. Preliminaries and notation

Before describing our proposed algorithm Erlang Planning Network (EPN), it is necessary to first formally define the problem of reinforcement learning based on Markov decision processes. Moreover, EPN contains some basic algorithmic modules, mainly including trajectory imagination, World Model and Actor-Critic. Their basic implementations are the cornerstones of the EPN core improvement, so it is necessary to give a primary definition and a short introduction.

3.1. Model-based reinforcement learning

To provide a mathematical framework for our algorithm, Markov decision processes (MDPs) can be defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma, h)$. $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space. And p denotes the transition function $p(s'|s,a)$: $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$. r denotes the reward function $p(r|s)$: $\mathcal{S} \to \mathcal{R}$. Moreover, ρ_0 represents the initial state distribution, and γ is the discount factor.

In a Markov decision process, a reinforcement learning agent seeks to learn a policy to optimize accumulated incentives by exploring. The dynamic process of the MDP is that beginning state $s_0 \in \mathcal{S}$ is initialized according to distribution ρ_0, the agent chooses an action $a_0 \in \mathcal{A}$, then the agent gets reward $R(s_0, a_0)$ given by the environment, and the MDP randomly transits to some successor state $s_1 \sim P(s'|s,a)$. The above process iterates until it encounters a terminal state s_T, or no end. The goal is to choose actions over time to maximize the expected cumulative reward $J(\pi) = \mathbb{E}_{s_t \sim \pi}\left[\sum_{t=1}^{h} \gamma^t r\right]$, which makes agent prefer immediate rewards and consider future reward.

3.2. Trajectory imagination

In the bi-level planning, lower-level agents and upper-level agent learn to plan with trajectory imagination. Normally, t is used to represent time step of observation or state transition in the real environment. Besides, τ denotes the time step of state transition in the imagine environment built by the world model. When a planning agent gets the current state s_t encoded by the encoder from observations and actions before, we write s_t down as s_τ and regard s_τ as the beginning of the trajectory imagination. Endless or too long-term imagination will lose the reference to make decisions in the present state because of the model-bias and environmental uncertainty. Therefore, there is a imagination horizon H to limit the τ as

$$\tau \in [t, t+H]$$

Planning agents use the current policy π to make a decision a_τ whose form is a distribution, and randomly select the determined action to be performed according to this action distribution. Then the transition function of world model predicts the next state s_τ where τ is updated as $\tau + 1$. These two processes alternate until the end of the imagination. Therefore, a sequence of trajectory imagination can be represented as

$$\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$$

In order to sufficiently explore the state space $\mathcal{S}$, the agents can imagine multiple groups simultaneously. Whenever the trajectory imagination is terminated, the planning agents will be updated through the rewards of states from trajectory imagination and then provide the actions a_t for the current state s_t.

3.3. World model

Considering that the trajectory sampling of the planning process requires accurate long-term prediction in the latent state space, this paper takes recurrent state-space model (RSSM, [10]) as the transition model. The deterministic transition model, such as the recurrent neural network, has poor multi-step prediction accuracy, making it easy for the planner to exploit inaccuracies. The stochastic transition model like the state-space model has difficulty inheriting information left over from multiple time steps. RSSM can steadily learn to predict multi-step states by dividing the states into stochastic and deterministic components. Concretely, RSSM is composed of the following modules:

$$\begin{aligned} &\textit{Deterministic state model:} && h_t = f(h_{t-1}, s_{t-1}, a_{t-1}) \\ &\textit{Stochastic state model:} && s_t \sim p(s_t \mid h_t) \\ &\textit{Observation model:} && o_t \sim p(o_t \mid h_t, s_t) \\ &\textit{Reward model:} && r_t \sim p(r_t \mid h_t, s_t) \end{aligned}$$

where $f(h_{t-1}, s_{t-1}, a_{t-1})$ is implemented as a Gated Recurrent Unit (GRU). Briefly, the model is to partition the state into a stochastic part s_t and a deterministic part h_t, which depend on the stochastic and deterministic parts of the previous time step via GRU, respectively. RSSM applies a parameterization of the approximate state posterior using an encoder. To avoid deterministic shortcuts from the input to the reconstruction, all information from the observations must be processed through the sampling step of the encoder. Using the encoder and reward model, two variational bounds are constructed on the data log-likelihood. The losses $\mathcal{L}(o_{1:T} \mid a_{1:T})$ for predicting the observations and the reward losses $\mathcal{L}(r_{1:T} \mid a_{1:T})$ can be written as Formula (1) and Formula (2), respectively.

$$\begin{aligned} \mathcal{L}(o_{1:T} \mid a_{1:T}) =& \mathbb{E}_{q(s_t \mid o_{\le t}, a_{<t})}[\log p(o_t \mid s_t)] \\ & - D_{KL}(q(s_t \mid o_{\le t}, a_{<t}) p(s_t \mid s_{t-1}, a_{t-1})) \end{aligned} \tag{1}$$

$$\begin{aligned} \mathcal{L}(r_{1:T} \mid a_{1:T}) =& \mathbb{E}_{q(s_t \mid o_{\le t}, a_{<t})}[\log p(r_t \mid s_t)] \\ & - D_{KL}(q(s_t \mid o_{\le t}, a_{<t}) p(s_t \mid s_{t-1}, a_{t-1})) \end{aligned} \tag{2}$$

3.4. Actor-Critic

MBRL methods focus on finding a value function that is consistent with a separately learned model. Therefore, EPN employs an actor-critic method [11] as a base form for implementing planning agents that can use the rewards imagined in the trajectory with a fixed horizon H to learn, while considering the expected rewards of states beyond the horizon H. The action model and value model of the actor-critic can be expressed as Formula (3) and Formula (4).

$$\textit{Action model:} \quad a_\tau \sim q_\phi(a_\tau \mid s_\tau) \tag{3}$$

$$\textit{Value model:} \quad v_\psi(s_\tau) \approx \mathrm{E}_{q(\cdot|s_\tau)}\left(\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau\right) \tag{4}$$

