Replication: MAXIMUM A POSTERIORI POLICY OPTIMISATION

Yi-Hsin Chen, Zhi-Yi Chin, Yu-Hsuan Li, Yu-Jie Chen
Department of Computer Science
National Chiao Tung University
{yhchen12101.cs09,joycenerd.cs09,evali890227.cs09,cyj407.cs09}@nycu.edu.tw

1 Problem Overview

The author proposed an reinforcement learning algorithm based on the EM algorithm (Moon [1996]) and used the thought of RL in inference to optimize the maximum probability of a trajectory at each step. Based on this, the paper came up with two MPO algorithms, one is similar to PPO (Schulman et al. [2017]) and TRPO (Schulman et al. [2015]), and the other is a new algorithm.

This paper achieves its goal with the help of an additional distribution. Therefore, not only the parameters of the model but also the distribution have to be trained. In order to train the two things, the author thus used EM algorithm: tune the distribution in E-step and train the model parameters in M-step.

2 Background and The Algorithm

2.1 Policy Improvement

Assume the event $O = 1$ be interpreted as obtaining maximum reward by choosing an action and assuming $p(O = 1|\tau) \propto \exp(\sum_t r_t/\alpha)$, which α is a temperature parameter. The goal is to optimize the strategy to maximize the probability of the event, that is, to maximize the following objectives:

$$\log p_\pi(O = 1) = \log \int p_\pi(\tau) p(O = 1|\tau) d\tau \geqslant \int q(\tau)[\log p(O = 1|\tau) + \log \frac{p_\pi(\tau)}{q_\pi(\tau)}]d\tau \quad (1)$$

$$= \mathbb{E}[\sum_t r_t/\alpha] - \mathrm{KL}(q(\tau)||p_\pi(\tau)) = \mathcal{J}(q, \pi) \quad (2)$$

Equation (1) and (2) uses an auxiliary distribution $q(\tau)$ to substitute the real (but difficult to get) distribution $p_\pi(\tau)$, also uses the evidence lower bound (ELBO) to get the lower bound of the formula. Thus, an RL problem becomes an inference problem, that we can use the EM algorithm to solve this problem.

The above optimization goal is expressed based on the trajectory; in the convenience of optimization, we generally hope to write it based on a single-step transition. Let $q(\tau) = p(s_0) \prod_{t>0} p(s_{t+1}|s_t, a_t) q(a_t|s_t)$; the above optimization goal can be rewritten as (note that KL over trajectories decomposes into a KL over the individual state-conditional action distributions):

$$\mathcal{J} = \mathbb{E}_{q(\tau), s_0=s, a_0=a}[\sum_{t\geq 1}^{\infty} \gamma^t [r_t - \alpha \mathrm{KL}(g_t||\pi_t)]] \quad (3)$$

At the same time, defined a regularized Q function:

$$Q^q_\theta(s, a) = r_0 + \mathbb{E}_{q(\tau) s_0=s, a_0=a}[\sum_{t\geq 1}^{\infty} \gamma^t r_t] \quad (4)$$

Also, $\mathrm{KL}(q_t||\pi_t) = \mathrm{KL}(q(a|s_t)||\pi(a|s_t, \theta))$, and the corresponding reward $\tilde{r}_t = r_t - \alpha \log \frac{q(a_t|s_t)}{\pi(a_t|s_t,\theta)}$

Preprint. Under review.

2.2 E step

E step aims to update q to optimize $\mathbb{J}$. Still, in the expression of the objective function $\mathbb{J}$, the q distribution controls the trajectory of the states; therefore, optimizing the objective function isn't easy. Therefore, there is no way to think that the trajectory of the state is still the original trajectory but to maximize the probability of selecting different actions in each state (according to q) to do one-step optimization. That optimizes the formula below:

$$\max_q \bar{\mathcal{J}}_s(q, \theta_i) = \max_q T^{\pi,q} Q_{\theta_i}(s, a) = \max_q \mathbb{E}_{\mu(s)}[\mathbb{E}_{q(\cdot|s)}[Q_{\theta_i(s,a)}] - \alpha \mathrm{KL}(q||\pi_i)] \tag{5}$$

