Replication: Meta-Gradient Reinforcement Learning

Chia-Hsiang Kao, Yu-Lin Tsai, Cheng-Hsun Chang
National Yang Ming Chiao Tung University

1 Problem Overview

The goal of reinforcement learning (RL) is to estimate or optimise the value function and to choose the proper action given a circumstance. However, unlike supervised learning, no teacher nor oracle is available to provide the true value function. Instead, the majority of RL algorithms estimate or optimise a proxy for the value function. This proxy is typically based on a sampled and bootstrapped approximation to the true value function, known as a return. The particular choice of return is one of the chief components determining the nature of the algorithm: the rate at which future rewards are discounted; when and how values should be bootstrapped; or even the nature of the rewards themselves. It is well-known that these decisions are crucial to the overall success of RL algorithms. The paper Xu et al. [2018] discusses a gradient-based meta-learning algorithm, where the return is adapted online, whilst interacting and learning from the environment.

In this research, the authors aim to find out the best form of return for the agent to maximise. They propose to learn the return function by treating it as a parametric function with tunable meta-parameters η, for instance the *discount factor* γ, or the *bootstrapping parameter* λ. The meta-parameters η are adjusted online as the agents interact with the environment, allowing the return to fit to the specific problem, and also to be dynamically adapted over time to the changing context of learning. They've derived a practical gradient-based meta-learning algorithm and show that this can significantly improve performance on large-scale deep RL applications.

2 Background and The Algorithm

Basic Definition

First, we define the n-step return as given a parametric value function approximator $v_\theta(S)$ and experience $\tau_t = \{S_t, A_t, R_{t+1}, ...\}$ as

$$g_\eta(\tau_t) = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n v_\theta(S_{t+n}) \tag{1}$$

and the λ-return as

$$g_\eta(\tau_t) = R_{t+1} + \gamma(1-\lambda) v_\theta(S_{t+1}) - \gamma\lambda g_\eta(\tau_{t+1}) \tag{2}$$

where η stands for the set of hyper-parameter $\{\gamma, \lambda\}$.

Meta-Gradient Prediction

Suppose we have a meta-objective $\bar{J}(\tau', \theta', \bar{\eta})$ to evaluate the performance of η, where τ' is an independent sampled trajectories and θ' is the updated parameters. (To note that η affects the updated parameters θ' and that $\bar{\eta}$ is set to 0.99 or 1.0 manually thus $\bar{\eta}$ is not associated with η.) Then we have the update of η as

$$\frac{\partial \bar{J}(\tau', \theta', \bar{\eta})}{\partial \eta} = \frac{\partial \bar{J}(\tau', \theta', \bar{\eta})}{\partial \theta'} \frac{d\theta'}{d\eta} \tag{3}$$

where $\frac{d\theta'}{d\eta} = \frac{d(\theta+f(\tau,\theta,\eta))}{d\eta}$ for some $f(\tau,\theta,\eta)$ in the update of parameter θ Thus,

$$\frac{d\theta'}{d\eta} = \frac{d\theta}{d\eta} + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta} + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta}\frac{d\theta}{d\eta} \tag{4}$$

$$= (I + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta})\frac{d\theta}{d\eta} + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta} \tag{5}$$

Then we have,

$$\triangle\eta = -\alpha * \frac{\partial\bar{J}(\tau',\theta',\bar{\eta})}{\partial\theta'}\{(I + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta})\frac{d\theta}{d\eta} + \frac{\partial f(\tau,\theta,\eta)}{\partial\eta}\} \tag{6}$$

Suppose we use a TD(λ) algorithm with mean squared error, i.e.,

$$J(\tau,\theta,\eta) = (g_\eta(\tau) - v_\theta(S))^2$$

then doing partial derivation gives us

$$\frac{\partial J(\tau,\theta,\eta)}{\partial\theta} = -2(g_\eta(\tau) - v_\theta(S))\frac{\partial v_\theta(S)}{\partial\theta} \tag{7}$$

The key idea of the meta-gradient prediction algorithm is to adjust meta-parameters η in the direction that achieves the best predictive accuracy. This is measured by cross-validating the new parameters θ' and second episode τ' and update the gradient w.r.t. η.