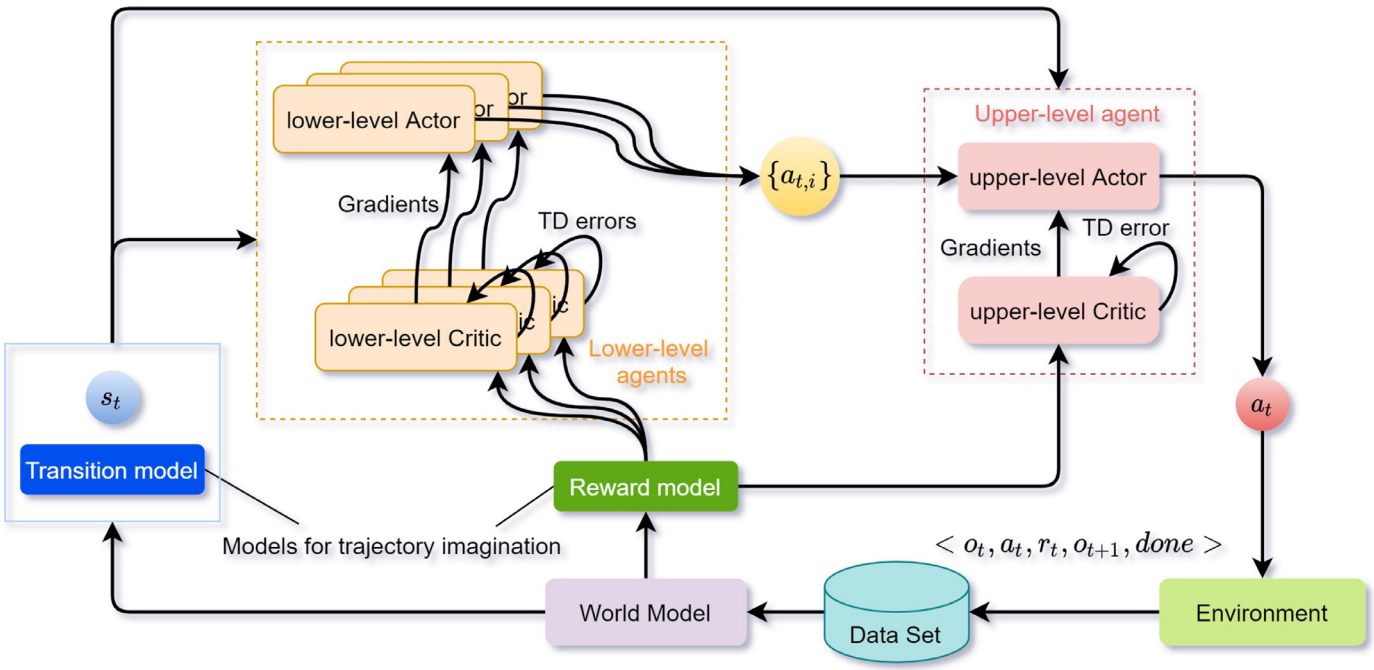


Fig. 1. Diagram of the Erlang Planning Network architecture.

ϕ denotes the parameters of the action model. ψ denotes the parameters of the value model. t indicates the current moment, while τ represents each moment point of the given state in the trajectory imagination. And γ is a discount factor. The action model gives a set of policies in the trajectory imagination based on the states given by the world model. The value model estimates the expected rewards of the action model from a state. To achieve the learning goal, the update target of the value function is estimated by the following Formula (5) [11], which can balance high variance and high bias in the value estimation.

$$\mathrm{V}_\lambda(s_\tau) \doteq \lambda^{H-1}\mathrm{V}_\mathrm{N}^H(s_\tau) + (1-\lambda)\sum_{n=1}^{H-1}\lambda^{n-1}\mathrm{V}_\mathrm{N}^n(s_\tau) \tag{5}$$

where h denotes $\min(\tau+k, t+H)$ and $\mathrm{V}_\mathrm{N}^k(s_\tau)$ is defined as Formula (6).

$$\mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{h-1}\gamma^{n-\tau}r_n + \gamma^{h-\tau}v_\psi(s_h)\right). \tag{6}$$

4. Methodology

Erlang Planning Network (EPN) follows the world model described in the previous section and makes great improvements by using multiple perspectives on the policy learning. So this section will first show the architecture of EPN, and then focus on the acquisition and synthesis of multiple perspectives in details.

4.1. Architecture

The architecture of the Erlang Planning Network (EPN) is shown in Fig. 1, which consists of three main components: a world model, a set of lower-level planning agents, and an upper-level planning agent. The learning process of EPN is iterative with two alternating processes: model updating and sampling. Among them, the model updating can be further divided into three stages based on the three main components of EPN. The world model uses samples from data set $\mathcal{D}$ to learn a representation from a observation of the environment to a hidden state, which can be simply expressed as $\mathcal{O} \to \mathcal{S}$. The state transition model is then updated iteratively by a recurrent process that receives the previous hidden state and hypothetical next actions $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$ in a low-dimensional latent space, and to create a mapping $\mathcal{S} \to \mathcal{R}$ of each latent state to its corresponding reward. Then the lower-level agents use the encoded latent state and transition model to imagine trajectories and estimate the rewards of these trajectories with the reward model to update the lower-level agents themselves. Update of upper-level agent looks similar to that of each lower-level agent. The difference is that the policy update of the upper-level agent need to consider the advice of lower-level agents. After the three components have been updated in turn, the algorithm samples data in the real environment and collects the samples into the data set $\mathcal{D}$ for the world model to learn in the next round. With the architecture designed as above, the major goal of EPN is to use the multi-perspective knowledge provided by the sub-agents to decrease the impact of errors on planning.

4.2. Policy learning

In this section, we present the policy learning of bi-level planning in two parts: lower and upper levels. And a novel policy learning framework is proposed for each of the two parts.

