Reinforcement Learning Assignment 2-Value Based

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

1. Introduction

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. 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:

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

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 bootstrapping), and actor-critic with baseline subtraction combined with bootstrapping.

3. Experiments and results

4. Conclusions