Winter 2022: COMP 579 Reinforcement Learning GROUP 007: Final Report

Pingsheng Li
McGill University
260906286

Ye Yuan
McGill University
260921269

Xichong Ling
McGill University
260888765

Guangyi Zhang
McGill University
260708622

Abstract

In this report, we attempt to combine two well-known existing works in deep reinforcement learning in a new context. One of them aims to combine model-based and model-free approaches in order to avoid the disadvantages of only using one kind of them, and the other one performs a case study comparing two different model-free policy gradient algorithms, which aims to gain a deeper understanding on the difficulty and importance of attributing performance gains in deep reinforcement learning. Our results suggest that (i) the effectiveness of some approaches in deep reinforcement learning might be vastly impacted when the question is slightly different, and (ii) simply adding random optimization techniques may not always yield better performance; one needs to carefully select the optimal set of techniques in order to achieve more desirable results.

1 Introduction

Given the rapidly rising numbers of researches in Deep reinforcement learning, it becomes increasingly pivotal to effectively combine multiple relevant researches together to tackle with diverse problems that one may encounter in real life. Motivated by this, we attempt to combine two existing and relevant works in deep reinforcement learning of Nagabandi et al. [2017] and Engstrom et al. [2020]. The first work tries to combine model-based methods with model-free approaches because each method alone has its own advantages and pitfalls. Although model-free deep reinforcement learning algorithms can learn very complicated locomotion skills Schulman et al. [2015], such methods often suffer from exceedingly low sampling efficiency, requiring millions of samples or more in order to achieve ideal performance. In contrast, model-based methods are generally considered better in terms of sampling efficiency Chatzilygeroudis et al. [2018], yet it often suffers from over-fitting due to learning with few samples, making them difficult to be applied in more complex, high-dimensional tasks. But Nagabandi et al. [2017] finds a way to combine both methods, by letting a model-free agent imitate a agent trained by model-based approach, given that the reward function is known, to obtain a good starting point, then fine-tuning the agent by a model-free algorithm: Trust Region Policy Optimization (TRPO), which successfully improves both sampling efficiency and final performance. The other work conducts a comparative analysis on two model-free policy gradient algorithms: Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) Engstrom et al. [2020], which shows that a significant amount of performance improvement for PPO actually comes from seemingly small modifications that are originally considered auxiliary tricks. We try to combine these two works in a new context where the reward function is considered unknown, and we replace the final stage fine-tuning algorithm in the first work, i.e. TRPO, with PPO. Then we select a few tricks discussed in the second work to see if they could provide any improvement.

Preprint. Under review.

2 Framework

2.1 Model-based Approach

Learning Dynamics via Neural Network First, in order to make the agent capable of performing planning more effectively, we utilize a deep neural network, $\hat{f}_\theta(s_t, a_t)$, to approximate the dynamics of the environment, where the parameter vector θ represents the weights of the network. A valid choice for $\hat{f}_\theta(s_t, a_t)$ would take the input as a concatenated vector of the current state s_t and action a_t, and generate the predicted next state $\hat{s}_{t+1}$. However, a better choice is to parameterize the states differences $\Delta s = s_t - s_{t+1}$ instead of the next state via the neural network $\hat{f}_\theta(s_t, a_t)$ since predicting the next state is very difficult to learn when the states differences $\Delta s = s_t - s_{t+1}$ are very small.

Then the training data is collected via executing random actions at every timestep and recording the corresponding experience $\tau = (s_0, a_0, r_0, \ldots, a_{T-2}, s_{T-1})$ up to T timesteps. Then we transform the collected data into state-action pairs as inputs (s_t, a_t) and corresponding outputs Δs with some Gaussian noise, and store them in dataset D. After that, the neural network $\hat{f}_\theta(s_t, a_t)$ is trained via stochastic gradient descent (SGD) using the mean square loss function: $L(\theta) = \frac{1}{|D|} \sum_{(s_t, a_t, s_{t+1} \in D)} \frac{1}{2} ||\Delta s - \hat{f}_\theta(s_t, a_t)||^2$

Planning and Model-based Control In order to utilize the learned model $\hat{f}_\theta(s_t, a_t)$ for planning, we could first try to optimize the sequence of actions $A_t^{(H)} = (a_t, \ldots, a_{t+H-1})$ over a finite horizon H, using the prediction made by the learned dynamics model $\hat{f}_\theta(s_t, a_t)$, via maximizing the sum of the undiscounted future rewards within horizon H: $A_t^{(H)} = argmax_{A_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{s}_{t'}, a_{t'})$ where $\hat{s}_{t'+1} = \hat{s}_{t'} + \hat{f}_\theta(s_{t'}, a_{t'})$.