4.2.1. Lower-level agents

In the world model of EPN, the state transition has a fixed timescale t_s. t_i denotes the execution time of a lower-level planning agent's decision, which is i times t_s. And I is specified as the set of all i. As shown in Fig. 2, the t_i of the agent in Fig. 2(a) is equal to t_s, while the one in Fig. 2(b) is twice as large as t_s. Whenever an action $a_\tau \sim q_\phi(a_\tau \mid s_\tau)$ is derived from the current state s_τ, the agent's actor with t_i will ignore the states given by the transition model $i-1$ times and enforce the action i times. Therefore, the action model of a sub-agent can be expressed as Formula (7), and the value model follows the one described above 4.

$$\begin{cases} a_\tau \sim q_{\phi_i}(a_\tau \mid s_\tau) & i \mid ((\tau - t)/t_s) \\ a_\tau = a_{\tau-1} & otherwise \end{cases} \tag{7}$$

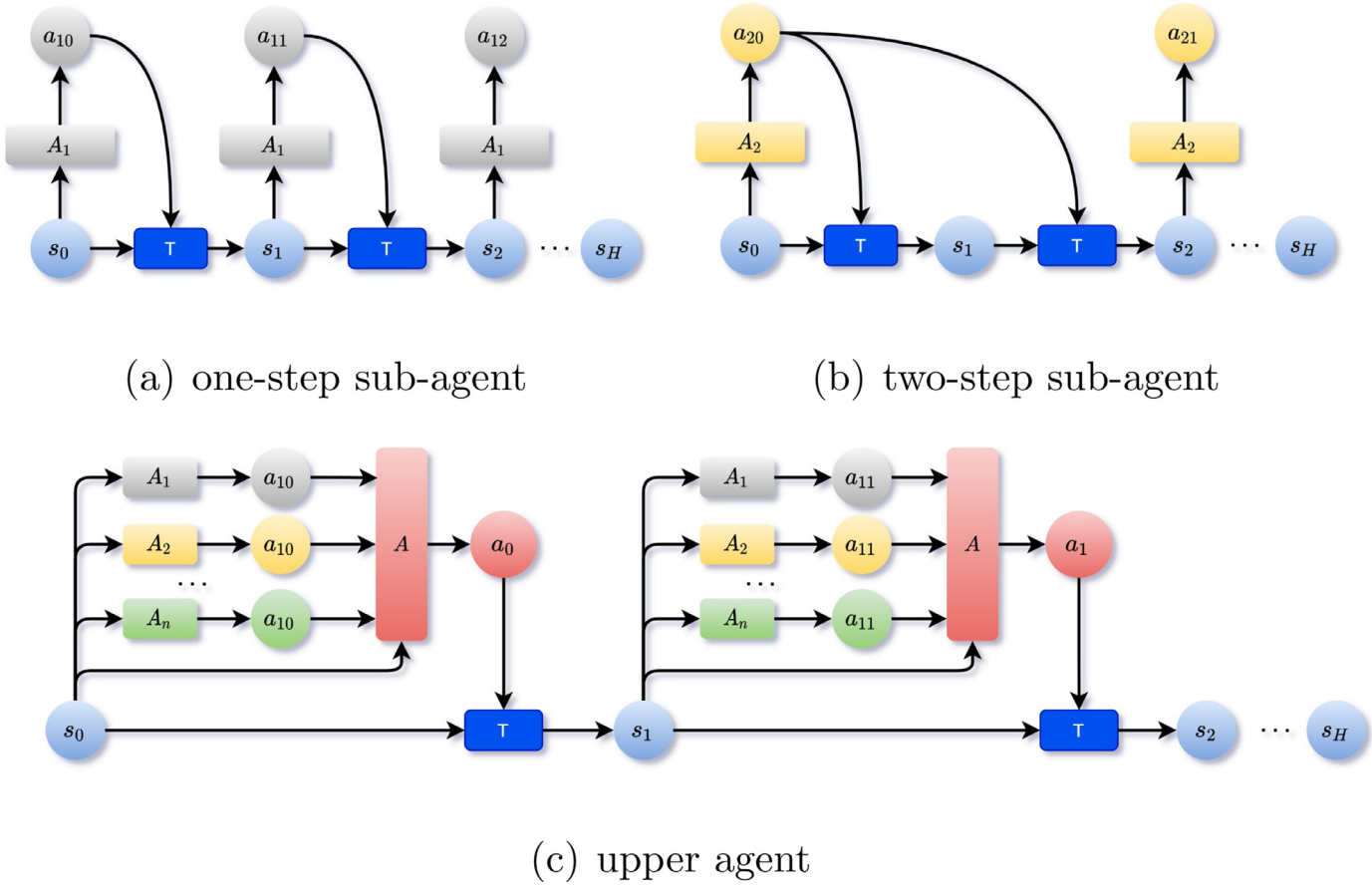


Fig. 2. Diagram of trajectory imagination in latent state space. Both (*a*) and (*b*) present the trajectory imaginations of the sub-agents. The action repeat of the sub-agent in (*a*) is one, while in (*b*) it is two. Figure (*c*) shows that upper-level planning agent to image a trajectory by referring to decisions of sub-agents. Obviously, the upper-level agent only considers the sub-agents' actions, does not refer to the time scales of these actions.

Formula (7) can be abbreviated as $a_\tau \sim q_{\phi_i}(a_\tau \mid s_\tau, a_{\tau-1})$. Therefore, the action model and value model of a sub-agent with parameter i can be written as Formula (8) and (9), respectively.

$$\textit{Action model:} \quad a_\tau \sim q_{\phi_i}(a_\tau \mid s_\tau, a_{\tau-1}) \tag{8}$$

$$\textit{Value model:} \quad v_{\psi_i}(s_\tau) \approx \mathrm{E}_{q_{\phi_i}(\cdot|s_\tau)}\left(\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau\right) \tag{9}$$

Since the neighboring states in the transition are similar, their distributions of due actions and values after the policy convergence are similar too. The above described way of enforcing repetition as a new way of imposing noise makes the lower level agents to be multi-perspective. The optimizations of low-level agents are similar and can be expressed as Formula (10) and Formula (11). In order to ensure that the policy update of each lower-level planning agent is stable, the world model keeps stationary while the agents are learning behaviors.

$$J\left(\pi_{\phi_i}\right) = \mathrm{E}_{q_\theta, q_{\phi_i}}\left[\sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)\right] \tag{10}$$

$$J\left(\pi_{\psi_i}\right) = \mathrm{E}_{q_\theta, q_{\psi_i}}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\left\| v_{\psi_i}(s_\tau) - \mathrm{V}_\lambda(s_\tau)\right\|^2\right] \tag{11}$$

4.2.2. Upper-level agent

As shown in Fig. 2(c), the decision of upper-level planning agent is based on both the current state s and the decisions a_i of all lower-level agents in the state s.