Also, Bellman operator $T^{\pi,q} = \mathbb{E}_{q(a|s)}[r(s, a) - \alpha \mathrm{KL}(q||\pi_i) + \gamma \mathbb{E}_{p(s'|s,a)}[V_{\theta_i}(s')]]$, thus we can choose $q_i = \arg\max \bar{\mathcal{J}}(q, \theta_i)$ in every E step. Since the penalty term α isn't easy to determine, in contrast, KL divergence is more invariant (under different circumstances), therefore came up with the hard constraint version below (similar to TRPO):

$$\max_q \mathbb{E}_{\mu(s)}[\mathbb{E}_{q(a|s)}[Q_{\theta_i(s,a)}]] s.t. \mathbb{E}_{\mu(s)}[\mathrm{KL}(q(a|s), \pi(a|s, \theta_i))] < \epsilon \tag{6}$$

There are two ways to solve this optimization problem:

1. Use parametric variational distribution: think that q is a parameterized model, so we can directly optimize the above goal and get a new q; we can observe that q can be used as a new policy, so there is no need to do M-step because M-step is to minimize KL divergence of the distribution q and the policy π, we can directly set the policy π to the distribution q. TRPO and PPO are optimized based on Equation 6.
2. Use non-parametric representation: To express $q(s, a)$ in a non-parametric way, that is, to express their values on the collected samples; in this way, an additional M-step is needed to obtain a generalizable representation of the policy. Compared with the parametric representation method, this method is more innovative. The following mainly talks about how to implement this method.

There is a close form solution for Equation 6:

$$q_i(a|s) \propto \pi(a|s, \theta_i) \exp(\frac{Q_{\theta_i}(s, a)}{\eta^*}), \tag{7}$$

where we can obtain η^* by minimizing the following convex dual function,

$$g(\eta) = \eta\epsilon + \eta \int \mu(s) \log \int \pi(a|s, \theta) \exp(\frac{Q_{\theta_i}(s, a)}{\eta}) dads \tag{8}$$

In the actual algorithm, several actions can be collected for each state sample, and then the extreme points of Equation 8 can be calculated first, and then Equation 7 can be used to calculate q on each state-action sample. Next, according to the q on these state-action samples, a generalizable policy network, namely M-step, is obtained through supervised learning.

2.3 M step

The goal of E-step is $q_i = \arg\max \bar{\mathcal{J}}(q, \theta_i)$; on the other hand, the goal of M-step is $\theta_{i+1} = \arg\max_\theta \mathcal{J}(q_i, \theta)$, which is as follow:

$$\max_\theta \mathcal{J}(q_i, \theta) = \max_\theta \mathbb{E}_{\mu_q(s)}[\mathbb{E}_{q(a|s)}[\log \pi(a|s, \theta)]] + \log p(\theta) \tag{9}$$

Then, choose a Gaussian prior $p(\theta) \approx \mathcal{N}(\mu = \theta_i, \Sigma = \frac{F_{\theta_i}}{\lambda})$, where θ_i are the parameters of the current policy distribution, F_{θ_i} is the empirical Fisher information matrix, and λ is a positive scalar. Then Equation (9) can become as follow:

$$\max_\pi [\mathbb{E}_{\mu_q(s)}[\log \pi(a|s, \theta)] - \lambda \mathrm{KL}(\pi(a|s, \theta_i), \pi(a|s, \theta))] \tag{10}$$

Similarly, use hard constraints and iteratively optimize Lagrangian multipliers and parameters.

