Reinforcement Learning for Beginners

Jin Hyun Park

1 Introduction

This material is to help understand the mathematical foundation of reinforcement learning. Unlike other simplified or abbreviated mathematical equations, this article aims to deliver deep mathematical background of reinforcement learning. It will cover various topics of reinforcement learning algorithms beginning from Q-learning and SARSA to the latest algorithms. In addition, code for each algorithm will be provided if it is possible. Please note that the writer of this article is currently studying reinforcement learning and it is been only a few months to study this topic by himself; therefore, there might be some errors or misunderstandings!

2 Preliminary Facts

- Logarithmic Properties:
  - Product Rule
$$\log_a(xy) = \log_a x + \log_a y \tag{1}$$
  - Quotient Rule
$$\log_a \frac{x}{y} = \log_a x - \log_a y \tag{2}$$
- Expectation of x:
$$E[x] = \int_x xp(x)\,dx, \quad x \sim p(x) \tag{3}$$
- Expectation of f(x):
$$E[f(x)] = \int_x f(x)p(x)\,dx, \quad x \sim p(x) \tag{4}$$
- Marginalization:
$$\int_x p(x,y)\,dx = p(y)\int_y p(x,y)\,dy = p(x) \tag{5}$$
- Bayesian Rule:
$$p(x,y) = p(x|y)\cdot p(y) \tag{6}$$
$$p(x,y|z) = p(x|y,z)\cdot p(y|z) \tag{7}$$
- Law of large numbers (LLN):
$$E[x] \triangleq \int_x xp(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i, \quad x \sim p(x) \tag{8}$$

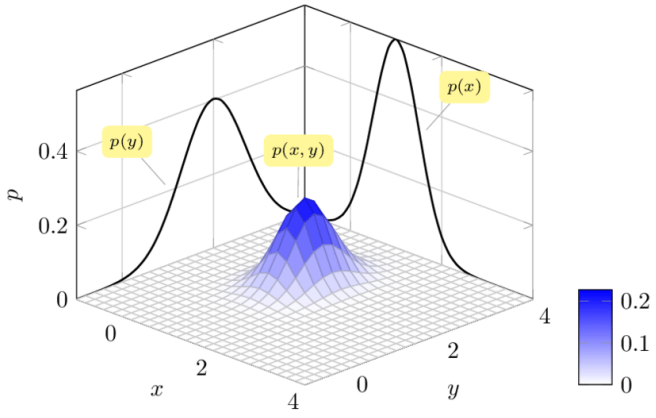

Figure 1: Visualising marginalization [1]

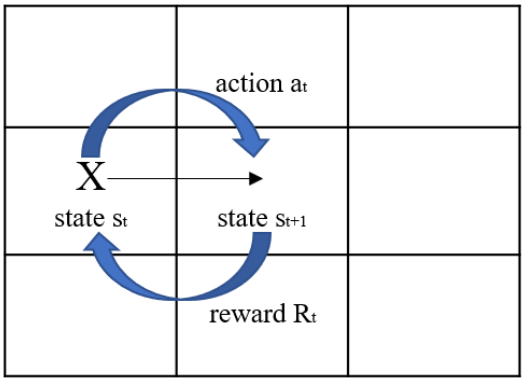

Figure 2: Figure to help understand basic principles of reinforcement learning

3 Return

A return is defined by R. Let's take a simple example to understand this. There is an agent X located in state s_t and the agent takes the action a_t at time t. Then, the reward that the agent will get is R_t (See Figure 2). G_t is a sum of R_ts multiplied by discount factor γ.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \tag{9}$$
$$= R_t + \gamma G_{t+1} \tag{10}$$

4 Discount Factor (γ)

The discount factor determines how important the agent's immediate reward is. If γ is small, the agent will be myopic and learn about actions which are focused on the immediate reward. The reward cannot be easily propagated when it is small. In opposition, if γ is large, the agent will not be myopic and take the future rewards into account when learning actions. The reward can be easily propagated when it is large. According to this discussion [2], γ is

equivalent to selecting a time horizon. We can formally write this by the following

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \tag{11}$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{12}$$

$$= \sum_{\Delta t=0}^{\infty} e^{-\Delta t/\tau} R_{t+\Delta t} \tag{13}$$

where $\gamma = e^{-1/\tau}$ and $k = \Delta t$. We can clearly observe that the time horizon τ is related to the discount factor γ. For example, if $\gamma = 1$, then $\tau = \infty$ where it means the reward infinitely propagates. See Figure 3 for another example.

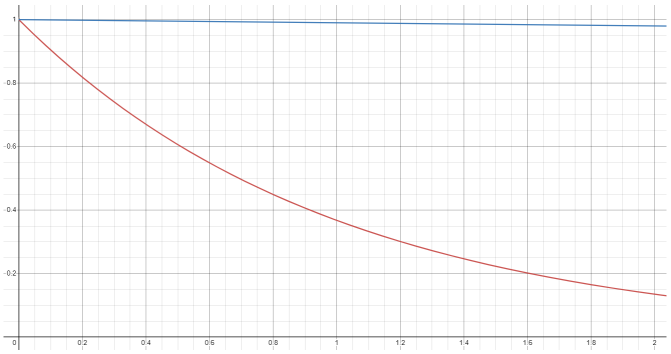

Figure 3: A graph to help understand the notion of time horizon. Assume $\tau = 1$ for red graph: $y = e^{-\Delta t}$. Assume $\tau = 100$ for blue graph: $y = e^{-\Delta t/100}$. Keep in mind that Δt is always positive.

5 State Value Function and Action Value Function

State Value Function is an expectation of G_t given the state s_t

$$V(s_t) \triangleq \int_{a_t:a_\infty} G_t \cdot p(a_t, s_{t+1}, a_{t+1}, \dots | s_t)\, da_t : a_\infty \tag{14}$$

Action Value Function is an expectation of G_t given the state s_t and the action a_t

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}:a_\infty} G_t \cdot p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)\, ds_{t+1} : a_\infty \tag{15}$$

6 Bellman Equation

It is called Bellman equation when V and Q can be represented by using next V and next Q, respectively.

- Representing V by using Q: Using the Bayesian rule (see equation 7), $p(a_t, s_{t+1}, a_{t+1}, \dots | s_t)$ in State Value Function (see equation 14) can be converted into $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t) \cdot p(a_t | s_t)$. By replacing $p(a_t, s_{t+1}, a_{t+1}, \dots | s_t)$ into $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t) \cdot p(a_t | s_t)$, we can get the following equation.

  $$V(s_t) \triangleq \int_{a_t} \int_{s_{t+1}:a_\infty} G_t \cdot p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)\, ds_{t+1} : a_\infty\, p(a_t | s_t) da_t \tag{16}$$

  And this equation simplifies into the following equation.

  $$V(s_t) \triangleq \int_{a_t} Q(s_t, a_t) \cdot p(a_t | s_t)\, da_t \tag{17}$$

  This equation implies that $V(s_t)$ is an expectation of $Q(s_t, a_t)$ under the action a_t.