Therefore, the optional range of its timescale is limited by the timescale of world model and the lower-level agents. The time scale of the upper-level planning agent can be defined as

$$t_* \in \left[t_s, \min_i(t_i)\right] \cap \left\{w \cdot t_s \mid w \in \mathbb{N}^+\right\}. \tag{12}$$

Considering this, a novel actor critic approach is created where the input of the actor is defined as the combination of the current state s_τ and the decisions $\{a_{\tau,i} \mid i \in I\}$ suggested by all lower-level agents in the current state s_τ. The upper-level agent's action model and value model can be expressed as follows respectively.

$$\textit{Action model:} \quad a_\tau \sim q_\omega(a_\tau \mid s_\tau, \{a_{\tau,i} \mid i \in I\}) \tag{13}$$

$$\textit{Value model:} \quad v_\kappa(s_\tau) \approx \mathrm{E}_{q_\omega(\cdot|s_\tau)}\left(\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau\right) \tag{14}$$

The input of value function is just the current state s_τ. Because it is difficult for the value function to effectively extract information from the decisions $\left\{a_{\tau,i} \mid i \in I\right\}$, and the environment with these decisions is unstable for the value function, which can lead to oscillations and even divergences in the update of the value function. Similar to the principle of policies update for lower-level planning agents, both the world model and the sub-agents should keep stationary to ensure that upper-level planning agent can be updated steadily. The optimization of upper-level agents is presented in Formula (15) and Formula (16).

$$J(\pi_\omega) = \mathrm{E}_{q_\theta, q_{\phi_i}, q_\omega}\left[\sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau, \{a_{\tau,i} \mid i \in I\})\right] \tag{15}$$

$$J(\pi_\kappa) = \mathrm{E}_{q_\theta, q_{\phi_i}, q_\kappa}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\|v_\kappa(s_\tau) - \mathrm{V}_\lambda(s_\tau)\|^2\right] \tag{16}$$

Let Φ_i and Ψ_i respectively denote the parameter spaces of the lower-level agent actor and critic with t_i. In addition, Ω and K are used as the parameter spaces of the upper-level planning agent's actor and critic. The optimizations of the upper-level planning policy and the lower-level planning policies form a bi-level optimization problem, which can be formally defined as follows.

$$\max_{\omega} \mathrm{E}_{q_\theta, q_{\phi_i}, q_\omega}\left[\sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau, \{a_{\tau,i} \mid i \in I\})\right]$$

s.t. $\omega \in \Omega$

$$\min_{\kappa} \mathrm{E}_{q_\theta, q_{\phi_i}, q_\kappa}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\|v_\kappa(s_\tau) - \mathrm{V}_\lambda(s_\tau)\|^2\right]$$

s.t. $\kappa \in K$

$$\phi_i' = \arg\max_{\phi_i} \mathrm{E}_{q_\theta, q_{\phi_i}}\left[\sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)\right]$$

s.t. $\phi_i \in \Phi_i$

$$\psi_i' = \arg\min_{\psi_i} \mathrm{E}_{q_\theta, q_{\psi_i}}\left[\sum_{\tau=t}^{t+H} \frac{1}{2}\|v_{\psi_i}(s_\tau) - \mathrm{V}_\lambda(s_\tau)\|^2\right]$$

s.t. $\psi_i \in \Psi_i$

4.3. Convergence proof for upper-level planning

Theorem 1 (Contraction mapping theorem)**.** *Let X be a complete metric space, and $F: X \to X$ be a contraction. Then F has a unique fixed point, and under the action of iterates of $F: X \to X$, all points converge with exponential speed to it.*

Proof. Let $\mathcal{V}$ denote the value vector space of a planning agent, then a value vector $\mathbf{v} \in \mathcal{V}$ can be defined as Formula (17).

$$\mathbf{v} = \begin{bmatrix} v(s_1) \\ v(s_2) \\ \vdots \\ v(s_n) \end{bmatrix} \tag{17}$$

We define the metric of the value vector space as infinite norm in Formula (18).

$$d(\mathbf{v_1}, \mathbf{v_2}) = \max_{s \in \mathcal{S}} |v_1(s) - v_2(s)|, \forall \mathbf{v_1}, \mathbf{v_2} \in \mathcal{V} \tag{18}$$

Obviously, $< \mathcal{V}, d >$ is a complete metric space, then the Bellman Equation of a value vector iteration can be written as Formula (19).

$$\mathbf{v}' = F^\pi(\mathbf{v}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v} \tag{19}$$

In Formula (19), $\mathbf{v}$ and $\mathbf{v}'$ denote the value vectors before and after one update, respectively. And $\mathcal{P}^\pi$ is the state transition matrix, and $\mathcal{R}^\pi$ is a fixed reward vector.

$$\begin{aligned} d(T^\pi(\mathbf{v_1}), T^\pi(\mathbf{v_2})) &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v_1}) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v_2})\|_\infty \\ &= \|\gamma \mathcal{P}^\pi (\mathbf{v_1} - \mathbf{v_2})\|_\infty \\ &\le \|\gamma \mathcal{P}^\pi \|\mathbf{v_1} - \mathbf{v_2}\|_\infty\|_\infty \\ &\le \gamma \|\mathbf{v_1} - \mathbf{v_2}\|_\infty = \gamma d(\mathbf{v_1}, \mathbf{v_2}) \end{aligned} \tag{20}$$

According to Formula (20), the $F_\pi(\mathbf{v})$ is a contracting mapping. Based on contracting mapping Theorem 1, the value vector iteration converges uniquely at a linear rate γ to $\mathbf{v}^*$.