2.4 Policy evaluation

In this paper, the policy improvement step is dependent on the information given by the Q function, so the accuracy of the estimation of the Q function will directly affect whether this algorithm success or not. Because this paper will do an off-policy algorithm to improve sample efficiency, therefore, the estimation of the Q function uses Retrace to estimate:

$$\begin{aligned}
&\min_{\phi} L(\phi) = \min_{\phi} \mathbb{E}_{\mu_b(s),b(a|s)}[(Q_{\theta_i}(s_t, a_t, \phi) - Q_t^{\text{ret}})^2], \text{with} \\
&Q_t^{\text{ret}} = Q_{\phi'}(s_t, a_t) + \sum_{j=t}^{\infty} \gamma^{j-t} (\prod_{k=t+1}^{j} c_k)[r(s_j, a_j) + \mathbb{E}_{\pi(a|s_{j+1})}[Q_{\phi'}(s_{j+1}, a)] - Q_{\phi'}(s_j, a_j)], \\
&c_k = \min\left(1, \frac{\pi(a_k|s_k)}{b(a_k|s_k)}\right)
\end{aligned} \tag{11}$$

3 Detailed Implementation

Since the original paper mentioned two cases, non-parametric and parametric one, we implement the former one which has better performance prominently than other algorithms. In this section, we separate to two parts. One is the model and module architecture, and the other is the training procedure, which mainly focus on the algorithm implementation.

3.1 Models and Modules

3.1.1 Actor

As the common deep reinforcement learning methods, Actor is used to describe the policy actually performing the decisions. We assume that the policy is given by a Gaussian distribution with a full covariance matrix, i.e, $\pi(a|s, \theta) = N(\mu, \Sigma)$. Our neural network outputs the mean μ and Cholesky factor A such that $\Sigma = AA^T$. The lower triangular factor A has positive diagonal elements enforced by the softplus transform $A_{ii} \leftarrow \log(1 + \exp(A_{ii}))$.

The architecture of Actor is shown in Figure 1. We construct a three-layer structure to produce the μ and A. The output dimension of μ would be the dimension of actions, and the dimension of A would be the same as the covariance matrix, which is $\frac{n(n+1)}{2}$, and n is the dimensions of actions. After we gain the μ and A, we could get the corresponding Gaussian distribution to sample some actions.

3.1.2 Critic

The target of Critic is to evaluate the state. Consequently, it would output the Q-value according to the input state s and action a. Critic share similar architectures with Actor. The first two layers are the same as Actor, which are fully-connected with ReLU as the activation function. The last layer only produce one dimension, which is the Q-value. The structure is shown in Figure 1.

3.1.3 Replay Buffer

Since MPO algorithm is off-policy, we write a replay buffer to store episodes. The replay buffer save episodes, and each episode contains the state, the action, the next state, and the reward.

3.2 Training Procedure

We will introduce our setting first. The whole algorithm could separate to three parts, policy evaluation, E step, and M step. The procedure of the non-parametric case is described in Algorithm 1.

3.2.1 Initialization and Setup

Unlike the most RL paper using gym as the environments, MPO chooses DeepMind Control Suite Tassa et al. [2018] as their environments. The most differences between these two environments

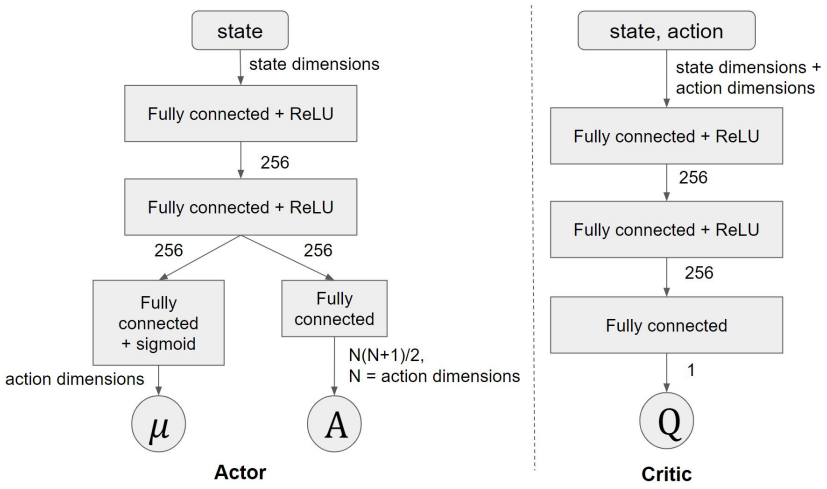

Figure 1: Actor and Critic Architecture

Hyperparameter	values
Dual Constraints ϵ	0.1
Decoupling mean ϵ_μ	0.1
Decoupling covariance ϵ_Σ	0.0001
Discount factor (γ)	0.99
Adam learning rate	0.0003