However, there are two major problems with this objective: first, the original paper Nagabandi et al. [2017] assumes the reward function $r(\hat{s}_t, a_t)$ is known in advance, whereas **in our context the reward function is assumed to be unknown**; second, computing the exact optimal solution is intractable. For the first problem, we decide to use a neural network as reward function approximator $\hat{r}_\theta(s_t, a_t)$ instead, which is trained from the previously collected data D. For the second problem, we choose to use sampling-based method to estimate the optimal solution via random sampling K sequences of actions and select the optimal actions sequence that gives the best total future rewards. But, the agent will only execute the first action a_t of the chosen sequence, transition to the updated state s_{t+1}, and then recalculate the optimal action sequence at the next time step in order to reduce the error introduced by the dynamics function.

2.2 Model-free Approach

To further refine the actor network obtained via imitation, we apply Proximal Policy Optimization (PPO) instead of Trust Region Policy Optimization (TRPO) that is used in the original paper Nagabandi et al. [2017]. Both policy gradient algorithms are valid choices for model-free fine-tuning because they do not require any critic or value function for initialization. But PPO was originally developed as a refinement of TRPO Schulman et al. [2017]. Therefore, we expect some improvements on performance after this modification.

Initializing the Model-Free Learner via Imitation First we collect some trajectories with the model-based control mentioned before using the learned dynamics function $\hat{f}_\theta$, and only keep those that actually achieve relatively high rewards. Then these trajectories are stored into a dataset $D*$, and we then train a neural network policy (the actor network) $\pi_\phi(a|s)$ to imitate the actions a_t given states s_t of these "expert" trajectories. Here, π_ϕ is parameterized as a conditionally Gaussian policy $\pi_\phi(a|s) \sim N(\mu_{\phi(s)}, \Sigma\pi_\phi)$, in which the mean is the output of the actor network $\mu_{\phi(s)}$, and the covariance $\Sigma\pi_\phi$ is a fixed diagonal matrix. This network is trained by minimizing a mean square loss function: $L(\theta) = \frac{1}{2} \sum_{(s_t, a_t) \in D*} ||a_t - \mu_{\phi(s_t)}||^2$ via stochastic gradient descent. After this training process is completed, we now have an actor network initialized via imitating a model-based agent, and it's ready for the model-free fine-tuning later.

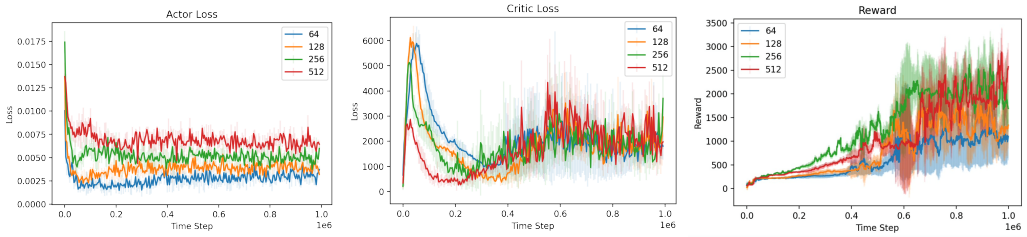
Model-Free Fine Tuning Now we are able to fine-tune the actor network with PPO, along with a newly initialized critic network for value estimation. The pseudo-code for our implementation of PPO is given in the appendix.

Implementation Matters in Proximal Policy Optimization We also change a few parameters (e.g. the width of the neural networks) and select some tricks discussed in Engstrom et al. [2020] (e.g. activation functions and reward scaling) to see if there could be any improvement and if the results are consistent.