In EPN, the lower-level agents π_i can converge, when the world model $< \mathcal{P}_\theta, \mathcal{R}_\theta >$ has converged. And it can be expressed as Formula (21).

$$\mathbf{v}^*_{\psi_i} = \mathcal{R}_\theta{}^{\pi_{\phi_i}} + \gamma_i \mathcal{P}_\theta{}^{\pi_{\phi_i}} \mathbf{v}^*_{\psi_i} \tag{21}$$

According to (13), the upper-level agent can converge, when the world model $< \mathcal{P}_\theta, \mathcal{R}_\theta >$ and all lower-level agents have all converged. And the convergence can be expressed as Formula (22).

$$\mathbf{v}^*_\kappa = \mathcal{R}_\theta{}^{\pi_\omega} + \gamma \mathcal{P}_\theta{}^{\pi_\omega} \mathbf{v}^*_\kappa \tag{22}$$

□

4.4. Algorithm summary

The algorithm of our MBRL approach EPN can be summarized as Algorithm 1. It uses a latent model to imagine trajectories, lower-level agents to extract auxiliary information, and upper-level agents to interact with the real environment. To avoid harm in the process of using auxiliary information, a new actor-critic approach is constructed.

Algorithm 1 Erlang Planning Network.

World Model:		**Hyper-parameters**:	
Representation	$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$	Seed episodes	S
Transition	$q_\theta(s_t \mid s_{t-1}, a_{t-1})$	Time scales	I
Reward	$q_\theta(r_t \mid s_t)$	Collect interval	C
Action Pool	$\{q_{\phi_i}(a_{t,i} \mid s_t) \mid i \in I\}$	Batch size	B
Value Pool	$\{v_{\psi_i}(s_t) \mid i \in I\}$	Sequence length	L
Action	$q_\omega(a_t \mid s_t, \{a_{t,i} \mid i \in I\})$	Imagination horizon	H
Value	$v_\kappa(s_t)$	Training repeat	G

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize parameters $\theta, \{\phi_i\}, \{\psi_i\}, \omega, \kappa$ of neural networks randomly.
while *not converged* **do**
  for *update step* $c = 1.C$ **do**
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$
    Update θ using representation learning.
    for *update step* $g = 1.G$ **do**
      for *time scale* i *in* I **do**
        Imagine trajectories $\{(s_\tau, a_{\tau,i})\}_{\tau=t}^{t+H}$
        Predict rewards $\mathrm{E}(q_\theta(r_\tau \mid s_\tau))$ and values $v_{\psi_i}(s_\tau)$
        Update ϕ_i and ψ_i
      end for
    end for
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ with $q_\omega(a_\tau \mid s_\tau, \{a_{\tau,i} \mid i \in I\})$
    Predict reward $\mathrm{E}(q_\theta(r_\tau \mid s_\tau))$ and value $v_\kappa(s_\tau)$
    Update ω, κ
  end for
  $o_1 \leftarrow env.reset()$
  for *time step* $t = 1.T$ **do**
    Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ and $a_t \sim q_\omega(a_t \mid s_t, \{a_{t,i}\})$
    Add experience noise to action.
    $r_t, o_{t+1} \leftarrow env.step(a_t)$
    Extend dataset $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(o_t, a_t, r_t)_{t=1}^{T}\}$
  end for
end while

5. Experiments

In this section, we evaluate the performance of our proposed bi-level planning algorithm called EPN using parallel multiscale

agent. First, we provide a detailed evaluation of the individual multi-perspective sub-agent and analyze their effect on performance. Then, we explore how each sub-agent influences the overall performance and explain our results in the DeepMind Control Suite [34]. Finally, we compare our approach on standard benchmark tasks against state-of-the-art MBRL approaches. The training time is from 20 to 60 h (depending on the task and the number of sub-agents) on a single Nvidia A100-PCIE-40GB GPU.

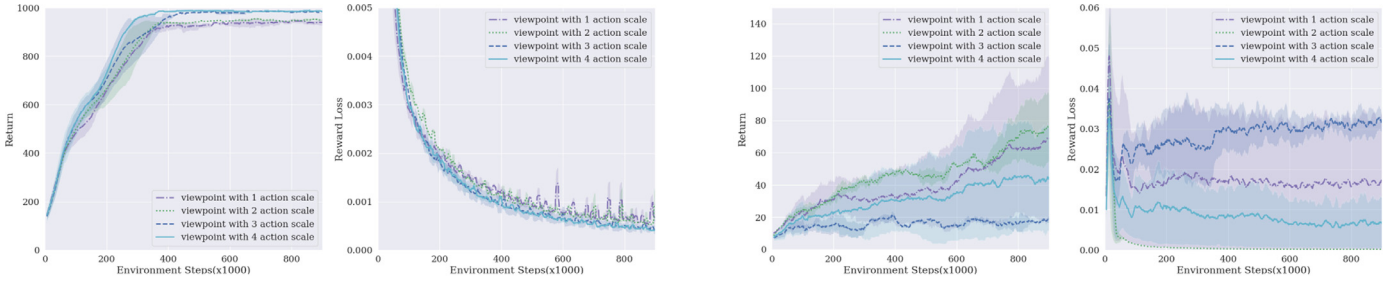
5.1. Experiment settings

Sub-agents are designed to provide policy from multiple perspectives. The different policies are learned in the imagination from the rewards obtained with a variety of action repetitions. The multi-perspective of sub-agents and knowledge abstraction of upper-actor are the key points proposed in this paper. Therefore, only these two core improvements of EPN are analyzed and compared. First, we experiment with individual agents of multiple perspectives to evaluate the impact of various views. Second, we design the experiment to compare the final algorithm EPN with the dreamer, which is the state-of-the-art method in the literature. Also, we set up the PlaNet as a baseline. To verify the robustness of the algorithm, we randomly select five environments for testing. We will publish the source code after the paper is accepted.

