Reinforcement Learning Assignment 3 Policy-Based

Wei Chen [1] **Yijie Lu** [1] **Jialiang Wang** [1]

1. Introduction

In this assignment, we did experiments on policy-based methods in reinforcement learning, which are *REINFORCE* and *Actor-critic*. In our experiments, firstly, in the part of *REINFORCE* algorithm, we choose different learning rate to make sure how it effects the results and try to explain it. And in the part of *Actor-critic* algorithm, we did ablation experiment which consists of actor-critic with bootstrapping, actor-critic with baseline subtraction and actor-critic with bootstrapping plus baseline subtraction.

The roadmap of our assignment report is as follows, in chapter 2 we give our implementation and experiment design. In chapter 3, we give our experiments results and analysis. In the final chapter, we give our conclusions.

2. Implementation and experiment design

To implement algorithms *reinforce* and *actor-critic*, we build a simple fully connected network based on *Tensorflow*, where the structure of the networks (or models) for both two algorithms are identical, equally consisting of four layers (the units of these four layers are 4, 32, 64, and 2, respectively).

2.1. Reinforce

Reinforce is a Monte Carlo Policy Gradient approach in which optimization should be conducted after every episode $(S_1, A_1, R_2, S_2, A2, ...R_T, S_T)$ is collected. The optimization to policy could be defined as follow:

$$\theta = \theta + \eta \nabla log \pi_\theta(S_t, a_t) V_t \tag{1}$$

where V_t, θ, η, and $\pi(S_t, a_t)$ represent the value of a state, the parameters of the policy model, learning rate, and the log probability of sampling an action a_t from distribution given a state S_t. In general, V_t represents the value of state V_t; nevertheless, we replace it with the two distinct calculations (described later) of the return value G_t in our practice.

In practice, an action is sampled to interact with the *Cartpole-v1* environment by bringing the current state into the policy model, which outputs the predicted probability of actions. We repeat this interaction described above infinitely until a game is over. Next, the episode collected is brought to compute the logarithm of the probability of the selected action and the value of state (can be derived from the calculation of the return value G_t); however, an episode could be used to compute the return value G_t corresponding to each state S either according to the equation (2) requiring full rewards, or according to the equation (3) based on 1-step target (bootstrapping):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + ... + \gamma^{T-t-1} R_T \tag{2}$$

$$G_t = R_{t+1} + \gamma V(S_{t+1}) \tag{3}$$

where the initial state is S_1 and γ represents the discount factor. In our implementation, both two computations work well similarly, but we decided to carry out our follow-up experiment with the latter one, i.e., 1-step target (equation (3)). Finally, once the log probability of selected (or sampled) action and the return value (or value of state) for each state in an episode are obtained, gradient ascent is applied to maximize the loss, that is, $\pi_\theta(S_t, a_t)V_t$. After an update, an episode for such update would be abandoned, and we then collect a new one for the next update.

2.2. Actor-Critic

Instead of computation of the return value G_t (or V_t) that requires complete rewards in an episode, *Actor-Critic* additionally introduce a value model, i.e., critic network to directly approximate the value of state V_t. Critic network likewise takes as input the current state; however, the prediction is the value of state V_t. For actor network here, it accepts the current state S_t and then predicts the distribution over actions.

In practice, we call the actor network to interplay with the *Cartpole-v1* environment and receive the next observation. Then, we need to conduct the training on critic network while computing and returning the advantage function $A(S_t, a_t)$ in which bootstrapping (1-step target) and baseline subtraction are introduced. Instead of start training after sampling a complete episode (or trace), critic model and actor model are updated after every performed action. The application of bootstrapping (see equation (4)) and baseline subtraction (see equation (5)) are defined below:

$$Q(S_t, a_t) = \begin{cases} R_T & \text{if terminal} \\ R_{t+1} + \gamma V(S_{t+1}) & \text{else} \end{cases} \tag{4}$$

$$A(S_t, a_t) = Q(S_t, a_t) - V(S_t) \tag{5}$$

Based on the two equations above, once the advantage function is computed with the help of critic model, we accordingly yield the optimization to policy (actor model) defined as:

$$\theta = \theta + \eta \nabla log \pi_\theta(S_t, a_t) A(S_t, a_t) \tag{6}$$

Additionally, to experiment on actor-critic without bootstrapping, the estimate of the $Q(s_t, a_t)$ is calculated based on the way of the *Monte Carlo* (see euqation 2), while to experiment on that without baseline subtraction, the actor is updated according to the equation below:

$$\theta = \theta + \eta \nabla log \pi_\theta(S_t, a_t) Q(S_t, a_t) \tag{7}$$

where we re-write a new program to calculate the $Q(S_t, a_t)$ when bootstrapping is not considered, since the calculation of $Q(S_t, a_t)$ based on *Monte Carlo*, requires a complete episode (or trace); therefore, actor model and critic model here are updated after a game is over.