Table 1: Parameters for non parametric variational distribution

is that they uses different sorts of data structures to describe actions and observations. **Since the control suite uses quite complicated data structures, we are struggling with transforming it to fit Pytorch tensor.** As a consequence, we find a useful package 'dm_control2gym'(Seilair [2018]) to simply convert the environments from the control suite to gym environment and enable visualization. The 'dm_control2gym' package provides the wrapper and viewer which allows us to directly operate the data structure as gym environments.

In the implementation stage, first of all, we need to set up everything. For the models, we initialize the two actors and two critics. One actor is the target one, also regarded as the old one, and the other one is we actually perform. So do the two critics. Besides, we also set up some hyperparameters, which is shown in Table 1. Most values are the same as the original MPO paper.

The description of Algorithm 1 is a little bit different to the original MPO paper. We modify the loop condition that makes iteration loop as the outer one, and the batches as the inner loop which is easier to understand. We set up our default number of re-run episodes L_{max} to 3, that is, batches will be re-sampled for 3 times from the current replay buffer in the inner loop. In short, the whole procedure in the outer loop in each iteration is that the replay buffer would sample trajectories and calculate the mean return and mean reward in the current replay buffer.

3.2.2 Policy Evaluation

To simplify, for the policy evaluation part, we do not follow Eq. 11 in our implementation but use TD(0). In specific, we follow the following steps to update the critic network: (1) take samples from the replay buffer, which contains (state, action, reward, next state); (2) for each sample, we predict the policy distributions of the next state by target actor network; (3) sample 64 actions from policy distribution for the next state; (4) compute expected Q value of the next state by target critic network with next state, and sampled next actions; (5) compute Q value of the state with critic network by state and action; (6) use TD(0) with Q value of the state, expected Q value of next state and reward to update the critic network.

Algorithm 1 MPO (Non parametric variational distribution)

Input = ϵ, ϵ_Σ, ϵ_μ, L_{max}
$i = 0$
Initialise $Q_{\omega_i}(a, s), \pi(a|s, \theta_i), \eta, \eta_\mu, \eta_\Sigma$
for each worker **do**
    $k = 0$
    while $k < 1000$ **do**
        update replay buffer $\mathcal{B}$ with L trajectories from the environment
        // Find better policy by gradient descent
        $L_{curr} = 0$
        while $L_{curr} < L_{max}$ **do**
            sample a mini-batch $\mathcal{B}$ of $N(s, a, r)$ pairs from replay
            sample M additional actions for each state from $\mathcal{B}$,$\pi(a|s, \theta_i)$ for estimating integrals
            compute gradients, estimating integrals using samples
            // Q-function gradient:
            $\delta_\phi = \partial_\phi L'_\phi(\phi)$
            // E-Step gradient:
            $\delta_\eta = \partial_g(\eta)$
            Let: $q(a|s) \propto \pi(a|s, \theta_i) \exp(\frac{Q_{\theta_t}(a,s,\phi')}{\eta})$
            // M-Step gradient:
            $[\delta_{\eta_\mu}, \delta_{\eta_\Sigma}] = \alpha \partial_{\eta_\mu, \eta_\Sigma} L(\theta_k, \eta_\mu, \eta_\Sigma)$
            $\delta_\theta = \partial_\theta L(\theta, \eta_{\mu_{k+1}}, \eta_{\Sigma_{k+1}})$
            send gradients to chief worker
            wait for gradient update by chief
            fetch new parameter $\phi, \theta, \eta, \eta_\mu, \eta_\sigma$
            $L_{curr} = L_{curr} + 1$
        $i = i + 1$
        $k = k + 1$
        $\theta_i = \theta, \phi' = \phi$

3.2.3 E step

In the E-step of policy improvement, we follow the steps we describe in Section 2.2. Our goal is to find the optimized distribution q, which can be obtained by current policy, Q-value function and eta according to Eq.7. Q-value function Q should store all possible state-action pair, which cannot be fulfilled in continuous environment. Hence, we sample actions for every states in a mini-batch to build Q. Second, we can optimize the dual function either by Eq. 14 or by Eq.8 to get the η^* of Eq.7.