We compare the performance of our method on a subset of tasks with the continuous action space. These control tasks consist of typical challenges in MBRL, such as a long horizon, sparse reward, etc. In all tasks, the only observations are third-person camera images of size $64 \times 64 \times 3$ pixels. We use the same hyperparameters across all tasks. The parameters of RSSM are the same as dreamer unless specified. To speed up the training, the initialization dataset with $S = 5$ episodes uses random actions to carry out 500 steps in parallel and the RSSM is also trained in parallel. The replay buffer size is 1000000, and the batch size for sampling is 51.

5.2. Multi-perspective analysis

For comparing multi-perspective policy on the continuous control tasks, we train the sub-agent to enlarge the cumulative rewards on latent imagination samples, which is generated by learned word model. The hidden layer size is 200 and the imagi-



(a) Cartpole-Balance

(b) Acrobots-Swingup

(c) Hopper-Hop

(d) HalfCheetah-v2

(e) Ant-v2

Fig. 3. Results of the experiments on different action time scales. The viewpoint with 1 action scale is the original dreamer algorithm. The rest of different action scales represents that different actions are performed on the time scale.

Table 1
Comparison of action selection schemes on the continuous control tasks of the DeepMind Control Suite. The metrics used for comparison include episode average score and standard deviation. The EPN algorithm involved in the comparison has two perspectives, as two perspectives can exhibit optimal performance of EPN. The baselines are the evolutionary algorithm CEM, the model-free algorithm D4PG, and the well-known A3C agent.

Env	Acrobot-swingup	Hopper-hop	Cartpole-balance	Halfcheetah-v2	Ant-v2
A3C	41.9	0.5	951.6	-	-
D4PG	91.7	242.0	**992.8**	-	-
PlaNet	10.4 ± 3.1	1.9 ± 0.1	404.8 ± 81.8	948.0 ± 50.8	865.2 ± 5.6
Dreamer	72.6 ± 50.3	172.8 ± 73.5	947.1 ± 10.4	1983.6 ± 1065	937.7 ± 13.3
EPN2	**150.5** ± 5.7	**316.9** ± 18.9	988.4 ± 3.6	**2674.3** ± 57	**1018.8** ± 51
CEM	43.62 ± 4.42	0.40 ± 0.37	875.54 ± 0.86	1557.33 ± 194	0 ± 0

nation horizon is given as $H = 15$ for each unique perspective sub-actor. The learning rates 8×10^{-5} is preset for both actor and value model models. In contrast to A3C [35,36], the action repetition scale provides a more varied policy instead of different parameters trick. Dreamer proposed that the final performance is sensitive to the control frequency hyper parameter and the parameter is set to 2 works best across tasks. Therefore, in order to verify the robustness and advantage of our algorithm, the control frequency 2 is adopted in learning the dynamic model.

Fig. 3 shows the result of the multi-perspective actor experiment, which contains the episode reward during the training of the different action timescales executed in the selected test environment. The shadows around the lines show one standard deviation across 2 seeds. The four action perspectives have resulted in better performance, and notably, multi-perspective actors can reduce the multi-time calculation of the action. The above experimental results of Fig. 3 demonstrate that sub-agents have various perspectives on the world model at different temporal action scales. Multi-perspective analysis, which belongs to sub-agents, can provide more information to facilitate escaping from local optimum, but without excessive performance loss or even higher return.

The reward function is the final evaluation metric, and the learning effectiveness of the reward function will have a major impact on the general performance. In the selected environments, as shown in cartpole-balance and acrobots-swingup, the smaller reward loss value can obtain a better final performance. In the Ant-v2, Hopper-hop and HalfCheetah-v2 environments, the multi-view scales were more clearly irregular in the correlation between the loss of reward function values and the return values. Therefore, it becomes especially critical to consider the policies at different scales in an integrated manner.

5.3. Integrating multi-perspective analysis

In the second experiment, we will corroborate the upper-layer agent can effectively absorb the information of the multi-perspective views by the sub-agents. The multi-perspective action repetition of four sub-agents is set to $I \in \{1, 2, 3, 4\}$. The hidden layer size upper-level actor network is all (200, 200, 200). The hid-