2.3. Experiment design

The optimizers for the models in both two algorithms are all *Adam*, and we also apply the idea of early-stopping. The *Cartpole-v1* environment is where a pole is attached by an un-actuated joint to a cart, while the goal is to prevent the pole from falling over. To make it keep balance, the agent we developed should be supposed to continually apply a force (+1/-1) to the cart. In our implementation, if the pole falls over, the agent would receive a -1 bonus, while if the pole is successfully pushed 500 times, the agent would get a +10 bonus. In other scenarios (within 500 pushes), the bonus returned is +1.

There are primarily two experiments we made:

- Distinct learning rate (0.01, 0.001 and 0.0001) for both *reinforce* and *actor-critic* (only actor-critic with baseline subtraction combined with bootstrapping)
- Comparison between *actor-critic* without baseline subtraction combined with bootstrapping; actor-critic with baseline subtraction or with bootstrapping; and actor-critic with baseline subtraction combined with bootstrapping.

3. Experiments and results

3.1. REINFORCE

We consider the criterion for a good model (or a good result) is that the average number of steps of the last 100 episodes is more than 250. The performance of different learning rates are presented in Figure 1, Figure 2 and Figure 3. We compare the model effects at different learning rates: 0.01, 0.001, and 0.0001.

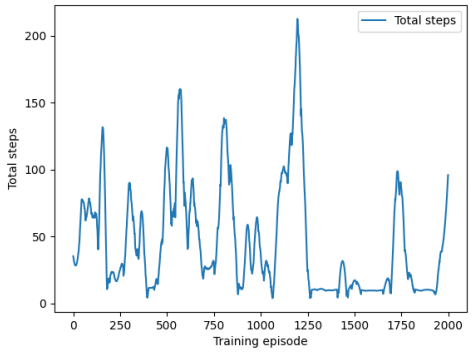

Figure 1. REINFORCE algorithm, the learning rate is 0.01 and GAMMA is 0.95

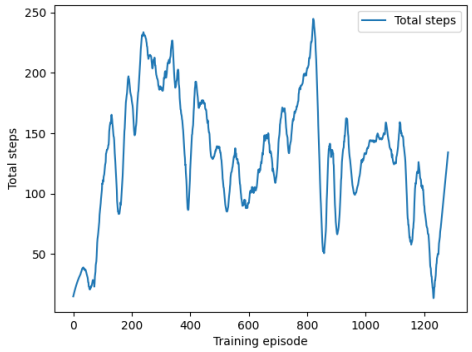

Figure 2. REINFORCE algorithm, the learning rate is 0.001 and GAMMA is 0.95

For *REINFORCE*, the model performs best when the learning rate is 0.0001 and the total steps grow rapidly to 400, When there is a significant regression, the intelligence is able to relearn the good strategy. An inappropriate learning rate makes an unstable training process. The larger the learning rate, the less steps per episode can reach. When the learning rate is 0.001, the peak value of total steps reaches around 250, however, the peak value only reaches 200, when the learning rate is 0.01. A similar situation occurs with the *actor-critic* model, where the model performs best at a learning rate of 0.0001.

