Learning to Reweight Imaginary Transitions for Model-Based Reinforcement Learning

Wenzhen Huang,[1,2] **Qiyue Yin,**[1,2] **Junge Zhang,**[1,2] **Kaiqi Huang**[1,2,3]

[1] School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
[2] CRISE, Institute of Automation, Chinese Academy of Sciences, Beijing, China
[3] CAS Center for Excellence in Brain Science and Intelligence Technology, Beijing, China
huangwenzhen2014@ia.ac.cn, {qyyin,jgzhang, kqhuang}@nlpr.ia.ac.cn

Abstract

Model-based reinforcement learning (RL) is more sample efficient than model-free RL by using imaginary trajectories generated by the learned dynamics model. When the model is inaccurate or biased, imaginary trajectories may be deleterious for training the action-value and policy functions. To alleviate such problem, this paper proposes to adaptively reweight the imaginary transitions, so as to reduce the negative effects of poorly generated trajectories. More specifically, we evaluate the effect of an imaginary transition by calculating the change of the loss computed on the real samples when we use the transition to train the action-value and policy functions. Based on this evaluation criterion, we construct the idea of reweighting each imaginary transition by a well-designed meta-gradient algorithm. Extensive experimental results demonstrate that our method outperforms state-of-the-art model-based and model-free RL algorithms on multiple tasks. Visualization of our changing weights further validates the necessity of utilizing reweight scheme.

Introduction

Reinforcement learning (RL) algorithms are typically divided into two categories, i.e., model-free RL and model-based RL. The former directly learns the policy from the interactions with the environment, and has achieved impressive results in many areas, such as games (Mnih et al. 2015; Silver et al. 2016). But these model-free algorithms are data-expensive to train, which limits their applications to simulated domains. Different from model-free approaches, model-based reinforcement learning algorithms learn an internal model of the real environment to generate imaginary data, perform online planning or do policy search, which holds promise to provide significantly lower sample complexity (Luo et al. 2018).

Previously, model-based RL with linear or Bayesian models has obtained excellent performance on the simple low dimensional control problems (Abbeel, Quigley, and Ng 2006; Deisenroth and Rasmussen 2011; Levine and Koltun 2013; Levine and Abbeel 2014; Levine et al. 2016). But these methods are hard to be applied to high-dimensional domains. Since neural network models can represent more complex transition functions, model-based RL with them can solve higher dimensional control problems (Gal, McAllister, and Rasmussen 2016; Depeweg et al. 2017; Nagabandi et al. 2018). However, learned high-capacity dynamics models ineluctably face predicting error, which results in the suboptimal performance and even catastrophic failures (Deisenroth and Rasmussen 2011).

Plenty of approaches have been proposed to alleviate the above problem. For example, (Chua et al. 2018) learns an ensemble of probabilistic models to mitigate the model error. (Clavera et al. 2018) also learns the ensemble of models, and meta-trains a policy to adapt all the models so that the policy can be robust against model-bias. Among this line of research, a type of solution tries to tune model usage to reduce adverse effects of the imaginary data generated by inaccurate models, and promising results have been obtained. (Kalweit and Boedecker 2017) only uses imaginary trajectories in the case of high uncertainties of Q-function. (Heess et al. 2015) only uses imaginary data to compute policy gradients. (Janner et al. 2019) replaces model-generated rollouts begin from the initial state distribution with short model-generated rollouts branched from the real data.

Above simple tuning schemes would result in that the generated data is always ignored in some training processes even it is completely accurate. Since samples with large prediction errors in the imaginary experience will lead to the value or policy function trained on it being inaccurate, adaptively filtering the samples with large prediction errors can reduce the performance degradation caused by the model bias. This makes a basic motivation of our study. However, the prediction error of an imaginary transition is difficult to obtain, because it is hard to decide a threshold of prediction error to determine whether the sample should be abandoned or not. For instance, when the value or policy function is very imprecise, even the samples with relatively large prediction errors can be used to optimize the function.

To handle above predication errors problem, we attempt to adaptively tune model usage through reweighting the imaginary samples according to their potential effect on training, which is totally different from previous model usage approaches. More specifically, we measure the effect through comparing the values of the optimization object (e.g, TD error) computed on the real samples before and after updating the functions using the imaginary transition. In this way, the filtering process can be taken as selecting an appropriate