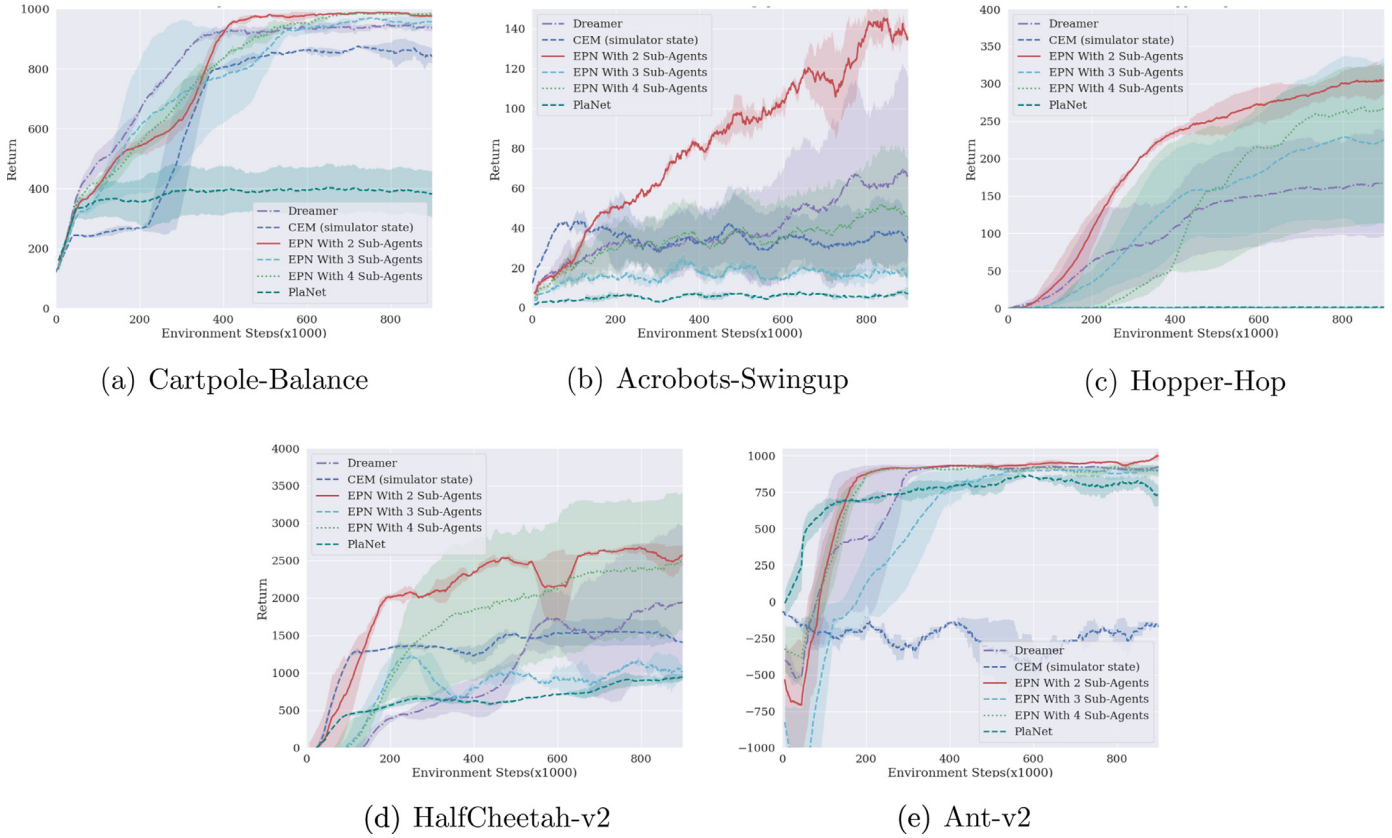


Fig. 4. Comparison of the curves of the algorithms' training process. The curves in the figure show the scores of each algorithm with increasing number of environmental steps. The algorithms involved in the comparison include CEM, Dreamer, PlaNet, and EPN with different numbers of sub-agents.

den layer architectures of all the actors are the same as the upper-level actor. The sub-agents are trained individually in a multi-process asynchronous method with various perspectives. The sub-actor training repetition parameter G is set to 5. The sub-agents are only used for inference and not updating when the upper-agents are trained.

Fig. 4 presents the return of training process. Dreamer, PlaNet, CEM, and EPN achieve episode returns after the same number of environment steps. As we can see from the results, EPN achieves a high score on the randomly selected five tasks. For simpler tasks like Cartpole-Balance the performance difference between different scales is not significant and the fusion between different policies has a relatively small impact on the final result. For more difficult tasks, fusing too much redundant information also makes the final performance degradation, but the performance is mostly better than the dreamer algorithm that is not fused as shown in Table 1. The results show that it is more effective to fuse two scales agent. To show the performance of EPN more intuitively, the specific scores of EPN and other algorithms are shown in Table 1. Among the algorithms involved in the comparison, model-free reinforcement learning algorithms are added.

From Fig. 4, we can draw the conclusion that with multi-perspective sub-agent set, our EPN improves its performance finally in the training maps. This shows the upper actor can extract more knowledge from the multiple perspectives given by the sub-agent and the two-level EPN network structure is effectively able to obtain greater returns. Two interesting things we'd like to mention: (1) At the beginning the episode return of EPN is fewer, because of the sub-actor contribute to some unreasonable behaviors. (2) The training repetition hyperparameter G has little effect on the final performance.

6. Conclusions and future work

To improve the performance of model-based reinforcement learning, we analyze the bottlenecks in the development of model-based reinforcement learning and creatively propose the idea of multi-perspective fusion to reduce the impact of model errors on policies. To concretize this idea, we propose EPN, a new MBRL method, in which the concept of multiple perspectives is specified in the time-scale dimension, and construct a feasible two-tier planning learning framework for decision fusion based on Actor-Critic. According to the experimental results, EPN has its obvious advantage over various current advanced reinforcement learning algorithms, and this advantage is especially obvious in complex tasks with large continuous action spaces. It is corroborated that multiple-perspectives in model-based reinforcement learning has capability to attenuate the effects of environmental model errors.

More auxiliary information can be used in this framework for decision making, such as self-supervised multimodal subagent information. Because of multiple view analysis of sub-agents, we will take two or three times more computational overhead. Moreover, in the policy extraction part, we can use a non-learning algorithm to reduce the computational overhead. We present these investigations as future work. Further, even more architectures can be designed to validate the idea of multiple perspectives.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

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

[2] F. Huang, W. Li, J. Cui, Y. Fu, X. Li, Unified curiosity-driven learning with smoothed intrinsic reward estimation, Pattern Recognit. 123 (2022) 108352.

[3] J. Yang, Y. Zhang, R. Feng, T. Zhang, W. Fan, Deep reinforcement hashing with redundancy elimination for effective image retrieval, Pattern Recognit. 100 (2020) 107116.

[4] J. Sheng, Y. Hu, W. Zhou, L. Zhu, B. Jin, J. Wang, X. Wang, Learning to schedule multi-NUMA virtual machines via reinforcement learning, Pattern Recognit. 121 (2022) 108254.

[5] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S.M. Eslami, et al., Emergence of locomotion behaviours in rich environments, arXiv preprint arXiv:1707.02286 (2017).

[6] O.M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al., Learning dexterous in-hand manipulation, Int. J. Rob. Res. 39 (1) (2020) 3–20.

[7] X.B. Peng, P. Abbeel, S. Levine, M. van de Panne, DeepMimic: example-guided deep reinforcement learning of physics-based character skills, ACM Trans. Graph. (TOG) 37 (4) (2018) 1–14.

[8] A.S. Polydoros, L. Nalpantidis, Survey of model-based reinforcement learning: applications on robotics, J. Intell. Rob. Syst. 86 (2) (2017) 153–173.

[9] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, J. Ba, Benchmarking model-based reinforcement learning, arXiv preprint arXiv:1907.02057 (2019).