- Representing V by using next V: Using the Bayesian rule again (see equation 7), $p(a_t, s_{t+1}, a_{t+1}, \dots | s_t)$ in State Value Function (see equation 14) can be converted into $p(a_{t+1}, \dots | s_t, a_t, s_{t+1}) \cdot p(a_t, s_{t+1} | s_t)$. It should be noted that by using Markov Decision Process (MDP), this can be simplified into $p(a_{t+1}, \dots | s_{t+1}) \cdot p(a_t, s_{t+1} | s_t)$. By replacing $p(a_t, s_{t+1}, a_{t+1}, \dots | s_t)$ into $p(a_{t+1}, \dots | s_{t+1}) \cdot p(a_t, s_{t+1} | s_t)$, we get the following equation.

$$V(s_t) \triangleq \int_{a_t, s_{t+1}} \int_{a_{t+1}:a_\infty} (R_t + \gamma G_{t+1}) \cdot p(a_{t+1}, \dots | s_{t+1}) \, da_{t+1} : a_\infty \, p(a_t, s_{t+1} | s_t) \, da_t, s_{t+1} \tag{18}$$

$$\triangleq \int_{a_t, s_{t+1}} (R_t + \gamma V(s_{t+1})) \cdot p(a_t, s_{t+1} | s_t) \, da_t, s_{t+1} \tag{19}$$

$$\triangleq \int_{a_t, s_{t+1}} (R_t + \gamma V(s_{t+1})) \cdot p(s_{t+1} | s_t, a_t) \cdot p(a_t | s_t) \, da_t, s_{t+1} \tag{20}$$

Normally, $p(s_{t+1}|s_t, a_t)$ is called transition probability and $p(a_t|s_t)$ is called policy.

- Representing Q by using V: Using the Bayesian rule again (see equation 7), $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)$ in Action Value Function (see equation 15) can be rewritten as $p(a_{t+1}, s_{t+2}, \dots | s_t, a_t, s_{t+1}) \cdot p(s_{t+1} | s_t, a_t)$. We should keep in mind that we can apply MDP; thus, it can be reduced as $p(a_{t+1}, s_{t+2}, \dots | s_{t+1}) \cdot p(s_{t+1} | s_t, a_t)$. By replacing $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)$ into $p(a_{t+1}, s_{t+2}, \dots | s_{t+1}) \cdot p(s_{t+1} | s_t, a_t)$, we can get the following equation.

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}} \int_{a_{t+1}:a_\infty} (R_t + \gamma G_{t+1}) \cdot p(a_{t+1} : a_\infty | s_{t+1}) \, da_{t+1} : a_\infty \, p(s_{t+1} | s_t, a_t) ds_{t+1} \tag{21}$$

$$\triangleq \int_{s_{t+1}} (R_t + \gamma V(s_{t+1})) \cdot p(s_{t+1} | s_t, a_t) \, ds_{t+1} \tag{22}$$

- Representing Q by using next Q: Using the Bayesian rule again (see equation 7), $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)$ in Action Value Function (see equation 15) can be rewritten as $p(s_{t+2} : a_\infty | s_t, a_t, s_{t+1}, a_{t+1}) \cdot p(s_{t+1}, a_{t+1} | s_t, a_t)$. Here we can apply MDP again; hence, it can be reduced into $p(s_{t+2} : a_\infty | s_{t+1}, a_{t+1}) \cdot p(s_{t+1}, a_{t+1} | s_t, a_t)$. By replacing $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)$ into $p(s_{t+2} : a_\infty | s_{t+1}, a_{t+1}) \cdot p(s_{t+1}, a_{t+1} | s_t, a_t)$, we can get the following equation.

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}, a_{t+1}} \int_{s_{t+2}:a_\infty} (R_t + \gamma G_{t+1}) \cdot p(s_{t+2} : a_\infty | s_{t+1}, a_{t+1}) \, ds_{t+2} : a_\infty \, p(s_{t+1}, a_{t+1} | s_t, a_t) \, ds_{t+1}, a_{t+1} \tag{23}$$

$$\triangleq \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \cdot p(s_{t+1}, a_{t+1} | s_t, a_t) \, ds_{t+1}, a_{t+1} \tag{24}$$

$$\triangleq \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \cdot p(a_{t+1} | s_{t+1}) \cdot p(s_{t+1} | s_t, a_t) \, ds_{t+1}, a_{t+1} \tag{25}$$

7 Optimal Policy

Optimal policy is a policy that maximizes $V(s_t)$. Assuming $Q(s_t, a_t)$ has an optimal policy for every policy in the future[1], it can be formulated as:

$$p^*(a_t|s_t) = \arg\max_{p(a_t|s_t)} \int_{a_t} Q^*(s_t, a_t) \cdot p(a_t|s_t) \, da_t \tag{26}$$

The optimal policy of $p(a_t|s_t)$ is $p^*(a_t|s_t) = \delta(a_t - a_t^*)$ where $a_t^* \triangleq \arg\max_{a_t} Q^*(s_t, a_t)$. This simply means to choose a_t that maximizes $Q^*(s_t, a_t)$. It is important to note that $a_t^* \triangleq \arg\max_{a_t} Q^*(s_t, a_t)$ is the foundation of *Greedy action* in reinforcement learning. Check your understanding about optimal policy by question 1 and 2.

[1] This means that $p(a_{t+1}|s_{t+1})$, $p(a_{t+2}|s_{t+2})$, ... are all optimal policies. We will represent the optimal policy by using *. For example, the optimal policy for $p(a_{t+1}|s_{t+1})$ is $p^*(a_{t+1}|s_{t+1})$