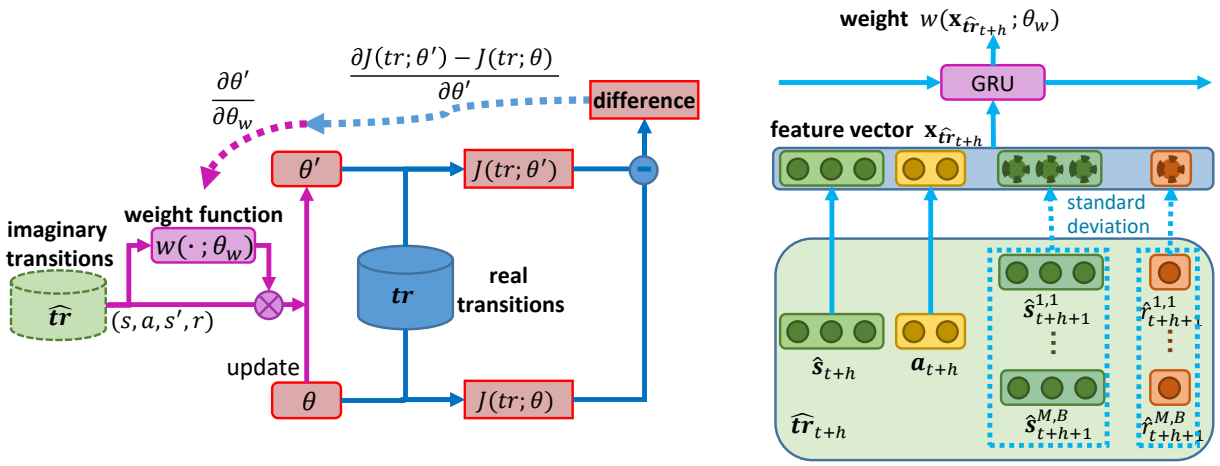


Figure 1: Training architecture (left) and Network architecture (right) for the weight function. We measure the negative effect of reweighted imaginary transitions through computing the difference of the losses computed on the real transitions before and after training with them, and minimize the difference to optimize the weight function by the chain rule.

weight from 0, 1 for each imaginary sample based on its effect. To achieve this, we train a weight function to minimize adverse effects of the samples after they being reweighted using the function. The weight function outputs a weight in the range between 0 and 1 for each transition based on its features, like the uncertainty of the predicted next state in the transition. The effect of a reweighted sample can also be measured by the evaluation criterion mentioned previously.

A main issue of using weight function lies in its optimization. Given a generated transition, a weight is predicted by the weight function and a weighted loss is accordingly calculated for updating parameters. Its effect is evaluated by the difference between the losses computed on the real transitions using the parameters before and after updating. As the loss is parameterized by the updated parameters and the update of parameters is parameterized by the output of the weight function, the function can be optimized through minimizing the difference using the chain rule. Our method can be considered as an instance of meta-gradient (Xu, van Hasselt, and Silver 2018; Zheng, Oh, and Singh 2018; Veeriah et al. 2019), a form of meta-learning (Thrun and Pratt 1998; Finn, Abbeel, and Levine 2017; Hospedales et al. 2020), where the meta-learner is trained via gradients through the effect of the meta-parameters on a learner also trained via gradients (Xu, van Hasselt, and Silver 2018).

To this end, we implement the algorithm by employing an ensemble of bootstrapped probabilistic neural networks and using Soft Actor-Critic (Haarnoja et al. 2018a,b) to update the policy and action-value function. We name this implementation as Reweighted Probabilistic-Ensemble Soft-Actor-Critic (ReW-PE-SAC). Experimental results demonstrate that ReW-PE-SAC outperforms the state-of-the-art model-based and model-free deep RL algorithms on multiple benchmarking tasks. We also analyze the predicted weights on the samples generated with different schemes in different stages of the training process, which shows that the learned weight function can provide reasonable weights for different generated samples in different stages of the training process. In addition, the critic loss updated with the weighted samples is obviously smaller than the one updated with the unweighted samples. This means that the learned weight function can filter out the samples with adverse effects by decreasing their weights.

The main contributions of this work are:

- We propose an effective tuning scheme of model usage through adaptively reweighting the imaginary transitions. Different from the simple tuning schemes proposed by previous works, this theme can adaptively filter generated samples with a certain degree of prediction error based on the precision of action-value and policy functions while maximizing the use of remaining generated samples.
- We use neural networks to predict the weight of each transition in the generated trajectories based on the well-designed features of the transitions and utilize meta-gradient method to optimize the weight network according to the above scheme. Thus, the learned weight network can be applied to new generated samples.
- Experimental results demonstrate that our method outperforms state-of-the-art model-based and model-free RL algorithms on multiple tasks.

Approach

Notation

Considering the standard reinforcement learning setting, an agent interacts with an environment in discrete time. The environment is described by state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, state transition probabilities $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$, and a discount factor $\gamma \in (0, 1]$, where state transition probabilities $p(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ denotes the probability density of the next state $\mathbf{s}_{t+1} \in \mathcal{S}$

given the current state $\mathbf{s}_t \in \mathcal{S}$, action $\mathbf{a}_t \in \mathcal{A}$, and reward function $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ present the reward according to the transition. At each time step t, the agent selects an action $\mathbf{a}_t$ according to the policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$, and then receives the next state $\mathbf{s}_{t+1}$ and the reward r_{t+1} from the environment. The objective of standard reinforcement learning is to learn a policy of the agent to maximize the discount cumulative rewards.

Overall Framework

Model-based reinforcement learning approaches attempt to learn a dynamics model to simulate the real environment and utilize the model to make better decisions. In most cases, the learned model is imperfect and not all the transitions generated by it are accurate, which means the value and policy functions would be misled by the transitions with prediction errors. Therefore, this paper proposes to adaptively reweight the generated transitions to minimize the negative effect of them for the training.

We train a weight function to minimize adverse effects of the transitions after they are reweighted. Specifically, for a transition, the weight function outputs a weight. The effect of a reweighted transition is measured by comparing the losses of value and policy functions computed on the real samples before and after the functions being updated by the reweighted transition. As the loss before being updated is fixed, minimizing the adverse effect is equal to minimizing the loss after being updated. This loss is parameterized by the updated parameters and the update of the parameters is parameterized by the weight function, thus we can optimize the function through minimizing the loss after being updated by the chain rule. The training process of weight function is shown in Figure 1(left).

