Reinforcement Learning

COMS4047A/COMS7053A

Lab 4 - Policy Gradients

Tamlin Love (tamlin.love1@students.wits.ac.za)
Shahil Mawjee (shahil.mawjee@students.wits.ac.za)

Overview

This lab will focus on Policy gradient methods.
Goals:

- Understand the policy-gradient approach to directly training a parameterised policy to maximise expected future rewards.
- Understand how the policy-gradient theorem allows us to improve the policy using on-policy.

Submission

Work in groups of 3 to 4 people. List group members in the *group_members.txt* file.

Once your have completed the lab, zip your code file and all plots from the exercises. Submit your zip file to Moodle.

1 REINFORCE: Monte Carlo Policy Gradient

In this section we will take a look at REINFORCE - a simple policy-based method. REINFORCE (and policy-based methods in general) directly optimise a parametrised policy in order to maximise future rewards.

We will try to learn a policy $\pi_\theta(a|s)$ which outputs a distribution over the possible actions a, given the current state s of the environment. The goal is to find a set of parameters θ to maximise the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] \tag{1}$$

where τ is a trajectory sampled from p_θ. The **policy-gradient theorem** gives us the derivative of this objective function:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right) \right] \tag{2}$$

- We have a policy $\pi_\theta(a|s)$ which tells the agent which action a to take, given the state s, and it is parameterised in terms of parameters θ.
- The goal is to maximise $J(\theta)$ by **choosing actions from this policy** that lead to high future rewards.
- We'll use gradient-based optimisation to update the policy parameters θ. We therefore want the gradient of our objective with respect to the policy parameters.
- We use the policy-gradient theorem to find an expression for the gradient. This is an expectation over trajectories from our policy and the environment.
- Since we can now sample trajectories $(s_0, a_0, R_0, s_1, a_1, R_1, \ldots)$ using our policy π_θ, we can approximate this gradient using **Monte-Carlo** methods.

This algorithm is called **Monte-Carlo REINFORCE**, and is one type of policy-gradient algorithm.

Let's use this to solve the **Cart-pole environment**!

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Figure 1: REINFORCE (Source: Sutton & Barto Section 13.3, page 328)

Exercise

This exercise must be completed within the `reinforce.py` file. We will create a REINFORCE agent to balance the cart-pole (`CartPole-v1`).

1. Complete the `compute_returns` method. This method is responsible for computing the return of a trajectory, given an array of rewards along with the discount factor γ.

2. Define a neural network within the `SimplePolicy` class. As the class name suggests, this neural network will be the agent's policy. The input will be the observation provide by the environment and the output will be an action to take.

3. Complete the `reinforce` algorithm.

4. Plot the learning curve (total return) for each episode on the y-axis. Add another line plot of the moving average with a windows size of 50. Refer to figure 2 as an example.

5. From figure 2, we can see that is a lot of variance. Average the returns over five runs and plot the mean and bounds of one standard deviation. Refer to figure 3 as an example

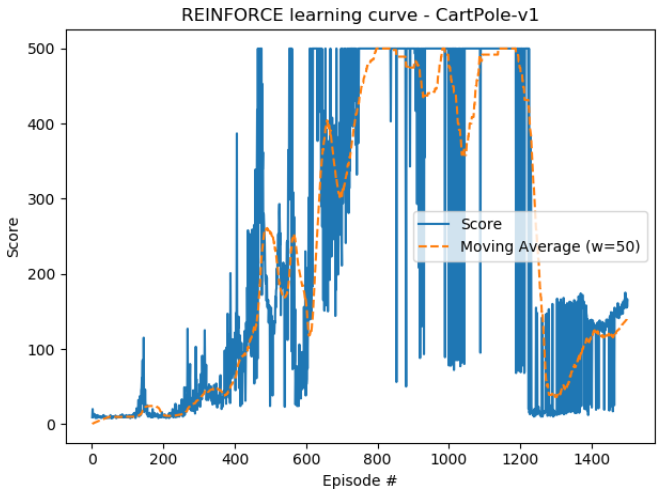

Figure 2: REINFORCE (single run)