3 Experiments

We choose a benchmark robotic RL environment: Hopper-v2, for experiments. We first collect data of 5,000 timesteps with random exploration policy, then train the dynamics neural network $\hat{f}_\theta(s_t, a_t)$ and the reward function neural network $\hat{r}_\theta(s_t, a_t)$ using this data. Then we keep collecting data with planning and model-based control discussed in section 2.1, until the number of episodes with 300 or more rewards reaches 100, and we discard all other relatively bad data and only keep these 100 episodes for the imitation learning of the actor network discussed in section 2.2. Finally, the trained actor network and a newly initialized critic network are fine-tuned by PPO algorithm. All neural networks used in this experiment have 3 feed-forward layers with ReLU activation function. Each layer contains 512 hidden units. The networks are updated with Adam optimizer for 10 epochs every time when data points of 4,000 timesteps are collected using a fixed learning rate $3e-4$. The discount factor γ is 0.99, and the total number of timesteps for training is 2 million. We also try changing network size, changing activation functions, and adding reward scaling (see pseudo-code in the appendix) to see if the results are consistent and if there is any improvement.

4 Results and Discussion



(a) Actor objective (b) Critic loss (c) Accumulated rewards

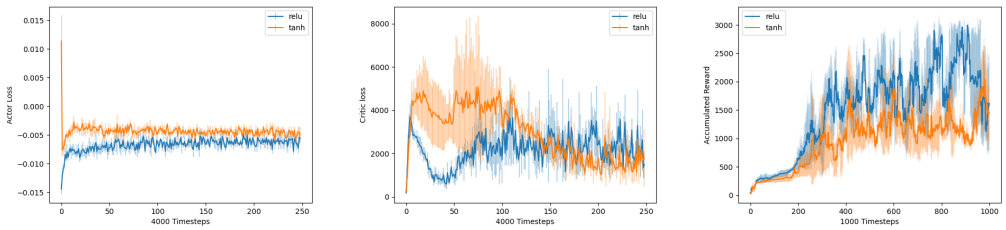
Figure 1: Learning curves with different numbers of hidden units in neural networks.

The first result is negative: the model-based approach discussed in section 2.1 is not very reliable. Although it's verified that the dynamics neural network $\hat{f}_\theta(s_t, a_t)$ and the reward function approximator neural network $\hat{r}_\theta(s_t, a_t)$ are trained successfully via inspecting the training and validation losses, when combined together to perform model-based control, they fail to cooperate well and the action sequences generated via planning are highly unstable. As a consequence, the time for collecting 100 episodes of good trajectories is also highly variable and might be very long, resulting in poor sampling efficiency. The imitation learning is considered successful since the actor network can achieve similar performance after trained on the collected relatively good episodes, but the entire procedure is less effective than simply doing model-free training at the beginning. Therefore, we did not include this part in the final submission.

This negative result is actually not surprising because there's an important difference between the context in Nagabandi et al. [2017] and ours: whether we know the reward function in advance or not. Since the reward approximator is only trained via data from random exploration at the beginning, it's very likely for powerful function approximators like neural networks to overfit. Also, the true reward function of Hopper-v2 is mainly about the velocity of moving, so the spatial translation symmetry of the rewards, i.e. the optimal action sequence for locomotion is independent of the exact position, cannot be effectively learned via this approach. This suggests the method in Nagabandi et al. [2017] may heavily depend on if we know the true reward function.

For the second part, each agent is only trained for 1 million timesteps due to time constraint, and the results are averaged across 3 runs. In Fig. 1, we can see that wider neural networks can achieve higher values of the actor objective (which is meant to be maximized), and can in general achieve better results (more rewards). Also, it's interesting to see the trends in their critic loss curves: all

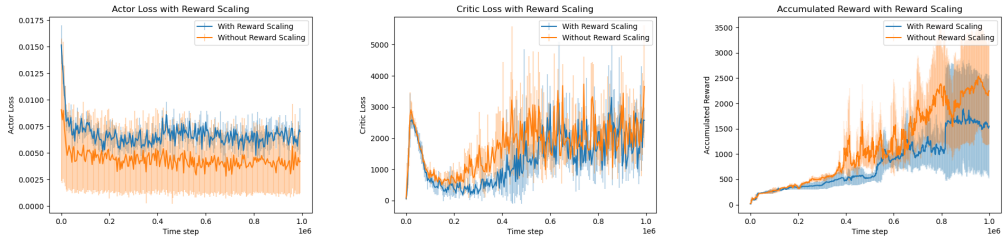
agents experienced a period of having high critic loss because they are new to the environment, so their initial value estimates are inaccurate. But wider neural networks have lower maximum critic loss in the initial stage of training because their learning dynamics are proved to be considerably simpler Lee et al. [2020] which may enable them to learn to estimate values faster and better. The non-convergence of critic loss in later stage is expected since the agent keeps exploring and constantly reevaluating the values of states.