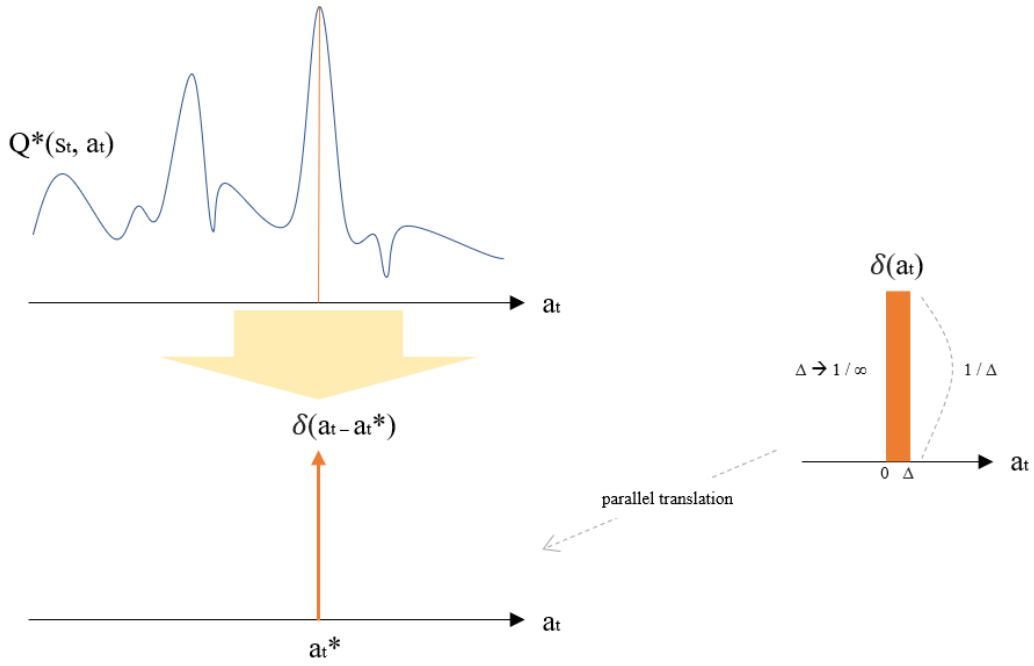

Figure 4: Visualisation of optimal policy

8 Monte Carlo and Temporal Difference

In the previous subsection, we have seen that we had to assume $Q(s_t, a_t)$ is optimal in order to acquire a_t^*. Then how can we get Q^*? In fact, it is not possible to acquire Q^* by solving an certain equation; rather, Q^* can be converged by using either Monte Carlo or Temporal Difference learning. Basically, we can think that there are two ways (Monte Carlo and Temporal Difference) to compute Q^*.

8.1 Monte Carlo

Monte Carlo method can be applied by using LLN (see equation 4). Retrieving $Q(s_t, a_t)$ from Action Value Function (see equation 15), we can derive the following.

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}:a_\infty} Gt \cdot p(s_{t+1} : a_\infty | s_t, a_t)\, ds_{t+1} : a_\infty \tag{27}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} {G_t}^{(i)} \tag{28}$$

How the above $Q(s_t, a_t)$ get closer to $Q^*(s_t, a_t)$? It is because $p(s_{t+1} : a_\infty | s_t, a_t)$ will always take greedy action. By taking the greedy action for each time step, $Q(s_t, a_t)$ will also get closer to $Q^*(s_t, a_t)$. Another thing that we can observe is that ${G_t}^{(i)}$ can be acquired when the agent reaches to the goal.

8.2 Temporal Difference (TD)

Similar to Monte Carlo, Temporal Difference method can be applied by using LLN (see equation 4). We will retrieve the definition of $Q(s_t, a_t)$ from Bellman Equation (see equation 25).

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}, a_{t+1}} (R_t + \gamma Q(s_{t+1}, a_{t+1})) \cdot p(a_{t+1}|s_{t+1}) \cdot p(s_{t+1}|s_t, a_t)\, ds_{t+1}, a_{t+1} \tag{29}$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} (R_t{}^{(i)} + \gamma Q(s_{t+1}{}^{(i)}, a_{t+1}{}^{(i)})) \tag{30}$$

Different from Monte Carlo method, the agent does not have to reach the goal to get a sample. The agent moves only one time to acquire $Q(s_t, a_t)$. This implies that the agent considers only the next circumstance $Q(s_{t+1}, a_{t+1})$ and this is exactly what equation 30 implies. The definition of TD is not fully explained. The author of this article strongly recommends to see the next subsection as well to fully understand the meaning of TD.

8.3 MC vs TD Trade-off

Firstly, let's discuss MC. We have learned that a single example of G_t can be acquired when the agent reaches to the goal (See equation 28). The variance of the G_t will be large; since, there are many ways to reach the goal. That is, the agent may take many steps before reaching the goal or few steps before reaching the goal. The good thing is that MC method is unbiased. This is because MC method simply chooses samples from G_ts.

On the other hand, TD method is biased; because, it samples the expectation. By looking at equation 30, we can see that $Q(s_t, a_t)$ is expectation of expectation. That is, inside $R_t{}^{(i)} + \gamma Q(s_{t+1}{}^{(i)}, a_{t+1}{}^{(i)})$, $Q(s_{t+1}{}^{(i)}, a_{t+1}{}^{(i)})$ is also expectation. We are sampling from expected value $Q(s_{t+1}{}^{(i)}, a_{t+1}{}^{(i)})$; hence, the sample is biased. The good thing is that TD method has small variance. This is because the agent only sees the next step.

Monte Carlo	Temporal Difference
bias ↓	bias ↑
variance ↑	variance ↓

Table 1: Comparison between MC and TD regarding bias and variance.

8.4 Incremental Monte Carlo

You might have heard about incremental Monte Carlo when studying reinforcement learning. We will talk about this and how it can be applied in TD method.

$$\overline{Q_N} \triangleq \frac{1}{N} \sum_{i=1}^{N} (R_t{}^{(i)} + \gamma Q(s_{t+1}{}^{(i)}, a_{t+1}{}^{(i)})) \tag{31}$$

$$= \frac{1}{N} (\overline{Q_{N-1}}(N-1) + R_t{}^{(N)} + \gamma Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)})) \tag{32}$$

$$= \overline{Q_{N-1}} + \frac{1}{N} (R_t{}^{(N)} + \gamma Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)}) - \overline{Q_{N-1}}) \tag{33}$$

We call this equation 33 as Incremental Monte Carlo update. So what is the meaning of this equation? Let's take a simple example to understand this. Consider $N = 10$ and we have currently 9 samples. Take a look at the following equation 34 that describes this situation. We can simply replace N to 10 in equation 33.

$$\overline{Q_{10}} = \overline{Q_9} + \frac{1}{10} (R_t{}^{(10)} + \gamma\, Q(s_{t+1}{}^{(10)}, a_{t+1}{}^{(10)}) - \overline{Q_9}) \tag{34}$$

The above implies that we can use $\overline{Q_9}$ to calculate $\overline{Q_{10}}$ which means that we can use the previous information to get the current information! We can see that the equation *incrementally* updates the $\overline{Q_N}$ and this is why it is called

Incremental Monte Carlo.

Let's play with the above equation a little bit more. We set $\frac{1}{N} = \alpha$ then we can acquire the following equation.

$$\overline{Q_N} = (1 - \alpha) \cdot \overline{Q_{N-1}} + \alpha \cdot (R_t{}^{(N)} + \gamma\, Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)})) \tag{35}$$

We can observe that the value of α has the boundary from 0 to 1; because, N is a natural number. If α is close to 0 then we stick to the $Q(s_t, a_t)$ (previous information); whereas, if α is close to 1 then we rely on the new information $R_t{}^{(N)} + \gamma Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)})$. α basically determines how much are we going to consider the new information when updating Q. This is why α is called learning rate.