Meta-Gradient Control

We consider A2C as the base reinforcement learning algorithm. Then the update of A2C can be written in the following form

$$-\frac{\partial J(\tau,\theta,\eta)}{\partial\theta} = (g_\eta(\tau) - v_\theta(S))\frac{\partial log\pi_\theta(A|S)}{\partial\theta} + b(g_\eta(\tau) - v_\theta(S))\frac{\partial v_\theta(S)}{\partial\theta} + c\frac{\partial H(\pi_\theta(\cdot|S))}{\partial\theta} \tag{8}$$

where the first term stands for control, the second term as in previous slides stands for prediction, and the third term acts as a regularization term on the entropy of policy $\pi_\theta(\cdot|S)$. Moreover, we see that

$$\frac{\partial f(\tau,\theta,\eta)}{\eta} \tag{9}$$

$$= \frac{\partial^2 J(\tau,\theta,\eta)}{\partial\eta\partial\theta} \tag{10}$$

$$= \alpha\frac{\partial g_\eta(\tau)}{\partial\eta}[\frac{\partial log\pi_\theta(A|S)}{\partial\theta} + b\frac{\partial v_\theta(S)}{\partial\theta}] \tag{11}$$

Suppose we consider the meta objective that has derivative identical to the control term in the previous slide, then

$$\frac{\partial\bar{J}(\tau',\theta',\bar{\eta})}{\partial\theta'} = (g_{\bar{\eta}}(\tau') - v_{\theta'}(S'))\frac{\partial log\pi_{\theta'}(A'|S')}{\partial\theta'} \tag{12}$$

with equation (10) and (11), once could calculate the increment/decrement of η; namely, $\triangle\eta$.

Therefore, the algorithm is summarized as the following. With one episode τ, we use the update function (8) in A2C to update the parameters θ and the gradient w.r.t. update in equation (10) is stored and accumulated. Secondly, the performance is cross-validated on a subsequent sample of episode τ' and parameters θ' with equation (11) calculated. Together, we form $\triangle\eta$ and update.

3 Detailed Implementation

We implement A2C as the base algorithm under CartPole-v1 environment and consider γ (i.e. the discounting factor) as the only hyperparameter to be meta-learned/meta-adapted.

```
Algorithm 1 A2C with meta-learned discounted factor
    Require: Initialized actor model $\theta$ and value model $\phi$
    Require: Initialize $\eta = \{\gamma\} = 0.99$
 1: for $iteration = 1, 2, \dots$ do
 2:     Sample n-step trajectory $\tau$.
 3:     Compute n-step Q-value $g_\eta(\tau)$ and n-step state-value $v_\phi(S)$
 4:     $\theta' := \theta + (g_\eta(\tau) - v_\phi(S))\frac{\partial log\pi_\theta(A|S)}{\partial\theta}$
 5:     $\phi' := \phi + (g_\eta(\tau) - v_\phi(S))\frac{\partial v_\phi(S)}{\partial\theta}$
 6:
 7:     Sample another n-step trajectory $\tau'$.
 8:     Compute n-step Q-value $g_\eta(\tau')$ and n-step state-value $v_{\phi'}(S')$
 9:     Compute $\frac{\partial\bar{J}(\tau',\{\theta',\phi'\},\bar{\eta})}{\partial\theta'} = (g_\eta(\tau') - v_{\phi'}(S'))\frac{\partial log\pi_{\theta'}(A'|S')}{\partial\theta'}$
10:     Update $\gamma$: $= \gamma + \alpha\frac{\partial f(\tau,\{\theta,\phi\},\eta)}{\eta} \cdot \frac{\partial\bar{J}(\tau',\{\theta',\phi'\},\bar{\eta})}{\partial\theta'}$
11:                  $= \gamma + \alpha\frac{\partial g_\eta(\tau')}{\partial\eta}(g_\eta(\tau') - v_{\phi'}(S'))\frac{\partial log\pi_\theta(A|S)}{\partial\theta}\frac{\partial log\pi_{\theta'}(A'|S')}{\partial\theta'}$
12:     Update $\theta := \theta'$
13:     Update $\phi := \phi'$
14: end for
```

Figure 1: Pseudocode for reproduction

The maximum reward of the CartPole-v1 environment is 500. We constrain the number of training steps to be 500 due to computation and time limits. For each run, the training is terminated as soon as the agent obtains rewards exceeding than 475 (475 is the reward threshold specified by the CartPole-v1 environment).

We use neural networks to implement the actor and the value network, both of which comprise three fully connected layers with a hidden size of 40 and with ReLU as activation function. Log-softmax operation is used to normalize the sum of the output of the actor network to one. The table below is the hyper-parameters used in our experiment.

Hyper-parameters used in the replication experiment	
Optimizer	Adam
Learning rate	0.01
Clip gradient norm	0.5
Sample size	30
Initial gamma	0.99
Scalar coefficient (alpha)	0.0001

Table 1: The hyper-parameters used in our experiments.

4 Empirical Evaluation

Main result

Now we present our main implementation results in Figure 2, where experiments with nine different random seeds are conducted and for each experiment we show the *Naive A2C algorithm* (i.e. the left subplot) and *A2C with meta-learned gamma* (i.e. the right subplot).