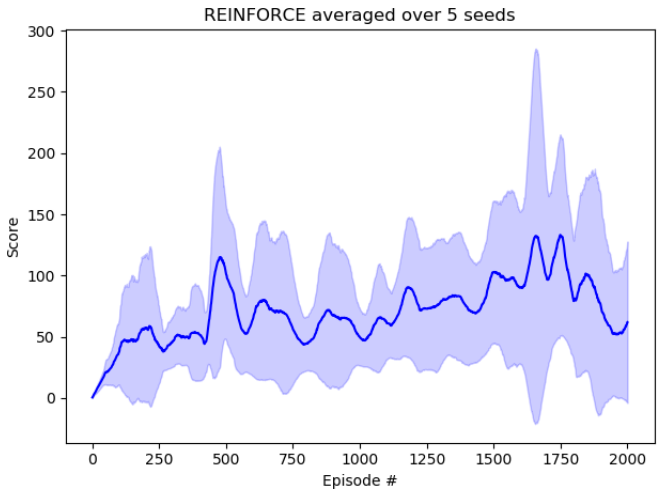

Figure 3: REINFORCE (averaged over 5 runs)

2 REINFORCE with baseline

From the previous exercise we can see that we need to reduce the variance of REINFORCE and improve its performance.

Firstly, future actions should not change past decisions. Present actions only impact the future. Therefore, we can change our objective function to reflect this:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right] \tag{3}$$

We can also reduce variance by subtracting a state dependent baseline to get:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} \left(\gamma^{t'-t} r(s_{t'}, a_{t'}) \right) - b(\tau) \right) \right] \tag{4}$$

For the baseline we will take an naive approach by using the average of the rewards over a single trajectory. As a final trick we normalise the rewards by dividing by the standard deviation.

Exercise

This exercise must be completed within the `reinforce.py` file. We will create a "REINFORCE with baseline" agent to balance the cart-pole (`CartPole-v1`).

1. Complete the `compute_returns_naive_baseline` method. This method is responsible for computing the return of a trajectory, at each time-step. The input will be an array of rewards along with the discount factor γ. Remember to apply baseline operations as described above.

2. Complete the `reinforce_naive_baseline` algorithm.

3. Run REINFORCE and REINFORCE with baseline on `Cart Pole`, then average the total return over five runs and plot the mean with boundaries of one standard deviation on the same axis (as done in exercise 1.5).

3 Actor-Critic

Although the REINFORCE with learned baseline method learns both a policy and a state-value function, we do not consider it to be an actor–critic method because its state-value function is used only as a baseline, not as a critic. That is, it is not used for bootstrapping (updating the value estimate for a state from the estimated values of subsequent states), but only as a baseline for the state whose estimate is being updated. This is a useful distinction, for only through bootstrapping do we introduce bias and an asymptotic dependence on the quality of the function approximation. As we have seen, the bias introduced through bootstrapping and reliance on the state representation is often beneficial because it reduces variance and accelerates learning. REINFORCE with baseline is unbiased and will converge asymptotically to a local minimum, but like all Monte Carlo methods it tends to learn slowly (produce estimates of high variance) and to be inconvenient to implement online or for continuing problems. With temporal-difference methods we can eliminate these inconveniences, and through multi-step methods we can flexibly choose the degree of bootstrapping. In order to gain these advantages in the case of policy gradient methods we use actor–critic methods with a bootstrapping critic.