We employ an ensemble of bootstrapped probabilistic neural networks as the dynamics model, which can provide an estimated uncertainty for each generated transition. The weight function can predict the weights for the transitions more reasonably based on their estimated uncertainties. We use Soft Actor-Critic (Haarnoja et al. 2018a,b) to update the q-value and policy functions, which is an off-policy RL algorithm so that we can use the old experience to evaluate the effect of the updated parameters. We call this implementation as ReWeighted Probabilistic-Ensemble Soft-Actor-Critic (ReW-PE-SAC).

In the following, we would first present how to obtain the ensemble of networks, then describe the network architecture of the weight function, finally explain how to optimize the weight function.

Dynamics Model

In our method, the dynamics model is not only required to generate the transitions, but also needed to provide the other information that is useful for evaluating the weights of these transitions, like uncertainty.

In order to measure the uncertainties of generated transitions, we train an ensemble of B-many bootstrapped probabilistic models like (Chua et al. 2018). The B models have the same architecture but different parameters θ_b and training datasets R_b. Each dataset R_b is generated by sampling with replacement N times from the replay buffer R, where N is equal to the size of R. Each probabilistic model is a neural network that predicts the probability distribution of the next state $\mathbf{s}'$ based on the input state $\mathbf{s}$ and action $\mathbf{a}$. The probability distribution is described by a Gaussian distribution, $\mathcal{N}(\mu_{\theta_b}(\mathbf{s}, \mathbf{a}), \Sigma_{\theta_b}(\mathbf{s}, \mathbf{a}))$. The predicted next state is obtained by sampling from the Gaussian distribution, $\mathcal{N}(\mu_{\theta_b}(\mathbf{s}_n, \mathbf{a}_n), \Sigma_{\theta_b}(\mathbf{s}_n, \mathbf{a}_n))$. Reward function $r(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is assumed as given in advance, like most works of literature related to model-based RL methods (Wang et al. 2019; Clavera et al. 2018; Chua et al. 2018).

Given a state $\mathbf{s}_t$ and an action sequence $\mathbf{a}_{t:t+H-1} = \{\mathbf{a}_t, \ldots, \mathbf{a}_{t+H-1}\}$, the learned dynamics models can induce a distribution over the subsequent trajectories $\mathbf{s}_{t+1:t+H}$. Based on $\mathbf{s}_t$ and $\mathbf{a}_t$, we use the ensemble of probabilistic models to induce B-many Gaussian distributions of the next state $\mathbf{s}_{t+1}$, and then sample M states $\{\hat{\mathbf{s}}_{t+1}^{mb}\}_{m=1}^{M}$ from each Gaussian distributions $\mathcal{N}(\mu_{\theta_b}(\mathbf{s}_t, \mathbf{a}_t), \Sigma_{\theta_b}(\mathbf{s}_t, \mathbf{a}_t))$. The reward function is applied to the predicted next states to evaluate the reward of them, $\hat{r}_{t+1}^{mb} = r(\mathbf{s}_t, \mathbf{a}_t, \hat{\mathbf{s}}_{t+1}^{mb})$. A state is randomly selected from the $M \times B$ predicted states $\{\hat{\mathbf{s}}_{t+1}^{mb}\}_{m=1,b=1}^{M,B}$ as the next input $\hat{\mathbf{s}}_{t+1}$. Then the selected state $\hat{\mathbf{s}}_{t+1}$ and the action $\mathbf{a}_{t+1}$ are used to generate the subsequent $M \times B$ states. In this way, we can get a transition set $\hat{\mathbf{tr}}_k = \{(\hat{\mathbf{s}}_{t+k}, \mathbf{a}_{t+k}, \hat{r}_{t+k}^{b,m}, \hat{\mathbf{s}}_{t+k+1}^{b,m})\}_{m=1,b=1}^{M,B}$ for each time-step $t+k, k = 0, ..., H-1$.

Weight Prediction Network

Estimating the weight on a single generated transition $(\mathbf{s}, \mathbf{a}, \hat{r}, \hat{\mathbf{s}'})$ is difficult, because we cannot obtain any information about the prediction accuracy of $\hat{r}$ and $\hat{\mathbf{s}'}$ from the single transition. Thus, we estimate the weight on the transition set $\hat{\mathbf{tr}}_k = \{(\hat{\mathbf{s}}, \mathbf{a}, \hat{r}^{b,m}, \hat{\mathbf{s}'}^{b,m})\}_{m=1,b=1}^{M,B}$ generated by the ensemble of probabilistic models for the input $(\mathbf{s}, \mathbf{a})$ instead of the single transition.

The weight function $\mathrm{w}(\mathbf{x_{tr}}; \theta_w) : \mathbb{R}^D \rightarrow (0, 1)$ is approximated by a neural network with parameters θ_w, where $\mathbf{x_{tr}}$ represents the feature vector of a generated transition set $\mathbf{tr} = \{(\mathbf{s}, \mathbf{a}, \hat{r}^{b,m}, \hat{\mathbf{s}'}^{b,m})\}_{m=1,b=1}^{M,B}$. The feature vector $\mathbf{x_{tr}}$ is composed of the states $\mathbf{s}$, the actions $\mathbf{a}$, the uncertainty on the predicted reward $\hat{r}$ and the uncertainties on each dimension of the predicted next state $\hat{\mathbf{s}'}$. The uncertainties are approximated by computing the standard deviation of rewards and the next states $\{(\hat{r}^{b,m}, \hat{\mathbf{s}'}^{b,m})\}_{m=1,b=1}^{M,B}$. The uncertainties imply the credibility of the generated transition $\hat{\mathbf{tr}}$, while the inputted state and action uniquely identify the transitions. In practice, we find the latter one enables the weight function to make a better prediction. To avoid the large disparities of different features, the feature vectors are normalized for each dimension before they are fed to the weight network.