[10] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, J. Davidson, Learning latent dynamics for planning from pixels, in: International Conference on Machine Learning, PMLR, 2019, pp. 2555–2565.

[11] D. Hafner, T. Lillicrap, J. Ba, M. Norouzi, Dream to control: learning behaviors by latent imagination, in: International Conference on Learning Representations, 2020. https://openreview.net/forum?id=S1lOTC4tDS

[12] S. Chiappa, S. Racaniere, D. Wierstra, S. Mohamed, Recurrent environment simulators, arXiv preprint arXiv:1704.02254 (2017).

[13] C. Finn, I. Goodfellow, S. Levine, Unsupervised learning for physical interaction through video prediction, Adv. Neural Inf. Process. Syst. 29 (2016) 64–72.

[14] D. Silver, H. Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, et al., The predictron: end-to-end learning and planning, in: International Conference on Machine Learning, PMLR, 2017, pp. 3191–3199.

[15] E. Banijamali, R. Shu, H. Bui, A. Ghodsi, et al., Robust locally-linear controllable embedding, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2018, pp. 1751–1759.

[16] M.C. Yip, D.B. Camarillo, Model-less feedback control of continuum manipulators in constrained environments, IEEE Trans. Rob. 30 (4) (2014) 880–889.

[17] D. Ha, J. Schmidhuber, World models, arXiv preprint arXiv:1803.10122 (2018).

[18] M. Deisenroth, C.E. Rasmussen, PILCO: a model-based and data-efficient approach to policy search, in: Proceedings of the 28th International Conference on machine learning (ICML-11), Citeseer, 2011, pp. 465–472.

[19] K. Chua, R. Calandra, R. McAllister, S. Levine, Deep reinforcement learning in a handful of trials using probabilistic dynamics models, Adv. Neural Inf. Process. Syst. 31 (2018).

[20] L. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R.H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al., Model based reinforcement learning for Atari, in: International Conference on Learning Representations, 2019.

[21] M. Henaff, W.F. Whitney, Y. LeCun, Model-based planning with discrete and continuous actions, arXiv preprint arXiv:1705.07177 (2017).

[22] B. Amos, L. Dinh, S. Cabi, T. Rothörl, S.G. Colmenarejo, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, M. Denil, Learning awareness models, in: International Conference on Learning Representations, 2018.

[23] P. Agrawal, A.V. Nair, P. Abbeel, J. Malik, S. Levine, Learning to poke by poking: experiential learning of intuitive physics, Adv. Neural Inf. Process. Syst. 29 (2016) 5074–5082.

[24] C. Finn, S. Levine, Deep visual foresight for planning robot motion, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 2786–2793.

[25] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, S. Levine, SOLAR: deep structured representations for model-based reinforcement learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 7444–7453.

[26] M. Watter, J. Springenberg, J. Boedecker, M. Riedmiller, Embed to control: a locally linear latent dynamics model for control from raw images, Adv. Neural Inf. Process. Syst. 28 (2015).

[27] S. Racanière, T. Weber, D.P. Reichert, L. Buesing, A. Guez, D. Rezende, A.P. Badia, O. Vinyals, N. Heess, Y. Li, et al., Imagination-augmented agents for deep reinforcement learning, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 5694–5705.

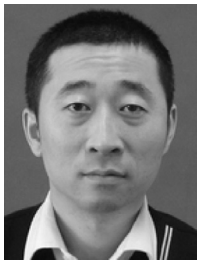
[28] C. Gelada, S. Kumar, J. Buckman, O. Nachum, M.G. Bellemare, DeepMDP: learning continuous latent space models for representation learning, in: International Conference on Machine Learning, PMLR, 2019, pp. 2170–2179.

[29] D. Hafner, T.P. Lillicrap, M. Norouzi, J. Ba, Mastering Atari with discrete world models, in: International Conference on Learning Representations, 2020.

[30] R.S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1–2) (1999) 181–211.

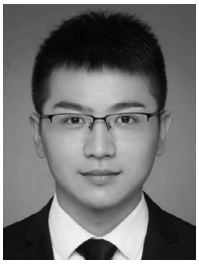
[31] P.-L. Bacon, J. Harb, D. Precup, The option-critic architecture, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[32] A. Jain, K. Khetarpal, D. Precup, Safe option-critic: learning safety in the option-critic architecture, Knowl. Eng. Rev. 36 (2021).
[33] R. Fox, S. Krishnan, I. Stoica, K. Goldberg, Multi-level discovery of deep options, arXiv preprint arXiv:1703.08294 (2017).
[34] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, M.A. Riedmiller, DeepMind control suite, arXiv preprint arXiv:1801.00690 (2018).
[35] G. Barth-Maron, M.W. Hoffman, D. Budden, W. Dabney, D. Horgan, T.B. Dhruva, A. Muldal, N. Heess, T. Lillicrap, Distributed distributional deterministic policy gradients, in: International Conference on Learning Representations, 2018.
[36] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937.



Jiao Wang received PhD degree from Pattern Recognition and Intelligent System of Northeastern University in 2006. He is served as professor in College of Information Science and engineering of Northeastern university. His main research focuses on computer games theory, including Chinese Chess, Go, Phantom Go and War-gaming. His team is top in this field in China.

Lemin Zhang received the BS degree in Automation from Qingdao University of Science and Technology, China, in 2019. Currently, he is pursuing the MS degree in Pattern Recognition and Intelligent Systems from Northeastern University, Shenyang, China. His research interests include deep reinforcement learning and its control applications.



Zhiqiang is currently pursuing the degree with the computer science and technology, Northeastern University. His research interests include model-representations in a reinforcement learning setting and control applications.

Can Zhu received the BS degree in automation from Northeastern University, Shenyang, China, in 2019, where she is currently studying toward the Graduate degree in pattern recognition and intelligent systems. Her research interests include self-driving car and deep reinforcement learning.

Zihui Zhao is studying for a master's degree in pattern recognition and Intelligent Systems, Control Science and Engineering, North-eastern University. Her research interest is the application of deep reinforcement learning in intelligent systems.