There are more things to talk before we move on to the next subsection. Firstly, have a look at the equation 33. We formally call '$R_t{}^{(N)} + \gamma Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)}) - \overline{Q_{N-1}}$' as 'TD error' (more formally, it is called '1-step TD error') and '$R_t{}^{(N)} + \gamma Q(s_{t+1}{}^{(N)}, a_{t+1}{}^{(N)})$' as 'TD target'. Secondly, have a look at the equation 35. If we use equation 35 to update Q, then we call this SARSA algorithm and this could be formally expressed as the following:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (R_t + \gamma\, Q(s_{t+1}, a_{t+1})) \tag{36}$$

If we choose the best action a_{t+1} among a_{t+1}s, then we call it Q-Learning and it could be formally expressed as the following:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) \tag{37}$$

9 n-step Temporal Difference

10 Model-free vs Model-based

In Markov Decision Process, there are two properties: transition function and reward function. The transition function is a function that, given a state and an action, outputs a probability of moving to a new state. The reward function is a function that outputs a reward, given a state and an action. Both the transition function and the reward function together, it is called as 'model' of environment. In general, it is known that if we have the transition and reward functions (if the model is given), we can use policy iteration or value iteration (dynamic programming) to compute an optimal policy. An model-based algorithm uses the transition and reward functions to estimate the optimal policy. In summary, a MDP model is the problem and the optimal policy is the solution to the problem.

However, we do not always have the model which means that we do not always have the transition and reward functions. That is, we cannot obtain the policy from MDP when the MDP is unknown. In this case, the agent needs to interact with environment and estimate the optimal policy. Over the time steps, the agent understands the environment by taking actions and getting rewards. This is so called as 'trial and error' approach. A model-free algorithm does not use the transition and reward functions to estimate the optimal policy.

There is a good way to distinguish between model-based and model-free algorithms. We can see if the algorithm uses the transition or reward functions. For example, Q-Learning and SARSA algorithms are model-free because it does not use any probabilities (transition or reward functions) defined by the MDP. On the other hand, the policy improvement algorithm uses probabilities defined by the MDP (See equation 38). Therefore, policy iteration which uses the policy improvement algorithm is a model-based.

$$Q(s_t, a_t) \leftarrow \sum_{s_{t+1} \in \mathcal{S}, R_t \in \mathcal{R}} p(s_{t+1}, R_t | s_t, a_t)(R_t + \gamma V(s_{t+1})) \tag{38}$$

11 Q-Learning vs SARSA

We will now see more about Q-Learning and SARSA. Q-Learning algorithm is one of the most renowned reinforcement learning algorithm. It is very crucial to fully understand this algorithm; since, there are many algorithms based on Q-Learning. Let's have a look at some basic topics before we move on to Q-Learning and SARSA.

11.1 Exploitation and Exploration

Reinforcement learning is all about finding the optimal policy and getting the optimal reward from the given environment. To do so, we should use exploitation and exploration algorithm. Let's see why this is important and how this could be implemented. Let's have a look at Figure 5.

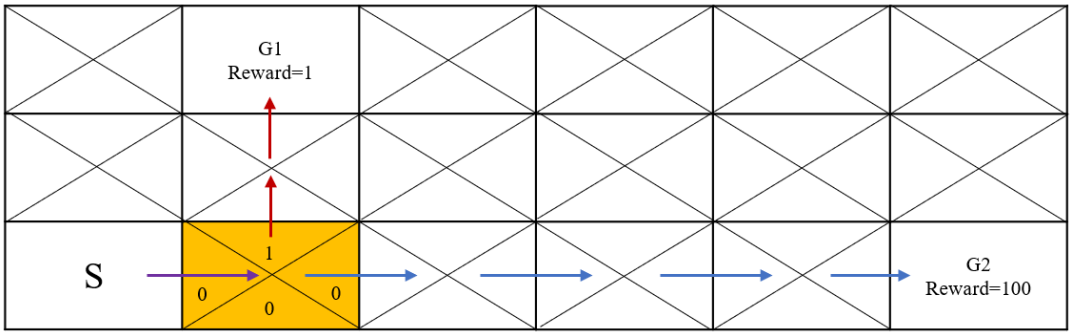

Figure 5: Red path: The first path that the agent found. Blue path: Another path that the agent can reach. Purple path: shared path

Imagine an agent starts from state S and found the path to $G1$ that rewards 1 to the agent. Assume the agent is using Q-Learning update rule. As you might have noticed, it is more likely to find $G1$ beforehand $G2$ because $G1$ is closer to S. This means that, without any exploration, the agent will always exploit the founded path (red path) even though there is a better state that rewards 100 (blue path). Here we can utilize exploitation and exploration algorithms.

11.1.1 ε-greedy

The first algorithm is ε-greedy (called epsilon greedy). ε-greedy enables the agent to find the better path by randomly moving on to the next state with probability ε. If you look at Figure 5, we can see that $Q(orange\_state, up) = 1$ and $Q(orange\_state, -up) = 0$. Without ε-greedy, the agent will always move to the upper state. However, the agent can explore to another state (e.g. $right, down, left$) by utilising ε-greedy. In this manner, the agent can find the blue path which gives more rewards.

ε-greedy Action Selection

Function *SELECT-ACTION(Q, ε)*:
    n ← uniform random number between 0 and 1;
    if $n < \varepsilon$ **then**
        A ← random action from the actions;
    else
        A ← $\arg\max_a Q(s, a)$;
    end if
    return selected action A;
end

Algorithm 1: ε-greedy

In fact, decaying ε-greedy is more general algorithm. The idea is very simple. The agent slowly decreases the degree of exploration for every episode. The agent does not have enough information in early episodes leading to explore more in early stage. After many episodes, the agent decreases the exploration rate to exploit the information.

Decaying ε-greedy Action Selection

for *each episode* i **do:**
  /* do something */;
  $e \leftarrow \varepsilon/(i+1)$;
  a $\leftarrow$ *SELECT-ACTION*(Q, e);
  /* do something */;
end

Algorithm 2: Decaying ε-greedy

11.1.2 Adding noise

The second algorithm is to add random noises to Q. To simplify the algorithm, let's assume there are only 4 actions that could be selected in a state.

Action Selection with random noise

Function *SELECT-ACTION(Q, noise)***:**
  $n_1, n_2, n_3, n_4 \leftarrow$ random noise (noise value depends on a model);
  $Q(s, a_i) \leftarrow Q(s, a_i) + n_i$;
  return $Q(s, a_i)$;
end

Algorithm 3: Random noise

ε-greedy algorithm will choose any action when n is smaller than ε. However, unlike ε-greedy algorithm, random noise algorithm helps the agent to avoid choosing the worst action but gives a chance to choose the second or third best action in a state (See Figure 6).