It seems that we still need policy $\pi(a|s, \theta)$; however, note that the reason we multiply $\pi(a|s, \theta)$ is to add the weight of each action. That is, we care more about the actions that appear more, which is implicit in sample. (The more often to be applied, the easier to be sampled). Therefore, we do not have to times the policy.

There is one more trick in this implementation: normalized. It is known that for every single state, the sum of the probability to all actions is one, which can be maintain by a normalization function. Another way to write Eq.7 is:

$$q_i(a|s) = \pi(a|s, \theta_i) \exp(\frac{Q_{\theta_i}(s, a)}{\eta}) \exp(-\frac{\eta - \gamma}{\eta})$$

where $\exp(-\frac{\eta-\gamma}{\eta})$ is the normalization term, equal to $\int \pi(a|s, \theta_i) \exp(\frac{Q_{\theta_i}(a,s)}{\eta})$. Just like mentioned before, we could omit $\pi(a|s, \theta)$, leading to

$$q_i(a|s) = \exp(\frac{Q_{\theta_i}(s, a)}{\eta}) \exp(-\frac{\eta - \gamma}{\eta})$$
$$\exp(\frac{\eta - \gamma}{\eta}) = \int \exp(\frac{Q_{\theta_i}(a, s)}{\eta}) da$$

We can thus deduce:

$$q_i(a|s) = \frac{\exp(\frac{Q_{\theta_i}(s,a)}{\eta})}{\int \exp(\frac{Q_{\theta_i}(a,s)}{\eta})da} = softmax(\frac{Q_{\theta_i}(s,a)}{\eta})$$

As a result, we can calculate $q_i(a|s)$ easily by $Q_{\theta_i}(s,a)$, η and softmax function

3.2.4 M step

In the M-step of policy improvement, the target is to optimize policy network by Eq.10. In this paper, the authors mentions they empirically found out that better results could be achieve by decoupling the KL constraint into two terms i.e. constrain the contribution of the mean and covariance separately. In specific,

$$\int \mu_q(s)\mathrm{KL}(\pi(a|s,\theta_i), \pi(a|s,\theta)) = C_\mu + C_\Sigma, \tag{12}$$

where

$$C_\mu = \int \mu_q(s)\frac{1}{2}(tr(\Sigma^{-1}\Sigma_i) - n + ln(\frac{\Sigma}{\Sigma_i}))ds,$$
$$C_\Sigma = \int \mu_q(s)\frac{1}{2}(\mu - \mu_i)^T\Sigma^{-1}(\mu - \mu_i)ds.$$

Therefore, the generalized objective of M-step is

$$\max_\theta \min_{\eta_\mu>0,\eta_\Sigma>0} L(\theta, \eta_\mu, \eta_\Sigma), \tag{13}$$

where

$$L(\theta, \eta_\mu, \eta_\Sigma) = \int \mu_q(s) \int q(a|s)log\pi(a|s,\theta)dads + \eta_\mu(\epsilon_\mu - C_\mu) + \eta_\Sigma(\epsilon_\Sigma - C_\Sigma),$$

$\eta_m u$ and η_Σ are Largrangian multipliers; $\epsilon_m u$ and ϵ_Σ are the threshold of C_μ and C_Σ, respectively. Since there are max and min in Eq. 13, we iteratively update θ, Largrangian multipliers. In specific, we fixed θ first and solve $\min_{\eta_\mu>0,\eta_\Sigma>0} L(\theta, \eta_\mu, \eta_\Sigma)$ to get Largrangian multipliers η_μ^* and η_Σ^*, and then solve $\max_\theta L(\theta, \eta_\mu, \eta_\Sigma)$ with substituting η_μ^* and η_Σ^* into $L(\theta, \eta_\mu, \eta_\Sigma)$. Iteratively solve these two steps until θ, until θ and Largrangian multipliers converge.