3.2. Actor-critic

3.2.1. LEARNING RATE

As in the model above, in the actor-critic algorithm, we tested the effect of three different learning rates using both baseline subtraction and bootstrapping. The training effects of the three learning rates are shown in Figure4, Figure5, and Figure6. From the training process diagram, we can find that when the learning rate is too large (such as 0.01),

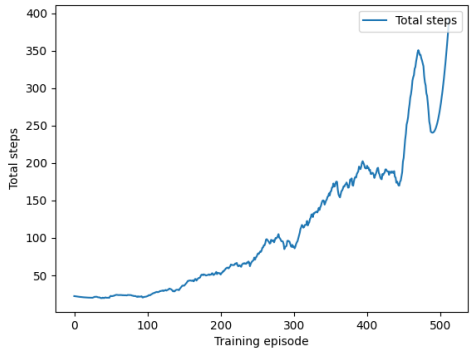

Figure 3. REINFORCE algorithm, the learning rate is 0.0001 and GAMMA is 0.95

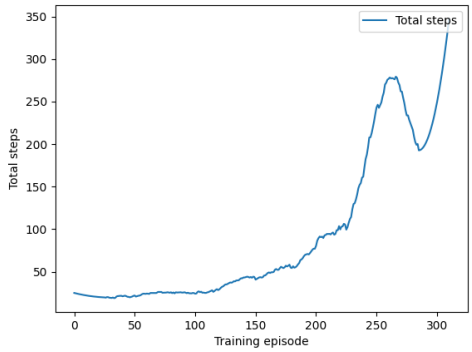

Figure 4. Actor-critic algorithm, the learning rate is 0.0001 and GAMMA is 0.95

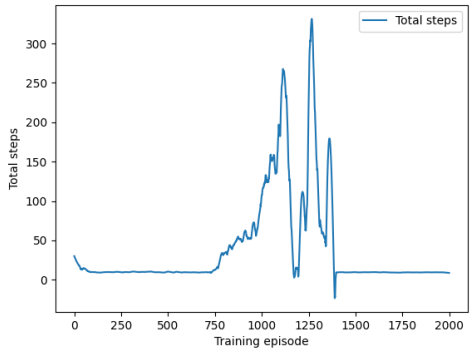

Figure 5. Actor-critic algorithm, the learning rate is 0.001 and GAMMA is 0.95

the model cannot get good results, but oscillates back and forth in a very low initial interval, and a good strategy cannot be obtained. As the learning rate is reduced (to 0.001), the model can get better results in some epsiodes but fluctuates a lot and eventually misses the optimal strategy resulting

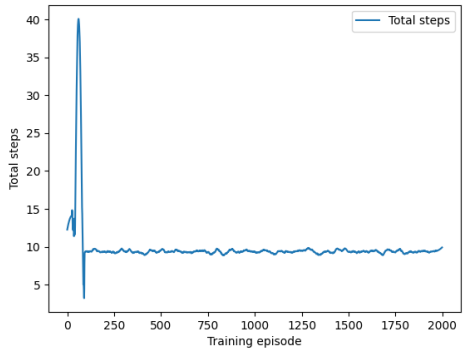

Figure 6. Actor-critic algorithm, the learning rate is 0.01 and GAMMA is 0.95

in poor final results. When the learning rate is adjusted to 0.0001, the model can stably improve the effect of the strategy, and the overall effect shows an upward trend. So in the actor-critic model, the learning rate of 0.0001 is still the best.

3.2.2. BOOTSTRAPPING + BASELINE SUBTRACTION

In order for the model to show better results, we use 0.0001 as the learning rate of the model (the learning rate of the actor network and critic network is the same). Since we used the bootstrapping+baseline subtraction model when doing the learning rate experiment, the training process is shown in the figure4.

Analyzing the training process, we found that when bootstrapping and baseline subtraction are used at the same time, the number of steps that the model can persist in each episode increases with the increase of training rounds, and the curve is relatively smooth. After enough episodes of training, the model can eventually achieve a good effect and stop training early. (Here we set the standard of good effect if 100 consecutive episodes in training can persist for more than 250 steps on average)

3.2.3. BOOTSTRAPPING(REMOVE BASELINE SUBTRACTION)

The training process of actor-critic with only bootstrapping is shown in the Figure7. Compared with the previous model for testing the learning rate, this model only uses bootstrapping and no baseline subtraction, so by comparing the two models with the same learning rate, we can analyze the effect of baseline subtraction. According to the comparison of the training process in Figure4 and Figure7, we can find that as the training progresses, the number of steps that the model persists in each episode gradually increases, and finally a good effect can be achieved. However, the