It is obvious that the credibility of $\hat{\mathbf{tr}}_k$ is related to the ones of its predecessors $\{\hat{\mathbf{tr}}_j\}_{j<k}$, due to that modeling errors in dynamics are accumulated with time-steps. Thus we select Gated Recurrent Units (GRU) (Cho et al. 2014) to integrate the features of the predecessors. The network archi-

tecture of weight function is shown in Figure 1(right).

Algorithm 1 Reweighted Probabilistic-Ensemble Soft-Actor-Critic (ReW-PE-SAC)

Input: the learning rate μ of θ_q, θ_π, α and the learning rate μ_w of θ_w
Init: initialize parameters θ_q, θ_π, α, θ_w and replay buffer $R \leftarrow \emptyset$
for $t = 1, 2, \ldots, N$ **do**
  Interact with the real environment based on the current policy, and add the transitions to replay buffer R
  Train the dynamics models using replay buffer R

  // Training the weight function
  Generate imaginary transitions $\{\hat{\mathbf{tr}}_h^i\}_{i=1,h=1}^{N_e,H}$
  Sample real transitions $\mathbf{tr}$ from replay buffer R
  Update θ_q, θ_π to θ'_q, θ'_π by Equation 1 using $\{\hat{\mathbf{tr}}_h^i\}_{i=1,h=1}^{N_e,H}$
  Compute the meta objective J_{meta} by Equation 4 on $\mathbf{tr}$
  Approximate the gradient of $\nabla_{\theta_w} J_{meta}$ by Equation 5
  Update $\theta'_w \leftarrow \theta_w - \mu_w \nabla_{\theta_w} J_{meta}$

  // Update value and policy network
  Generate imaginary dataset $\{\hat{\mathbf{tr}}_h^i\}_{i=1,h=1}^{N_t,H}$
  for $k = 1, 2, \ldots, K$ **do**
    Update θ_q, θ_π by Equation 6 on the reweighted imaginary samples
  end for
  Sample real transitions $\mathbf{tr}$ to update θ_q, θ_π
end for

Training the Weight Function

This section will show how to train the weight function so that it can predict appropriate weights for imaginary transitions to minimize their adverse effect.

The training of weight function can be split into two steps, evaluating the potential effects of the reweighted transitions and optimizing the weight function through minimizing the negative effects by the chain rule. We focus on the effects of the action-value and policy functions, and update the weight function through minimizing the effects of a mini-batch of imaginary transitions in each iteration.

For the first step, we sample N_e real states $\{\mathbf{s}_{t_i}^i\}_{i=1}^{N_e}$ and the corresponding real action sequences $\{\mathbf{a}_{t_i:t_i+H-1}^i\}_{i=1}^{N_e}$ from the replay buffer $\mathbb{D}$ to generate the imaginary transitions $\{\hat{\mathbf{tr}}_h^i\}_{i=1,h=1}^{N_e,H}$, where H is the planning horizon. Then we compute the weights of imaginary transitions $\mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w)$ and update the parameters of Q-network and policy network, θ_q and θ_π, with the reweighted losses of these imaginary samples:

$$\begin{aligned}\theta'_q &= \theta_q - \mu \frac{\partial \sum_{i,h} \mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w) J_Q(\hat{\mathbf{tr}}_h^i; \theta_q)}{\partial \theta_q}, \\ \theta'_\pi &= \theta_\pi - \mu \frac{\partial \sum_{i,h} \mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w) J_\pi(\hat{\mathbf{tr}}_h^i; \theta_\pi)}{\partial \theta_\pi},\end{aligned} \tag{1}$$

where μ is the learning rate of θ_q and θ_π. J_Q and J_π are the soft Bellman residual and the KL-divergence between the policy and the exponential of the soft Q-function (Haarnoja et al. 2018a,b), respectively. For a transition set $\mathbf{tr}$, J_Q and J_π are computed by

$$\begin{aligned}J_Q(\mathbf{tr}; \theta_q) = \sum_{(\mathbf{s},\mathbf{a},\mathrm{r},\mathbf{s}') \in \mathbf{tr}} \frac{1}{2} \big\{ Q(\mathbf{s}, \mathbf{a}; \theta_q) - \\ [\mathrm{r} + \gamma (Q(\mathbf{s}', \mathbf{a}'; \bar{\theta}_q) - \alpha \log \pi(\mathbf{a}'|\mathbf{s}'))] \big\}^2, \end{aligned} \tag{2}$$

$$J_\pi(\mathbf{tr}; \theta_\pi) = \sum_{(\mathbf{s},\mathbf{a},\mathrm{r},\mathbf{s}') \in \mathbf{tr}} \alpha \log(\pi(\hat{\mathbf{a}}|\mathbf{s}_t; \theta_\pi)) - Q(\mathbf{s}_t, \hat{\mathbf{a}}; \theta_q), \tag{3}$$

where $\bar{\theta}_q$ is the parameters of target Q-network, and α is the temperature parameter.