In our implementation, we iteratively solve Eq. 13 five times. In both steps, we use gradient descent to update. Note that, since Largrangian multipliers should be larger than zero, we clip the Largrangian multipliers to zeros if the updated is smaller than zero. To stable the training process of actor network, we use 'clip_grad_norm_' to clip the gradient which larger than 0.1 to 0.1.

4 Empirical Evaluation

In the original MPO paper section 5.1, the author evaluate on Control Suite. Therefore, we train several continuous tasks on Control Suite to show the performance of MPO with non-parametric variational case. Here we separate to two subsections to discuss different points of view.

4.1 Evaluation on Cartpole, Hopper, and Acrobot

The suite of continuous tasks that we are evaluating against contains 4 tasks, comprising with 3 domains. To acquire the ability of our implementation, we test on Carpole, Hopper and Acrobot. The simplest one to learn is Cartpole, and the middle level is Hopper. Finally, Acrobot is the most difficult task to cope with. The visualization of the tasks is revealed in Figure 2. Here the model parameters are the same as Table 1 when performing on all tasks.

We start by looking at the results for the simplest Cartpole domain with the balance task. From Figure 3, we could see the mean return is converged quickly around 200 iterations. The curve of the mean return is quite similar to the plot in the original MPO paper. For the second task, we perform on both hopper standing and hopping task. The mean return of both tasks is shown in Figure 4. The x-axis is the iteration, and we could see that both returns are rising. However, the curve is not as smooth

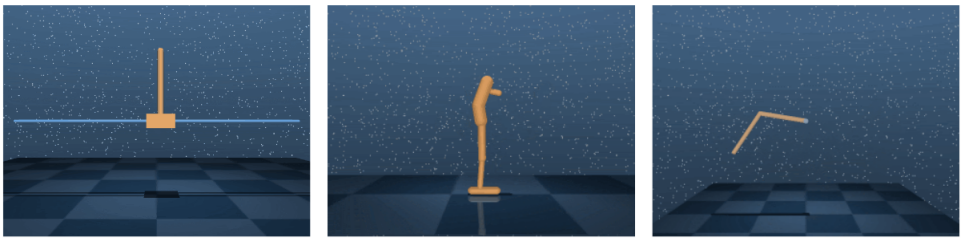

Figure 2: Control Suite domains that we have tried. Left to right: Cartpole, Hopper, Acrobot