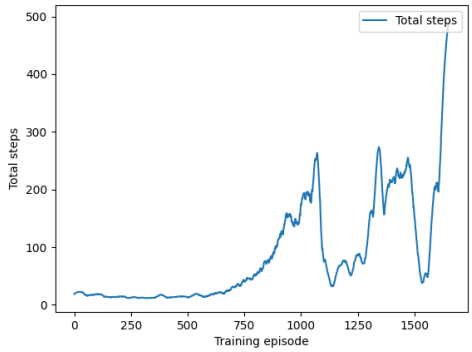

Figure 7. Actor-critic algorithm with bootstrapping(remove baseline subtraction)

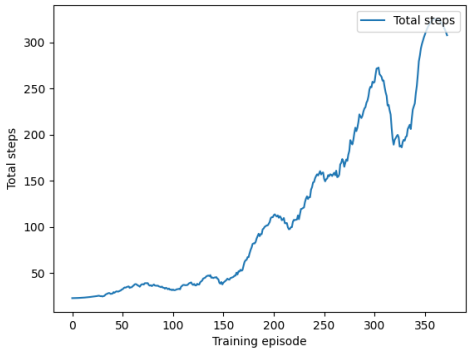

Figure 8. Actor-critic algorithm with baseline subtraction(remove bootstrapping)

training curve of the model without baseline subtraction is more volatile than the model with baseline subtraction. We analyze that this is because the use of the baseline method can reduce the variance of the gradient, thereby reducing the shock of the learning curve, allowing the model to converge to the best effect faster.

After the baseline subtraction is removed, the variance of the gradient will be very large, so the model will make many inefficient updates, resulting in slow convergence, requiring many episodes to achieve good results. This is also reflected in the training process diagram. In the training process using bootstrapping and baseline subtraction at the same time, the model achieves good results after about 500 episodes as shown in Figure4. However, in the training process of removing baseline subtraction as shown in Figure7, nearly 2000 episodes are needed to achieve good results.

3.2.4. BASELINE SUBTRACTION(REMOVE BOOTSTRAPPING)

The training process of this model is shown in Figure8. This model retains baseline subtraction but removes bootstrapping. Observing the training process diagram, we can find that the convergence speed of the model is still relatively fast, and only about 300 episodes are used to achieve a good effect. However, comparing the training process using bootstrapping, as shown in the Figure4, it can be found that the fluctuation of the learning curve between each episode is more obvious, and the trend fluctuation of the overall learning will be more obvious.

For the speed of convergence, we think this is because the model retains the baseline subtraction, so the variance of the gradient is not particularly large, so the model can have a relative fast convergence rate. As for the fluctuation showing in the training process, each episode model will sample a trace for training during training process, but due to the randomness of sampling, the traces sampled each time are different, so each update will also be very different. Bootstrapping is one of the ways to solve this large update variance problem, so when bootstrapping is removed, the fluctuation between each episode of the learning curve will become larger.

4. Conclusions

In conclusion, we implement two deep reinforcement learning models *REINFORCE* and *actor-critic*, then demonstrate their abilities to master difficult control policies for Cartpole, using only a four-layer network.

Through experiments we come to the following conclusions:

- For both *REINFORCE* and *actor-critic*: when learning rate is 0.0001, it performs better than 0.01 and 0.001. When the learning rate is too large, the optimal value will be skipped, which will lead to instability and poor performance. There are many actions in one episode, we have no idea which one is indeed helpful. *REINFORCE* algorithm has high variance and slow convergence.
- For bootstrapping within the actor-critic framework, it can trades-off bias for variance of update. Every episode we sample a trace for training, since the randomness of sampling, the update from each trace may differ between each other, and bootstrapping can help to solve this problem. So when we implement bootstrapping, the learning curve will be smoother, and the fluctuations between episodes will be relatively small.
- For baseline subtraction within the actor-critic framework, it can reduce the variance of the policy gradient. When the variance of the gradient is reduced, the network will try to perform the gradient in the correct direction when the gradient is ascending, so the conver-

gence speed of the network will be relatively faster. So when the baseline subtraction is used, the model can achieve good effect faster. At the same time, due to the reduction of the gradient variance, the fluctuation of the training curve will be relatively small.

- When we use baseline subtraction and bootstrapping at the same time, and use a suitable learning rate, the model can not only reduce the deviation caused by different sampling traces in different episodes, but also reduce the gradient, so as to converge faster, so use these two methods at the same time, the actor-critic algorithm can smoothly and quickly converge to a model with good effect.