For the second step, we sample N_v real transitions from the replay buffer $\mathbb{D}$, combined them into a set $\mathbf{tr}$, and compute the losses of q-value and policy functions on them with the updated parameters θ'_q and θ'_π,

$$J_Q(\mathbf{tr}; \theta'_q) + J_\pi(\mathbf{tr}; \theta'_\pi). \tag{4}$$

The gradient of the parameters of weight function θ_w is computed through the chain rule,

$$\begin{aligned}&\frac{\partial J_Q(\mathbf{tr}; \theta'_q) + J_\pi(\mathbf{tr}; \theta'_\pi)}{\partial \theta_w} \\ =&\frac{\partial J_Q(\mathbf{tr}; \theta'_q)}{\partial \theta'_q} \frac{\partial \theta'_q}{\partial \theta_w} + \frac{J_\pi(\mathbf{tr}; \theta'_\pi)}{\partial \theta'_\pi} \frac{\partial \theta'_\pi}{\partial \theta_w} \\ =&-\mu \sum_{h,i} \big[(\frac{\partial J_Q(\hat{\mathbf{tr}}_h^i; \theta_q)}{\partial \theta_q})^T \frac{\partial J_Q(\mathbf{tr}; \theta'_q)}{\partial \theta'_q} \\ &+ (\frac{\partial J_\pi(\hat{\mathbf{tr}}_h^i; \theta_\pi)}{\partial \theta_\pi})^T \frac{\partial J_\pi(\mathbf{tr}; \theta'_\pi)}{\partial \theta'_\pi} \big] \frac{\partial \mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w)}{\partial \theta_w}\end{aligned} \tag{5}$$

Once the gradient is obtained, the parameters θ_w can be updated by any optimization algorithm.

We alternately optimize the q-value, policy functions, and the weight function, so that the latter one can adaptively adjust the weights of imaginary transitions along with the change of the precision of the former ones. We sample N_t real states and the corresponding action sequences with an explore policy π_e which is obtained by changing the temperature parameter of current policy from α to $\lambda_e \alpha$ (λ_e is set to 10 in this paper). A larger temperature parameter is conducive to generating diverse transitions. Based on the sampled state and action sequences, we utilize the dynamics model to generate imaginary transitions $\{\hat{\mathbf{tr}}_h^i\}_{i=1,h=1}^{N_t,H}$ and use the weight function to reweight them. The gradients of q-value and policy functions are computed by

$$\begin{aligned}\nabla \theta_q &= \frac{\partial \sum_{i,h} \mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w) J_Q(\hat{\mathbf{tr}}_h^i; \theta_q)}{\partial \theta_q}, \\ \nabla \theta_\pi &= \frac{\partial \sum_{i,h} \mathrm{w}(\mathbf{x}_{\hat{\mathbf{tr}}_h^i}; \theta_w) J_\pi(\hat{\mathbf{tr}}_h^i; \theta_\pi)}{\partial \theta_\pi}.\end{aligned} \tag{6}$$

	Ant	HalfCheetah	Hopper	Slimhumanoid	Swimmer	Walker2d
ME-TRPO	282.2±18.0	2283.7±900.4	1272.5±500.9	-154.9±534.3	30.1±9.7	-1609.3±657.5
MB-MPO	705.8 ± 147.2	3639.0±1185.8	333.2±1189.7	674.4±982.2	**85.0±98.9**	-1545.9±216.5
PETS	1165.5±226.9	2795.3±879.9	1125.0±679.6	1472.4±738.3	22.1±25.2	260.2±536.9
POPLIN	2330.1±320.9	4235.0±1133.0	2055.2±613.8	-245.7±141.9	37.1±4.6	597.0±478.8
MBPO	4332.5±1277.6	10758.9±1413.7	**3279.8±455.0**	2950.4±819.1	26.3±13.3	4154.7±846.1
TD3	956.1±66.9	3614.3±82.1	2245.3±232.4	1319.1±1246.1	40.4±8.3	-73.8±769.0
SAC-200k	922.0±283.0	6129.3±775.7	2365.1±193.4	1891.6±379.2	49.7±5.8	1642.7±606.9
w.o reweighting	4033.5±1480.5	**11854.3±102.8**	2202.6±363.5	1436.8±490.8	26.6±25.4	2673.8±2264.8
Our Method	**4614.4±931.1**	9779.8±546.6	2824.0±159.9	**11755.9±11152.2**	**82.2±33.4**	**4961.9±457.8**
SAC-1000k	4994.9±719.5	10283.8±648.4	2990.3±214.3	29122.5±11129.0	86.8±6.4	5094.0±1371.3

Table 1: Final performance on the six environments. All the algorithms are run for 200k time-steps (except SAC-1000k). The results are shown with the mean and standard deviation averaged and a window size of 5000 times-steps.

We use Adam to update the parameters θ_q and θ_π. The temperature parameter α is optimized based on the generated transition sets without being reweighted.

The complete algorithm is shown in Alg. 1. In our algorithm, the real transitions are not only used to train the dynamics models, but also used to train the action-value and policy networks. The real samples can avoid too large prediction errors of the action-value function. When the predicted weights of generated samples are too low, the real samples can prevent algorithm from being in stagnation behavior.

Experiments

In this section, we evaluate our algorithm on six complex continuous control tasks from the model-based RL benchmark (Wang et al. 2019), which is modified from the OpenAI gym benchmark suite (Brockman et al. 2016). The six tasks are Ant, HalfCheetah, Hopper, SlimHumanoid, Swimmer-v0, and Walker2D, whose horizon length is fixed to 1000. The network architecture and training hyperparameters are given in the appendix. First, we compare ReW-PE-SAC on the benchmark against state-of-the-art model-free and model-based approaches. Then, we show the differences of the q-value losses with and without reweighting method. Next, we evaluate the robustness of our algorithm to imperfect dynamics model. Finally, we analyze the relation between the learned weights and the factors of the training iterations, the planning horizon, and the explore policy.