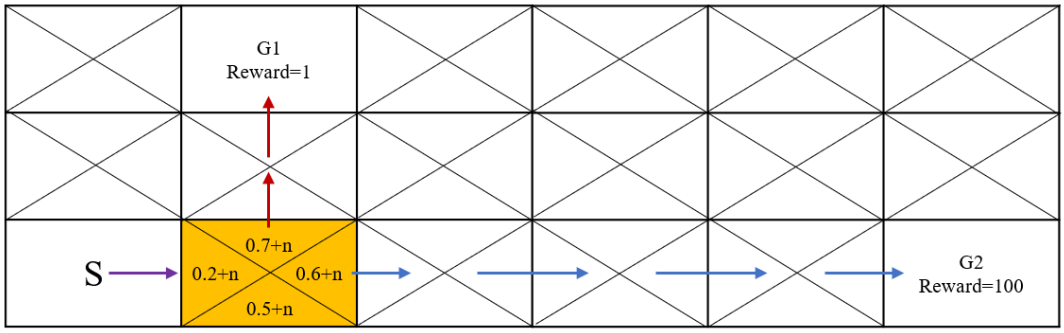

Figure 6: n denotes noise and subscripts of n is removed for simplification. Red path: The first path that the agent found. Blue path: Another path that the agent can reach. Purple path: shared path

11.2 Revisiting Discount Factor (γ)

Let's assume that an agent resides in 3 X 3 grid and the agent updates the grid by using Q-learning (See equation 37). There are many ways to reach the goal from starting state. Figure 7 shows two possible examples of the ways. Which one do you think it is more efficient? Indeed, the middle one looks more efficient than the other. The agent knows that moving to *right* is better than moving to *up* in *orange_state* because the $Q(orange\_state, right) = 0.9$ is greater than $Q(orange\_state, up) = 0.73$. Therefore, we can think the discount factor helps the agent to find the optimal path to the goal.

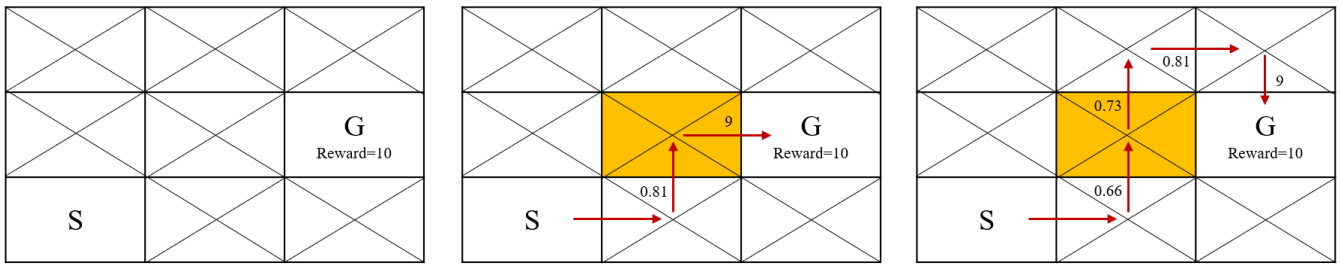

Figure 7: Visualising the effect of discount factor γ. S: starting state. G: Goal. Left board: Initialised Board. Middle board: The agent takes 3 steps to reach the goal. Right board: The agent takes 5 steps to reach the goal.

11.3 Q-Learning

Q-learning (off-policy TD control)

Algorithm hyper-parameters: learning rate (or step size) $\alpha \in (0, 1]$, $\varepsilon > 0$;

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal\_state}, \cdot) = 0$;
foreach *episode* **do**
  Initialize all states (place an agent at the starting state);
  foreach *step* **do**
    Choose a_t from s_t using policy derived from Q (e.g., ε-greedy);
    Take action a_t, observe R_t, s_{t+1};
    $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}))$;
    $s_t \leftarrow s_{t+1}$;
    (break when s_t is terminal)
  end foreach
end foreach

Algorithm 4: Q-learning

11.4 SARSA

SARSA (on-policy TD control)

Algorithm hyper-parameters: learning rate (or step size) $\alpha \in (0, 1]$, $\varepsilon > 0$;

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal\_state}, \cdot) = 0$;
foreach *episode* **do**
  Initialize all states (place an agent at the starting state);
  Choose a_t from s_t using policy derived from Q (e.g., ε-greedy);
  foreach *step* **do**
    Take action a_t, observe R_t, s_{t+1};
    Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g., ε-greedy);
    $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(R_t + \gamma Q(s_{t+1}, a_{t+1}))$;
    $s_t \leftarrow s_{t+1}$;
    $a_t \leftarrow a_{t+1}$;
    (break when s_t is terminal)
  end foreach
end foreach

Algorithm 5: SARSA

12 Deep Q-Learning

In this section, we will cover several recent papers about deep Q-learning. We firstly need to know that why there is a word 'deep' in front of Q-Learning. Take a look at the following Figure **??**.

12.1 DQN (NIPS 2013)

12.2 DQN (Nature 2015)

12.3 Double DQN (DDQN)

12.4 Dueling DQN

12.5 Prioritized Experience Replay (PER)

12.6 A Distributional Perspective on Reinforcement Learning

13 Policy Based Algorithms

Policy based algorithms are different from value based algorithms (i.e. Q-learning and SARSA). One of the critical problem of value based approach to solve reinforcement learning problem is that it could be only used in scenarios where actions are discrete. For example, let's assume that we would like to make the robot that moves its legs to move forward. The action space of the robot should be continuous rather than discrete, because the action space will be defined by an angle (See Figure 8). If the action is discrete, we should assign a node for each action, which means that $node_0$ corresponds to $\theta = 0$, $node_1$ corresponds to $\theta = 0.001$, and so on. In this way, it will be impossible for a DQN model to properly adjust its parameters to find the best action for each state.

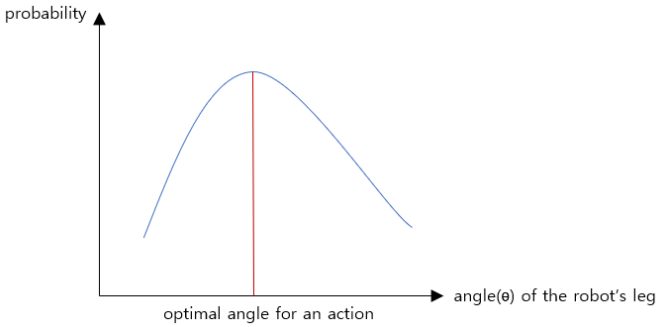

Figure 8: The action space of the robot is continuous.

In addition, when the action is continuous, Q function becomes complicated (i.e. Q function is non-linear). Finding the action that maximizes Q: $\arg\max_{a_t} Q(a_t|s_t)$ turns into a new optimization problem as we are going to use gradient descent method (See Figure 9). We need to solve an optimization problem for every step and this is not trivial.

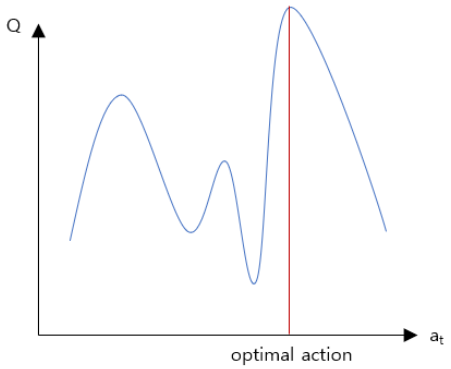

Figure 9: Finding the optimal action becomes an optimization problem.