Experiments with random seeds of $0, 1, 4, 7, 9$ clearly show a faster training speed with meta-adapted gamma in that the agent quickly achieves total reward of 500 compared to the conventional A2C algorithm. On the other hand, experiments with the remaining random seeds (i.e., $2, 3, 5, 6$) show

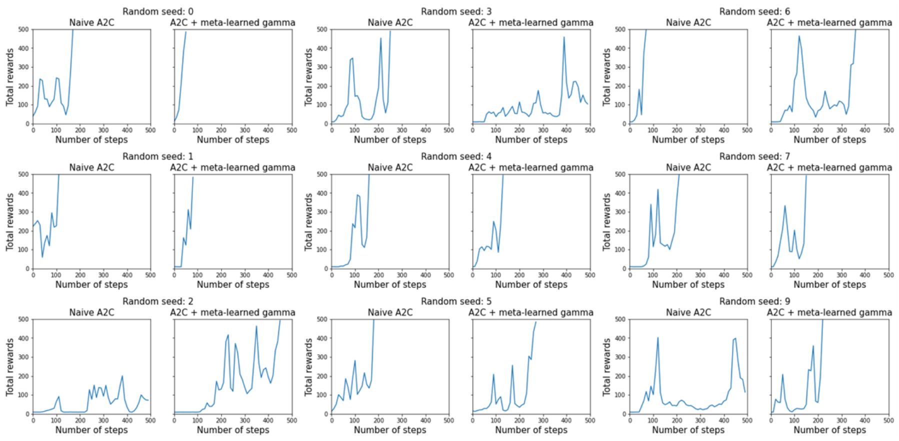

Figure 2: The main results obtained by training the agent using A2C algorithm and using the meta-adapted A2C algorithm.

$$\begin{aligned}\text{Update } \gamma\!: &= \gamma + \alpha \frac{\partial f(\tau, \{\theta, \phi\}, \eta)}{\eta} \cdot \frac{\partial \bar{J}(\tau', \{\theta', \phi'\}, \bar{\eta})}{\partial \theta'} \\ &= \gamma + \alpha \frac{\partial g_\eta(\tau')}{\partial \eta} \underbrace{(g_\eta(\tau') - v_{\phi'}(S'))}_{(1)} \underbrace{\frac{\partial log\pi_\theta(A|S)}{\partial \theta} \frac{\partial log\pi_{\theta'}(A'|S')}{\partial \theta'}}_{(2)}\end{aligned}$$

Figure 3: The decomposition of the update of hyperparameter γ.

slower training speed. This indicates that the meta-adapted A2C algorithm does not necessarily guarantee a better training profile. This is consistent with the original paper's empiric results since their experiments do not necessarily obtain improvement in all Atari games (some even degraded by -159%) (c.f. Figure 2 in Supplement [1]).

Discussion: decomposition of the update of γ

After implementation, we are unsatisfactory with the paper author's derivation and motivation as it seems to lack some insights and does not further interpret the formula they derive. Here, we try to interpret the meaning of this hyperparameter meta-adaptation. In Figure 3, we show that the update of γ contains two parts: (1) the discrepancy (error) between the target and the models' prediction; (2) the inner product of the policy gradients evaluated on two independent trajectories τ and τ'. The second term essentially estimates the *consistency* of the gradients and therefore entails the *confidence* of the update of the hyperpapermeter at the direction obtained by (1) term. Lower absolute values of (2) term indicate that the gradient is almost orthogonal, indicating the two estimated gradients are not consistent. Likewise, it is highly likely that the sampled error (i.e., (1) term) is untrustworthy.

We therefore decompose the update of gamma and visualize each component via explicitly plotting the (1) and (2) term. For (2), we normalize the gradients (i.e., compute the cosine similarity between gradients) so that we are less misled by the scale of gradients. In Figure 4, we see that at around 100^{th} epoch, there is a visible decrease in the value of gamma. Where does this decrease come from? At around 100^{th} epoch, we see that v_ϕ is larger than $g_\eta(\tau')$, therefore the γ to decrease so that the predicted value match the target value. (Ironically, this is similar to "when too many students do not pass the exam, we only have to decrease the minimum passing score.") Simultaneously, the cosine similarity at around 100^{th} epoch is relatively high (around $0.2 \sim 0.4$). Taking the two components together, we see the drop in the value of the hyperparameter γ.

Discussion: the initial value of gamma

Finally, now that the hyperparameters can update themselves, we wonder if it is workable for any initial value of γ. In Figure 6 and Figure 7 we set the initial discounting factor $gamma$ to be

[1]Please refer to the link: `https://arxiv.org/pdf/1805.09801v1.pdf`

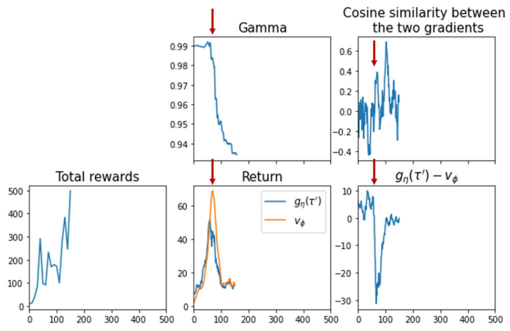

Figure 4: The decomposition of the update of hyperparameter γ: example 1.