Comparison with State of the Art

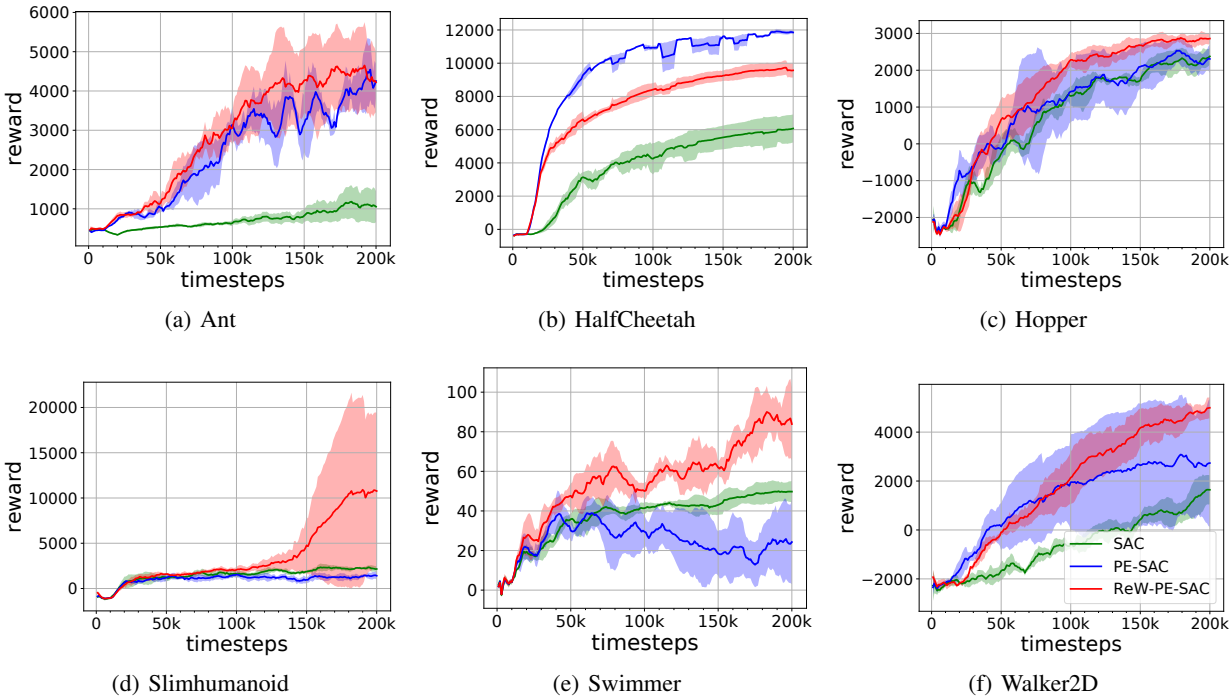
We compare ReW-PE-SAC with state-of-the-art model-free and model-based RL methods, including SAC (Haarnoja et al. 2018a,b)[1], TD3 (Fujimoto, Hoof, and Meger 2018), ME-TRPO (Kurutach et al. 2018), MB-MPO(Clavera et al. 2018), PETS (Chua et al. 2018), MBPO (Janner et al. 2019) and POPLIN (Wang and Ba 2019). We reproduce results from (Wang et al. 2019; Janner et al. 2019) and additionally run MBPO on the tasks of Slimhumanoid and Swimmer as the according experimental results are absent. We run our method ReW-PE-SAC for $200,000$ time-steps with 8 random seeds. To evaluate our reweighting mechanism, we also run PE-SAC on these six tasks which does not learn the weight function and directly use the imaginary transitions to train the policy and value networks. To measure the sample efficiency of ReW-PE-SAC, we additionally run SAC $1,000,000$ time-steps on each task. The results are summarized in Table 1, and the learning curves of SAC and our methods with or without reweighting are plotted in Figure 2.

As shown in Table 1, ReW-PE-SAC achieves better performance compared with all other state-of-the-art algorithms except MBPO running with $200,000$ time-steps in all the environments. Especially in the environments of Ant, Hopper, Swimmer and Walker2d, the performance of ReW-PE-SAC is comparable to the one of SAC running with $1,000,000$ time-steps, which demonstrates that ReW-PE-SAC has good sample efficiency. Compared with MBPO, ReW-PE-SAC is better on four environments and is slightly weaker in the tasks of HalfCheetah and Hopper.

Comparing the results of our methods with and without reweighting, ReW-PE-SAC and PE-SAC, the performance with reweighting is obviously higher on the most of the environments. This demonstrates that the learned weight function can provide appropriate weights to facilitate training a better policy. The performance gap of ReW-PE-SAC and PE-SAC on the environment of HalfCheetah is probably caused by that the weight function is overcautious, and the weights provided by it are too low.

From Figure 2(a,d), we find our method has a large performance variance in the tasks of Ant and Slimhumanoid. The most likely reason is that our method utilizes the collected transitions to evaluate the effect of imaginary transitions, while the number of collected transitions is insufficient for some tasks. This induces that the weights of some valid imaginary transitions could be underestimated, and then the learned policy would be relatively poor due to the lack of these valid transitions. We will consider constructing a more

[1]We select the PyTorch implement of soft actor-critic in https://github.com/pranz24/pytorch-soft-actor-critic to evaluate the performance. This implement includes using double-Q network, ignoring the artificial terminal signal and other tricks, so the performance is better than the one reported in (Wang et al. 2019).