Another advantage of policy based algorithms over value based algorithms is that the policy based algorithms can find a stochastic policy. This means that the policy can be represented by a probability distribution. We can parameterize(θ) the policy like Figure 10 and samples from the distribution to get the policy.

This method is very useful in the following scenario (See Figure 11). Let's assume the state can be represented by the following $s = [up, right, down, left]$. If there is a wall on that side then it is 1 otherwise 0. For instance, if the agent is in the grey area then $s = [1, 0, 1, 0]$. We call this *partially observable state* which means that the agent cannot distinguish the left grey state and right grey state, but we know that the left and right are different. The agent will think that the two states are the exactly same state. We will assume that the agent learns this game

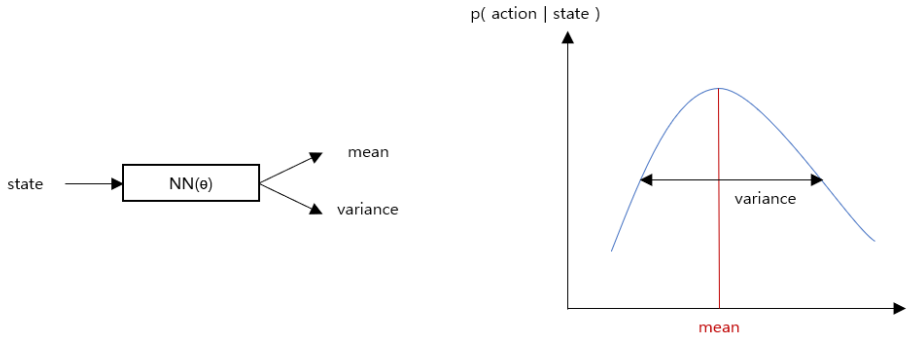

Figure 10: Input: state, Output: mean and variance of a policy

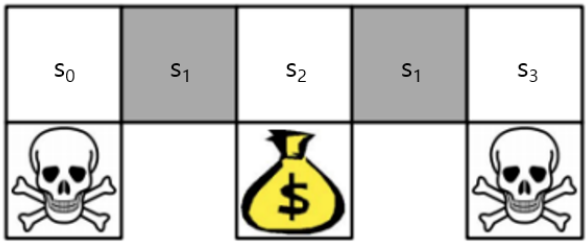

Figure 11: Understanding stochastic policy by using simple example. The agent cannot differentiate between the left and right grey state. This image is from Stanford CS234.

by using DQN and the agent is now at the grey state. There are two actions that the agent can perform: left or right. If $Q(grey, left) > Q(grey, right)$ then the agent moves to left (case 1) and if $Q(grey, left) < Q(grey, right)$ then the agent moves to right (case 2). When the agent is in the left grey state and case 1, the agent gets stuck, wandering between s_0 and s_1. When the agent is in the right grey state and case 1, the agent can reaches to the goal. The exact same problem arises for the second case (case 2) as well. Then, what will be the best policy for state s_1? The best policy for state s_1 is $p(left|s_1) = p(right|s_1) = \frac{1}{2}$. This can be realized by using a stochastic policy.

We have learned that policy based algorithm is better than the other in terms of two aspects: continuous action and stochastic policy. What how can we define our new objective function of policy based algorithm? In reinforcement learning, our goal is to always maximize rewards/returns. More precisely, it is to maximize *expected* rewards/returns. In policy based, maximizing the *expected* rewards/returns from the beginning is our goal. Let's retrieve the definition of return from equation 11.

$$G_0 = R_0 + \gamma R_1 + \gamma^2 R_2 + \dots \tag{39}$$

A policy based algorithm tries to maximize the expectation of G_0, namely $E[G_0]$. We will set $E[G_0]$ as our objective function and our goal is to maximize this.

$$J \triangleq E[G_0] \tag{40}$$

$$= \int_{s_0:a_\infty} G_0 \cdot p(s_0, a_0, s_1, a_1 \cdots)\, ds_0 : a_\infty \tag{41}$$

$$\tag{42}$$

by setting $\tau = s_0, a_0, s_1, a_1 \cdots$ we get the following (τ represents a trajectory):

$$J = \int_\tau G_0 \cdot p(\tau)\, d\tau \tag{43}$$

To maximize the expectation of G_0, we should find the optimal policy for each state. Then, where is the policy in equation 43? It is inside $p(\tau)$. Inside $p(\tau)$, we have $p_\theta(a_0|s_0)$, $p_\theta(a_1|s_1)$, $\cdots$. We can rewrite the objective function with parameter θ. See Figure 10 to see how the policy can be parameterized.

$$J_\theta = \int_\tau G_0 \cdot p_\theta(\tau)\, d\tau \tag{44}$$

The objective function is now defined and we have parameterized the function by using θ. We can now differentiate the function by θ and use gradient ascent method to maximize the objective function.

$$\nabla_\theta J_\theta = \nabla_\theta \int_\tau G_0 \cdot p_\theta(\tau)\, d\tau \tag{45}$$

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J_\theta \tag{46}$$

where

$$\nabla_\theta J_\theta \triangleq \frac{\partial J_\theta}{\partial \theta} \tag{47}$$

We call $\nabla_\theta J_\theta$ **policy gradient**. We will simplify equation 45. After simplification, we will be able to redefine $\nabla_\theta J_\theta$ using the policy $p(a|s)$. Calculating summation after differentiation (equation 48) and calculating differentiation after summation 45 is the same.

$$\nabla_\theta J_\theta = \int_\tau G_0 \cdot \nabla_\theta p_\theta(\tau)\, d\tau \tag{48}$$

Equation 48 is no longer expectation. Thus, we apply a trick to make the equation into an expectation format.

$$\nabla_\theta J_\theta = \int_\tau G_0 \cdot (\nabla_\theta \ln p_\theta(\tau)) \cdot p_\theta(\tau)\, d\tau \tag{49}$$