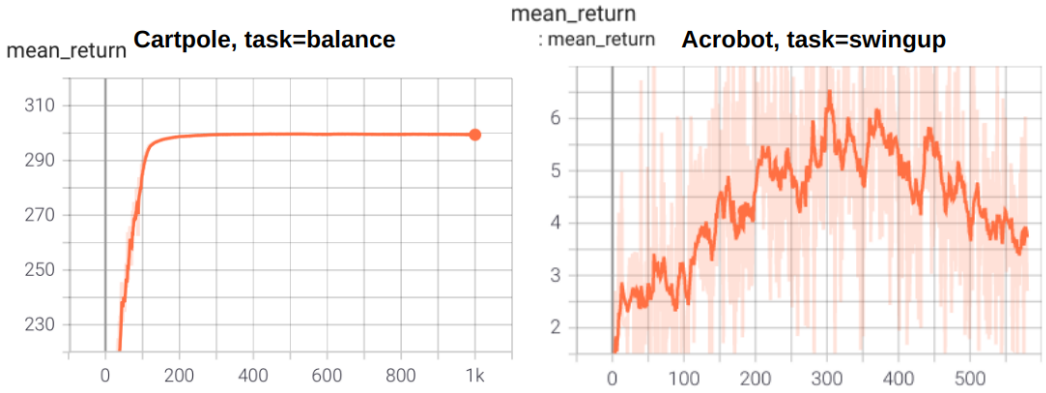

Figure 3: Mean return of Cartpole and Acrobot

as the one applied on Cartpole. In addition, it is obvious that standing task is easier to train than the hopping task. Since training is extremely time-consuming, we only ensure that the mean return is increasing so that we train for 1000 iterations only. Although the curves in our implementation and the ones in the original paper are not alike, it is guaranteed that our implementation is correct without any optimization or special tricks. The last task is Acrobot (two degrees of freedom) swing up task. The reward in the Acrobot task is the distance of the robots end-effector to an upright position of the underactuated system. It has one continuous action dimension and six observation space dimension. Desipte being low-dimensional, it is actually not an easy control problem. From the right plot of Figure 3, it is revealed that the mean return once increased slightly, but the trend seem to be decreasing after around 400 iterations. The mean return in Acrobot swing up task converged slower and achieved low return than other domain task. In comparison to the original paper, the difference of the results between our implementation and the paper is not pretty much, since we have mentioned that the Acrobot task is not easy to train.

4.2 Apply numerical tricks on Hopper task

When implementing the method, we found that the dual constraint η in the dual function, Equation 8, would congregate at a very small range of floating points around zero. The reason is that the $\exp(\cdot)$ operation in the dual function will have prominent difference when the Q-value inside the exponential operation is possible to be in both positive and negative side. This lead to numerical unstable. Consequently, we use a common trick similar to LogSumExp (LSE). We take out the maximal Q-value of $\frac{Q_{\theta_i}(s,a)}{\eta}$, and denote it as Q_{max}. Currently, the dual function would be:

$$g(\eta) = \eta\epsilon + \eta \int Q_{max} dads + \log \int \pi(a|s,\theta) \exp(\frac{Q_{\theta_i}(s,a)}{\eta} - Q_{max}) dads. \tag{14}$$

Equation 14 would limit the exponential operation results fixed in the negative side of $\exp(\cdot)$, and it would be easier to see the difference during the computation. We could see that the changes of η is prominent in the bottom row of Figure 5. Besides, the mean return on the same task reveals that the curve is much stable than the top-right one of Figure 5. For instance, at around 300 iterations in the