(a) Ant (b) HalfCheetah (c) Hopper

(d) Slimhumanoid (e) Swimmer (f) Walker2D

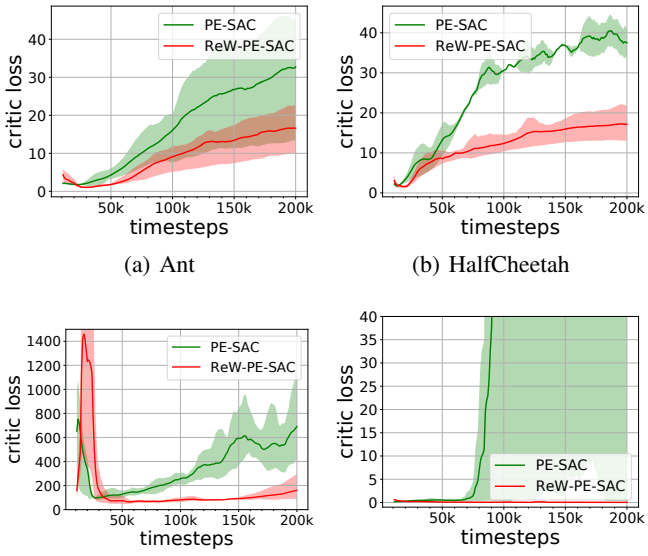
Figure 2: Learning curves for different tasks and algorithms. All the algorithms are run for 200k time-steps with 8 random seeds.

reasonable validation set in future work.

The Critic Losses of PE-SAC and ReW-PE-SAC

In this section, we compare the critic losses in cases with and without reweighting. We run the algorithms of PE-SAC and ReW-PE-SAC on the tasks of Ant, HalfCheetah, SlimHumanoid and Swimmer, and record the average critic losses of real samples in every episode. The minimum, maximum and mean of the losses in the same time-step are plotted in Figure 3.

As shown in the figure, ReW-PE-SAC can maintain lower critic losses than PE-SAC and prevent abnormal large losses. Combined with the learning curve for the task of Swimmer (shown in Figure 2(e)), we find the performance of PE-SAC is falling after about $70,000$ time-steps while the critic loss is also increasing sharply at around this time. So maintaining lower losses has contributed to improve the performance in most cases. The only exception is the task of HalfCheetah, in which the lower critic losses have not resulted in higher performance. The most likely reason is that an imprecise Q-value function is enough to train a good policy.



(a) Ant (b) HalfCheetah

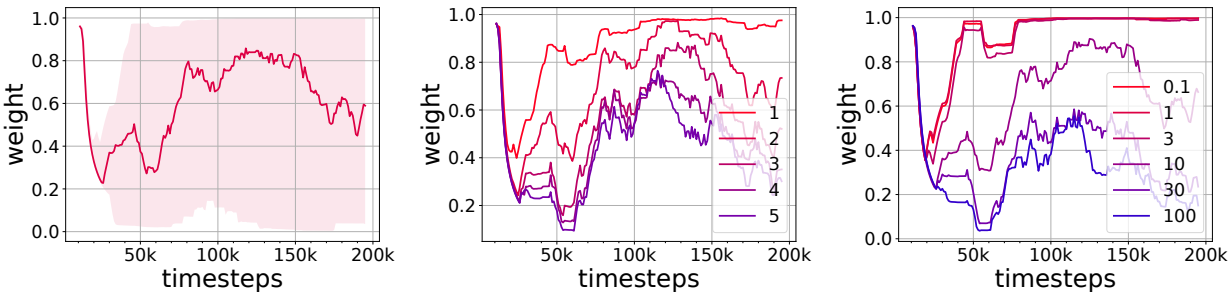
(c) SlimHumanoid (d) Swimmer

Figure 3: Critic losses in the cases with and without reweighting. The x-axis corresponds to time-step. The y-axis corresponds to average critic loss over 1 episode (1000 time-steps).

Robustness to Imperfect Dynamics Model

We construct the dynamics models with different prediction accuracy through adjusting the number of the hidden layers in them from 4 to 2. We run the algorithms of PE-SAC and ReW-PE-SAC with these dynamics models on the tasks of Ant. The learning curves of them are plotted in Figure 5.

When the number of the hidden layers is decreased, the performances of PE-SAC drops significantly. This means



(a) Overall Trend (b) Different Prediction Depth (c) Different Soft Scale

Figure 4: Predicted weights for different generated samples in different stages of training process.

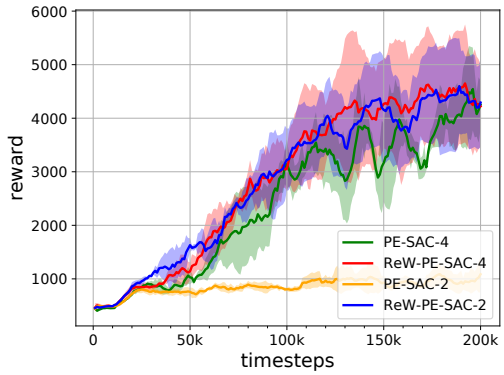


Figure 5: Learning curves for PE-SAC and ReW-PE-SAC with the dynamics models using different numbers of hidden layers.

that the dynamics models with 2 hidden layers have strong negative effect on the training process. The performances of ReW-PE-SAC remain roughly unchanged, which means that our method can effectively reduce the negative effect of the generated samples with prediction errors. The above analysis gives a possible explanation for the phenomenon that ReW-PE-SAC has higher performance improvement on the more complex tasks, like Slimhumanoid and Walker2d.