We can now make gradient with sample means. We will try to simplify $G_0 \cdot (\nabla_\theta \ln p_\theta(\tau))$. Firstly, G_0 can be expressed by the following:

$$G_0 = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots \tag{50}$$

Secondly, we can rewrite $p_\theta(\tau)$ by the following:

$$\begin{aligned}
p_\theta(\tau) &= p_\theta(s_0, a_0, s_1, a_1, s_2, a_2, \cdots) && (51)\\
&= p(s_0) \cdot p_\theta(a_0, s_1, a_1, \cdots | s_0) && (52)\\
&= p(s_0) \cdot p_\theta(a_0|s_0) \cdot p_\theta(s_1, a_1, \cdots | s_0, a_0) && (53)\\
&= p(s_0) \cdot p_\theta(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot p_\theta(a_1, s_2, a_2, \cdots | s_0, a_0, s_1) && (54)\\
&= p(s_0) \cdot p_\theta(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot p_\theta(a_1|s_0, a_0, s_1) \cdot p_\theta(s_2, a_2, \cdots | s_0, a_0, s_1, a_1) && (55)\\
&= p(s_0) \cdot p_\theta(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot p_\theta(a_1|s_0, a_0, s_1) \cdot p(s_2|s_0, a_0, s_1, a_1) \cdot p_\theta(a_2, s_3, a_3, \cdots | s_0, a_0, s_1, a_1, s_2) && (56)\\
&= p(s_0) \cdot p_\theta(a_0|s_0) \cdot p(s_1|s_0, a_0) \cdot p_\theta(a_1|s_1) \cdot p(s_2|s_1, a_1) \cdot p_\theta(a_2, s_3, a_3, \cdots | s_2) && (57)\\
&= \cdots && (58)
\end{aligned}$$

It is important to note that the parameter θ is only applied to the policies. It can be observed that there is no θ in $p(s_0)$ and other transition probabilities, namely $p(s_{t+1}|s_t, a_t)$. In logarithms, multiplication can be expressed by summation. Also, $\ln p_\theta(\tau)$ is differentiated by θ. Therefore, by calculating $\nabla_\theta \ln p_\theta(\tau)$, we get the following:

$$\nabla_\theta \ln p_\theta(\tau) = \nabla_\theta \sum_{t=0}^{\infty} \ln p_\theta(a_t|s_t) \tag{59}$$

We can rewrite $\nabla_\theta J_\theta$ by the following:

$$\nabla_\theta J_\theta = \int_\tau G_0 \cdot (\nabla_\theta \ln p_\theta(\tau)) \cdot p_\theta(\tau)\, d\tau \tag{60}$$

$$= \int_\tau (R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots) \cdot (\nabla_\theta \sum_{t=0}^{\infty} \ln p_\theta(a_t|s_t)) \cdot p_\theta(\tau)\, d\tau \tag{61}$$

$$= \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (\sum_{k=t}^{\infty} \gamma^k \cdot R_k)) \cdot p_\theta(\tau)\, d\tau \tag{62}$$

$$= \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (\sum_{k=t}^{\infty} \gamma^t \cdot \gamma^{k-t} \cdot R_k)) \cdot p_\theta(\tau)\, d\tau \tag{63}$$

$$= \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (\gamma^t \cdot G_t)) \cdot p_\theta(\tau)\, d\tau \tag{64}$$

When t gets larger, γ^t becomes very small. This implies that we apply very small weight to the future reward, therefore the agent cannot learn properly. After removing γ^t, we can acquire the simplified version of equation 45. You might wonder how the equation 61 could be simplified into 62. See (A) for more details.

$$\nabla_\theta J_\theta \approx \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \cdot p_\theta(\tau)\, d\tau \tag{65}$$

13.1 REINFORCE

Deriving REINFORCE algorithm begins from the equation 65. We can approximate the equation 65 by using sampling.

$$\nabla_\theta J_\theta \approx \frac{1}{N}\sum_{i=1}^{N}(\sum_{t=0}^{\infty}(\nabla_\theta \ln p_\theta(a_t^{(i)}|s_t^{(i)}) \cdot G_t^{(i)})) \tag{66}$$

where

$$\sum_{t=0}^{\infty}(\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \sim p_\theta(\tau) \tag{67}$$

The time step t starts from 0 to ∞ for each sample. Basically, we can get a single sample when the agent goes into the terminal state (i.e. when the episode finishes). Hence, there will be too much computation if we want to sample many samples (i.e. large N). To reduce the computation, we can set $N = 1$. Rewriting the objective function, we get the following:

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty}(\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \tag{68}$$

REINFORCE algorithm

Algorithm parameters: policy parameter θ;
Algorithm hyper-parameters: maximum episode T and learning rate α;

Initialize policy parameter θ, choose maximum episode T, and learning rate α;
foreach *episode* **do**
  Generate a trajectory $\tau : s_0, a_0, R_0, \ldots$, following $p_\theta(a_t|s_t)$;
  Calculate G_t using R_t where $t = 0, 1, \ldots, T-1$;
  foreach *step* **do**
    $\theta \leftarrow \theta + \alpha \cdot \sum_{t=0}^{\infty}(\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t)$;
  end foreach
  (Break when *episode* is equal to T)
end foreach

Algorithm 6: REINFORCE algorithm

REINFORCE algorithm is unbiased but high variance. It is unbiased, because we have used sampling method. However, its variance is very high, as a single trajectory that we have sampled would be highly different from the other trajectories that we will sample afterwards. The algorithm would take very long time to converge because of high variance.

13.2 Actor Critic (AC)

We will start from the equation 65 to induce Actor-Critic algorithm.

$$\nabla_\theta J_\theta \approx \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \cdot p_\theta(\tau)\, d\tau \tag{69}$$

$$= \int_\tau \sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \cdot p_\theta(s_{t+1}, a_{t+1}, \cdots | s_0, a_0, \cdots s_t, a_t) \cdot p_\theta(s_0, a_0, \cdots s_t, a_t)\, d\tau \tag{70}$$

$$= \sum_{t=0}^{\infty} \int_\tau (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \cdot p_\theta(s_{t+1}, a_{t+1}, \cdots | s_0, a_0, \cdots s_t, a_t) \cdot p_\theta(s_0, a_0, \cdots s_t, a_t)\, d\tau \tag{71}$$

$$= \sum_{t=0}^{\infty} \int_\tau (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot G_t) \cdot p_\theta(s_{t+1}, a_{t+1}, \cdots | s_t, a_t) \cdot p_\theta(s_0, a_0, \cdots s_t, a_t)\, d\tau \tag{72}$$

$$= \sum_{t=0}^{\infty} \int_{s_0, a_0, \cdots s_t, a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \tag{73}$$

$$\int_{s_{t+1}, a_{t+1}, \cdots} G_t \cdot p_\theta(s_{t+1}, a_{t+1}, \cdots | s_t, a_t)\, ds_{t+1}, a_{t+1}, \cdots p_\theta(s_0, a_0, \cdots s_t, a_t)\, ds_0, da_0, \cdots, s_t, a_t \tag{74}$$

Considering the equation 15, the above could be simplified by the following. The definition of Q is used.

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_0, a_0, \cdots s_t, a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot Q(s_t, a_t) \cdot p_\theta(s_0, a_0, \cdots s_t, a_t)\, ds_0, da_0, \cdots, s_t, a_t \tag{75}$$

After applying marginalization to p_θ, we get the following:

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t, a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot Q(s_t, a_t) \cdot p_\theta(s_t, a_t)\, ds_t, a_t \tag{76}$$

By solving the equation, we have replaced G into Q. Considering the fact that we were able to calculate G when the episode terminates, we no longer need to wait for the episode to be terminated by using Q. In Actor-Critic algorithm, Q is parameterized by w and it gets updated similar to SARSA algorithm.