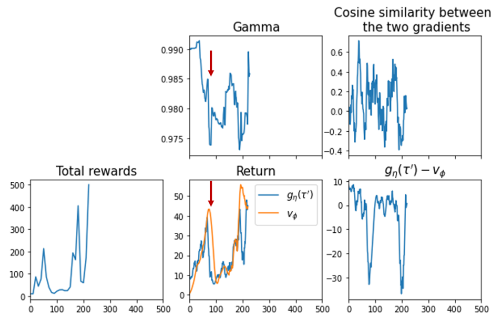

Figure 5: The decomposition of the update of hyperparameter γ: example 2.

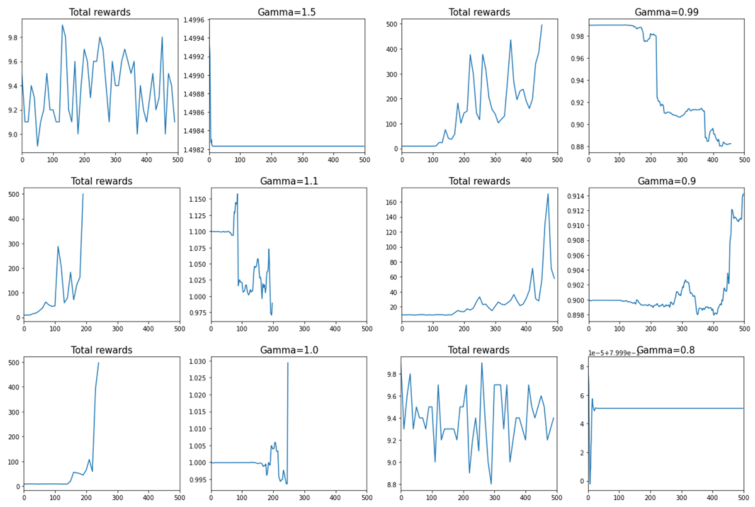

Figure 6: Different value of γ results in different training profile. Note that too larger (e.g.$\gamma = 1.5$) or too small (e.g.$\gamma = 0.9$) the discounting factor lead to instability in training.

$1.5, 1.1, 1.0, 0.99, 0.9$ and 0.8. The experiments show that the network does not converge in the case of $\gamma = 1.5, 0.9$ or 0.8. At the same time, we witness the drop in the value of γ when the discounting factor is initialized to be 0.99, in which case γ quickly drops to values below 0.95. It seems to break the stereotype that γ has to be properly set so that the value function can learn a true expected return (so that the actor can make decision correctly). This experiment shows that during training period, the hyperparameter γ has two roles: to represent the discounting factor and to work as a free variable to work as a buffer so that it can facilitate the training of network.

5 Conclusion

Our reproduction of meta-gradient reinforcement learning algorithm shows that, using A2C as based algorithm and considering γ (i.e. the discounting factor) as the only hyperparameter to be meta-learned/meta-adapted, we find that it often leads to faster convergence.

References

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

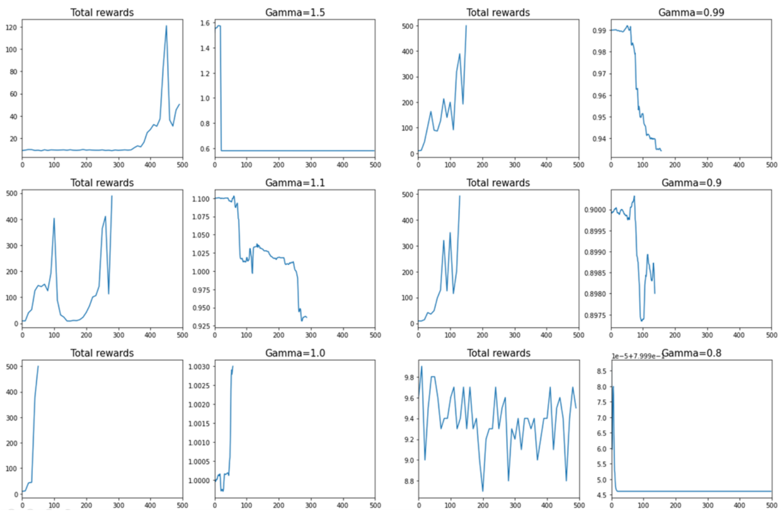

Figure 7: Different value of γ results in different training profile. Note that too larger (e.g.$\gamma = 1.5$) or too small (e.g.$\gamma = 0.9$) the discounting factor lead to instability in training.