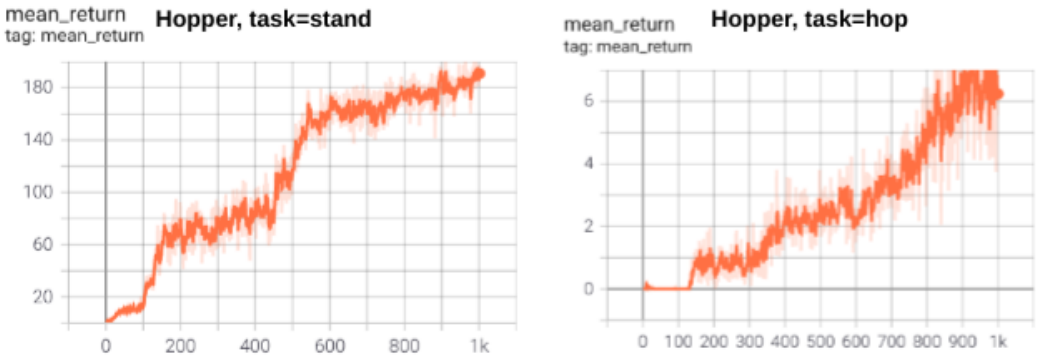

Figure 4: Mean return of Hopper with stand and hop task

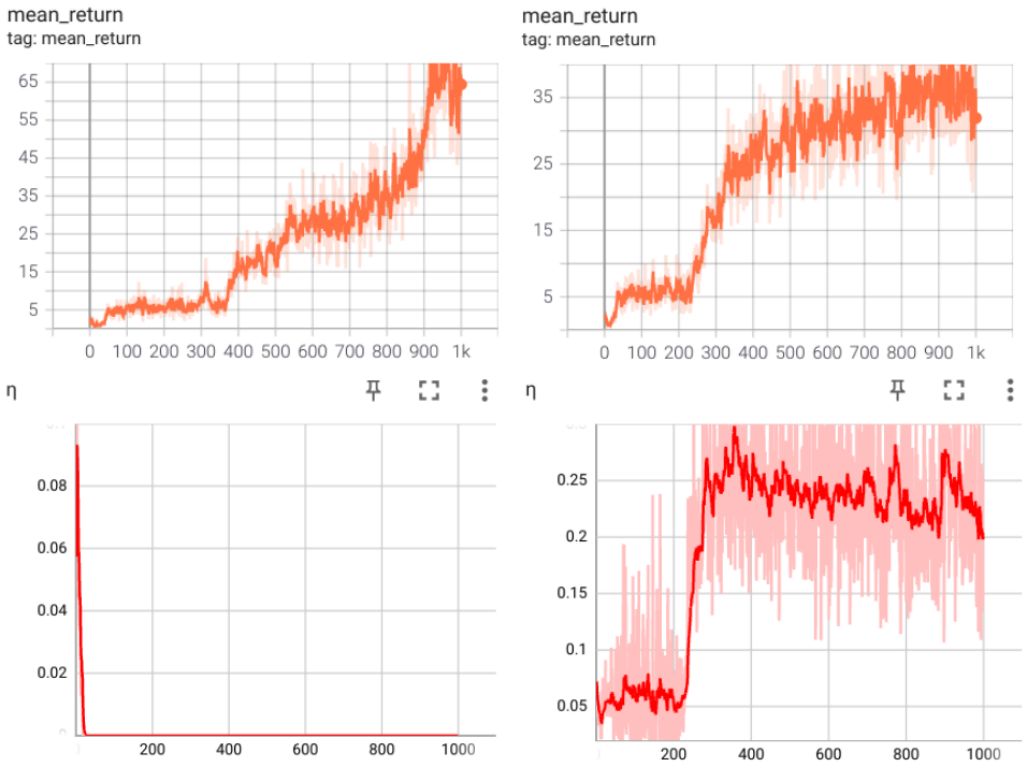

Figure 5: Mean return of Hopper with stable and unstable dual function. The left column is the original dual function, and the right one is the stable version. The first row is the mean return on Hopper with L1 loss, and the second row is the corresponding plot of η.

left-top sub-figure of Figure 5, the mean return has a sharp change. However, the mean return with stable dual function version, the return would not be extremely high in the near epochs.

5 Conclusion

The author provided a new RL training strategy via training model and auxiliary distribution. The main contribution is the idea transformation from inference to general RL task. The developed method is off-policy, so the training process is efficient by replaying the experience from buffer instead of interacting with the environment again and again, which is time consuming.

Furthermore, the author offered their theory with detailed explanation, helping the audience understand the theory behind quickly.Nevertheless, there is a typo in the section D of the appendix:

$$\exp(-\frac{\eta-\gamma}{\eta}) = \int \pi(a|s,\theta_i)\exp(\frac{Q_{\theta_i}(a,s)}{\eta})da$$

should be correct as

$$\exp(\frac{\eta-\gamma}{\eta}) = \int \pi(a|s,\theta_i)\exp(\frac{Q_{\theta_i}(a,s)}{\eta})da$$

Apart from replicating the algorithm from the paper, we also apply numerical tricks to stabilize the training process. Moreover, We are considering improving the method by modifying the E-step. In E-step, the author first set $q = \pi_{\theta_i}$ when constructing the Q-value function, leading to the optimization is one step. (Because we want to optimize q in E-step, but Q-value function is forced to follow π_{θ_i}, not the optimized q) This modification might be hard to implement due to the dramatically increasing complexity. However, if the above problem is overcome, the algorithm might benefit from faster convergence rate, which alleviate the slow converging issue in off-policy algorithm.

References

T.K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996. doi: 10.1109/79.543975.

John Schulman, F. Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/schulman15.html`.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Martin Seilair. Deepmind control suite to gym. `https://github.com/martinseilair/dm_control2gym`, 2018.