The Trend of the Predicted Weight

In this section, we analyze the overall trend of the predicted weights and the relation between the weights and the prediction depth and the soft scale λ_e. We run the algorithms of ReW-PE-SAC on the task of Swimmer with only 1 random seed, and record the predicted weights of generated samples at the first step of each episode. The predicted weights are changed with the process of training, so computing the average on different seeds is meaningless. The 25 precent point, median and 75 precent point are plotted in Figure 4(a). Then, we split these weights according to the prediction depth, and plot the median of the weights of different prediction depth in Figure 4(b). Finally, we generate some extra data using different $\lambda_e \in \{0.1, 1, 3, 10, 30, 100\}$, and plot the median of predicted weights on them in Figure 4(c).

In Figure 4(a), the weights are lower in the earlier and later stages but are higher in the middle stage (The weight function's initial output is about 0.95 as that the bias of last layer is initialized to 3.0.). The trend reflects the change of the accuracy of the dynamics model and the q-value and policy functions. In the earlier stage, the dynamics model is imprecise, so most of the generated transitions are rejected. Then, the weights become to increase as the improvement in the prediction precision of the dynamics model. However, in the later stage, the precision of q-value function also improves, while the model has reached its bottleneck. This results in the decline of the weights. From Figure 4(b), we find that the predicted weights decrease with the planning steps which accords with the fact that the prediction errors accumulates with steps. From Figure 4(c), we also found that the weights decrease with the scale which is caused by the difference of the distributions of the actions in the training and predicting process of the dynamics model. These phenomenons further verify that the learned weight function is reasonable.

Conclusion

In this paper, we have proposed a novel and efficient model-based reinforcement learning approach, which adaptively adjusts the weights of all generated transitions through training a weight function to reduce the potential negative effect of them. We measure the effect of reweighted imaginary transitions through computing the difference of the losses computed on the real transitions before and after training with them, and minimize the difference to optimize the weight function by the chain rule.

Experimental results show that our method obtains the state-of-the-art performance on multiple complex continuous control tasks. The learned weight function can provide reasonable weights for different generated samples in different stages of training process. We believe that the weight function can be utilized to adjust some hyper-parameters, like planning horizon, in the future.

Acknowledgments

This work is funded by the National Natural Science Foundation of China (Grand No. 61876181 No. 61673375 and No.61721004), Beijing Nova Program of Science and Technology under Grand No. Z191100001119043, the Youth Innovation Promotion Association, and CAS and the Projects of Chinese Academy of Science (Grant No. QYZDB-SSW-JSC006).

References

Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd International Conference on machine learning (ICML-06)*, 1–8.

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

Cho, K.; Van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Empirical Methods in Natural Language Processing*, 1724–1734.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 4754–4765.

Clavera, I.; Rothfuss, J.; Schulman, J.; Fujita, Y.; Asfour, T.; and Abbeel, P. 2018. Model-Based Reinforcement Learning via Meta-Policy Optimization. In *Conference on Robot Learning*, 617–629.

Deisenroth, M.; and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 465–472.

Depeweg, S.; Hernández-Lobato, J.; Doshi-Velez, F.; and Udluft, S. 2017. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on machine learning (ICML-17)*, 1126–1135. JMLR. org.

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on machine learning (ICML-18)*, 1587–1596.

Gal, Y.; McAllister, R.; and Rasmussen, C. E. 2016. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, 34.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018a. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on machine learning (ICML-18)*, 1861–1870.

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. 2018b. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* .

Heess, N.; Wayne, G.; Silver, D.; Lillicrap, T.; Erez, T.; and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2944–2952.

Hospedales, T.; Antoniou, A.; Micaelli, P.; and Storkey, A. 2020. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439* .

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, 12498–12509.

Kalweit, G.; and Boedecker, J. 2017. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, 195–206.

Kurutach, T.; Clavera, I.; Duan, Y.; Tamar, A.; and Abbeel, P. 2018. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592* .

Levine, S.; and Abbeel, P. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, 1071–1079.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17(1): 1334–1373.

Levine, S.; and Koltun, V. 2013. Guided policy search. In *Proceedings of the 30th International Conference on machine learning (ICML-13)*, 1–9.

Luo, Y.; Xu, H.; Li, Y.; Tian, Y.; Darrell, T.; and Ma, T. 2018. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533.

Nagabandi, A.; Kahn, G.; Fearing, R. S.; and Levine, S. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7559–7566. IEEE.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *NATURE* 529(7587): 484.

Thrun, S.; and Pratt, L. 1998. Learning to learn: Introduction and overview. In *Learning to learn*, 3–17. Springer.

Veeriah, V.; Hessel, M.; Xu, Z.; Rajendran, J.; Lewis, R. L.; Oh, J.; van Hasselt, H. P.; Silver, D.; and Singh, S. 2019. Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, 9306–9317.

Wang, T.; and Ba, J. 2019. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649* .

Wang, T.; Bao, X.; Clavera, I.; Hoang, J.; Wen, Y.; Langlois, E.; Zhang, S.; Zhang, G.; Abbeel, P.; and Ba, J. 2019. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057* .

Xu, Z.; van Hasselt, H. P.; and Silver, D. 2018. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, 2396–2407.

Zheng, Z.; Oh, J.; and Singh, S. 2018. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, 4644–4654.