(a) Actor loss (b) Critic loss (c) Accumulated rewards

Figure 2: Learning curves of agents using neural networks with different activation functions.

In Fig. 2, we can see that ReLU outperforms tanh in this experiment since it can obtain more rewards in general. Also, actor loss (actor objective multiplied by -1) and initial stage critic loss are both lower for ReLU, which maybe is relevant to the fact that neural networks with ReLU are recognized easier to train Nair and Hinton [2010]. However, as observed by Henderson et al. [2017], tanh is considered a more standard implementation in policy gradient algorithms. This result further confirms that even small details in the implementation of deep RL methods can have non-trial impacts on the performance Engstrom et al. [2020].



(a) Actor objective (b) Critic loss (c) Accumulated rewards

Figure 3: Learning curves with and without reward scaling technique.

Fig. 3 illustrates that reward scaling is hugely helpful in reducing the variance of and achieving higher values of actor objective, but its impact on critic loss is unclear. However, despite its stabilizing effects on the actor network's learning, reward scaling isn't necessarily improving the performance since the average accumulated rewards are even lower with it. This result is actually inconsistent with previous findings Engstrom et al. [2020], but their setting is different with ours since they examined four optimization tricks concurrently whereas we only pick one trick, i.e. reward scaling, which may explain the discrepancy. This result offers support that one needs to carefully select the set of optimization techniques in deep RL algorithms, and simply choosing random tricks may not always improve the results since improvements may only be achieved when some techniques are utilized in conjunction with others. Finally, after reducing the variance of conditional Gaussian from which the agent samples actions, our approach achieves 3301.54 ± 1.24 accumulated rewards and 155.94 ± 3.53 sampling efficiency score (3 latest results on the leaderboard by April 27^{th}).

5 Conclusion

In conclusion, the effectiveness of certain methods in deep RL may be significantly impacted when the question is slightly different, e.g. whether the reward function is known in advance. Also, certain optimization techniques in deep RL may not always improve the performance when they are applied alone. Sometimes it's the net effect of multiple optimization techniques that matters for performance improvements.

References

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials, 2018. URL https://arxiv.org/abs/1807.02303.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *CoRR*, abs/2005.12729, 2020. URL https://arxiv.org/abs/2005.12729.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017. URL http://arxiv.org/abs/1709.06560.

Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(12):124002, dec 2020. doi: 10.1088/1742-5468/abc62b. URL https://doi.org/10.1088%2F1742-5468%2Fabc62b.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017. URL http://arxiv.org/abs/1708.02596.

Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015. URL https://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

A Appendix

Algorithm 1 PPO Clip

1: Input: initial actor network parameters θ_0 and initial critic network parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set if trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ based on the current critic network V_{ϕ_k}
6: Update the actor network by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \text{argmax}_\theta \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \text{clip}(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(s,a))$$

via stochastic gradient ascent with Adam
7: Update critic network by minimizing mean-square error (MSE):

$$\phi_{k+1} = \text{argmin}_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$

via stochastic gradient descent with Adam
8: **end for**

Algorithm 2 PPO scaling optimization

procedure I(N)ITIALIZE-SCALING()
    $R_0 \leftarrow 0$
    R_S = RUNNING-STATISTICS() ▷ New running stats class that tracks mean, standard
end procedure
procedure S(C)ALE-OBSERVATION(r_t) ▷ Input: a reward r_t
    $R_t \leftarrow \gamma R_{t-1} + r_t$ ▷ γ is the reward discount
    ADD(R_S, R_t)
    return r_t/STANDARD-DEVIATION(R_S)
end procedure