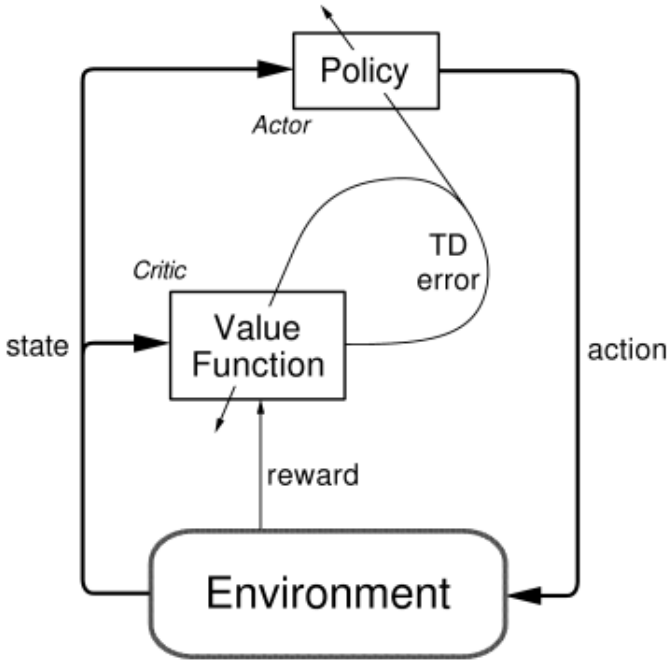

Figure 4: Actor Critic

Exercise

Create an agent to control the `LunarLander-v2` in environment. You can use any actor-critic algorithm.

1. Short write-up on the following details:
    - Explain the algorithm
    - Explain the actor and critic network architecture
2. Create a GIF of your agent controlling the `LunarLander`

4 REINFORCE with learned baseline (Optional)

Previously we implemented the REINFORCE algorithm and demonstrated how a baseline reduces variance and improves performance. We know that the value function v would be an ideal baseline (if we knew it) and used the average return as a rough approximation. In this section we will try to learn the value function so that we can use it as a baseline in REINFORCE. Like the policy, we will parametrise the value function using a simple neural network with parameters ω. Recall that the policy gradient theorem gives us

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\,(G_t - v_\omega(s_t))\right], \quad \text{where } G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) \tag{5}$$

Here we have used our parametrised value function v_ω as the baseline. **But how do we learn v_ω?** Well, we can use Monte-Carlo method! Since we can calculate the discounted returns G_t we can just minimize

$$\frac{1}{2}\sum_{t=0}^{T} |G_t - v_\omega(s_t)|^2 \tag{6}$$

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 5: REINFORCE with Baseline (Source: Sutton & Barto Section 13.4, page 330)

In practice we can define a composite loss:

$$\texttt{loss} = \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)\,(G_t - \text{detach}(v_\omega(s_t))) + \frac{1}{2}\sum_{t=0}^{T} |G_t - v_\omega(s_t)|^2. \tag{7}$$

This also allows us to perform the policy and value function updates in a single step.

Side note: detach is to ensure the gradients are only taken with respect to θ and not ω. $v_\omega(s_t)$ can be thought of as a constant.

Exercise

In this exercise we will train an agent to play `LunarLander-v2` using REINFORCE with a learned baseline. This exercise must be completed within the `reinforce_with_learned_baseline.py` file.

1. Define the policy and value function. For efficiency we can use a single network with two heads. For a given state, the first head will output a Categorical distribution over the actions while the second head will return the value of the state. Refer to figure 6.

2. Complete the `compute_returns` method, use the trick of 'normalising' the returns (i.e. subtract the mean and divide by the standard deviation) that we did in the previous exercise.

3. Complete the REINFORCE with learned baseline algorithm.

4. Create a GIF of your trained agent playing the `Lunar Lander`

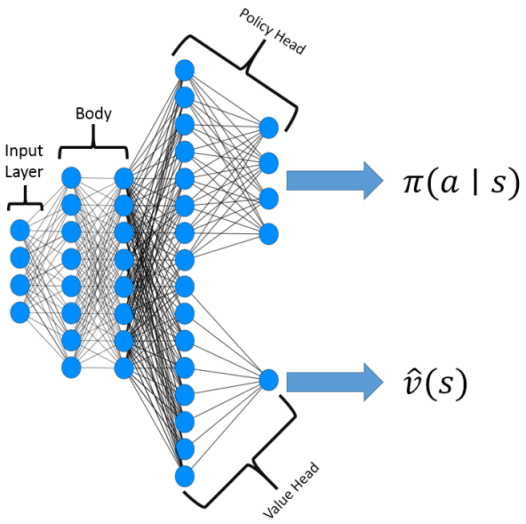

Figure 6: Two headed network