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t, a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot Q_w(s_t, a_t) \cdot p_\theta(s_t, a_t)\, ds_t, a_t \tag{77}$$

where Q_w gets updated by the following TD-error:

$$||(R_t + \gamma \cdot Q_w(s_{t+1}, a_{t+1})) - Q_w(s_t, a_t)||_2^2 \tag{78}$$

An actor updates $p_\theta(a_t|s_t)$ and a critic updates Q_w. Unlike REINFORCE algorithm, Actor-Critic algorithm updates its θ and w for every step. This is possible since Q can be updated once s_t and a_t are acquired by the policy. This implies that we do not take $\sum_{t=0}^{\infty}$ into account when running the algorithm. Instead of waiting for the episode to be finished, 1-step TD is used. Summing up all the details, we can write the following Actor-Critic algorithm.

Actor-Critic algorithm

Algorithm parameters: actor(policy) parameter θ and critic parameter w;
Algorithm hyper-parameters: maximum step T, discount factor γ, learning rate for actor α, and critic β;

Initialize θ and w. Choose maximum step T, discount factor γ, learning rate α, and β;
repeat
  Actor update: $\theta \leftarrow \theta + \alpha \cdot Q_w(s_t, a_t) \cdot \nabla_\theta \ln p_\theta(a_t|s_t)$;
  Critic update: $w \leftarrow w + \beta \cdot (R_t + \gamma\, Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)) \cdot \nabla_w Q_w(s_t, a_t)$
until *we find satisfactory parameters:* θ *and* w;

Algorithm 7: Actor-Critic algorithm

The algorithm is biased and low variance compared to REINFORCE algorithm. It is biased, because we ignored $\sum_{t=0}^{\infty}$ and used 1-step TD error. In addition, we have parameterized Q with w. On the contrary, it is low variance, since we have replaced G with Q. In fact, it was mentioned in subsection 8.3 that (1-step) TD method is biased and low variance.

13.3 Advantage Actor Critic (A2C)

Previously, we have covered that Actor Critic algorithm helps to reduce the variance compared to REINFORCE algorithm. Similarly, A2C algorithm aims to have less variance than AC algorithm. We will start from the following equation to derive A2C.

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t,a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot Q_w(s_t,a_t) \cdot p_\theta(s_t,a_t)\, ds_t, a_t \tag{79}$$

The question is, if we plug $b(s_t)$ instead of $Q(s_t,a_t)$, will $b(s_t)$ affect the integration? Let's see whether $b(s_t)$ affect the integration or not. Plugging $b(s_t)$ instead of $Q(s_t,a_t)$,

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t,a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot b(s_t) \cdot p_\theta(s_t,a_t)\, ds_t, a_t \tag{80}$$

$$= \sum_{t=0}^{\infty} \int_{s_t,a_t} \frac{\nabla_\theta p_\theta(a_t|s_t)}{p_\theta(a_t|s_t)} \cdot b(s_t) \cdot p_\theta(a_t|s_t) \cdot p(s_t)\, ds_t, a_t \tag{81}$$

$$= \sum_{t=0}^{\infty} \int_{s_t} \int_{a_t} \nabla_\theta p_\theta(a_t|s_t)\, da_t\, b(s_t) \cdot p(s_t)\, ds_t \tag{82}$$

$$\tag{83}$$

Here, we can observe the following,

$$\int_{a_t} \nabla_\theta p_\theta(a_t|s_t)\, da_t \tag{84}$$

$$\nabla_\theta \int_{a_t} p_\theta(a_t|s_t)\, da_t = 0 \tag{85}$$

The above proves that if we replace $Q(s_t,a_t)$ with any function that is not related with the action, then the function (specifically, $b(s_t)$ and it is called *baseline*) does not affect $\nabla_\theta J_\theta$. Instead of using $b(s_t)$, we will use $V(s_t)$ from now on. In addition, we have proved that the function without the action does not affect the integration. Hence, we will substitute $Q(s_t,a_t) - V(s_t)$ for $Q(s_t,a_t)$. It is known that the sample variance reduces when using the above substitution. The substitution is not the optimal way to reduce the variance. However, it is the most commonly used way to reduce the variance. We call $Q(s_t,a_t) - V(s_t)$ as *advantage*. Solving the equation with $Q(s_t,a_t) - V(s_t)$,

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t,a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot (Q_w(s_t,a_t) - V(s_t)) \cdot p_\theta(s_t,a_t)\, ds_t, a_t \tag{86}$$

$$= \sum_{t=0}^{\infty} E_{s_t,a_t}[\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (Q - V)] \tag{87}$$

The problem of the above equation is that we have too many parameters. We have already parameterized Q and p with w and θ, respectively. But, $V(s_t)$ has been introduced, hence we again need to parameterize it. In order to overcome the issue, we should use the equation 21. This allows us to use only two parameters. Rewriting the gradient,

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} E_{s_t,a_t}[\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (Q - V)] \tag{88}$$

$$= \sum_{t=0}^{\infty} E_{s_t,a_t}[\nabla_\theta \ln p_\theta(a_t|s_t) \cdot E_{s_{t+1}}[R_t + \gamma \cdot V(s_{t+1}) - V(s_t)|s_t,a_t]] \tag{89}$$

As the equation is formulated with only V and p, we can parameterize them with w and θ, respectively. We will expand the above expectation to the integration format, and we get the following.

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t,a_t} \nabla_\theta \ln p_\theta(a_t|s_t) \int_{s_{t+1}} (R_t + \gamma V(s_{t+1}) - V(s_t)) \cdot p(s_{t+1}|s_t, a_t)\, ds_{t+1}\, p_\theta(s_t, a_t)\, ds_t, a_t \tag{90}$$

$$= \sum_{t=0}^{\infty} \int_{s_t,a_t,s_{t+1}} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot (R_t + \gamma V(s_{t+1}) - V(s_t)) \cdot p(s_{t+1}|s_t, a_t) \cdot p(s_t) \cdot p_\theta(a_t|s_t)\, ds_t, a_t, s_{t+1} \tag{91}$$

The sample for the above equation is: $\ln p_\theta(a_t|s_t) \cdot (R_t + \gamma V(s_{t+1}) - V(s_t))$, and the sample follows $p(s_{t+1}|s_t, a_t)$, $p(s_t)$, and $p_\theta(a_t|s_t)$. This implies that once θ gets updated by the sample that we have sampled, we should sample from the updated distribution for the next time step. For example, after we sample N times from the distribution which is constructed with θ_t and update θ ($\theta_{t+1} \leftarrow \theta_t$), we should sample from the distribution created with θ_{t+1} for the next time step to update θ ($\theta_{t+2} \leftarrow \theta_{t+1}$). We can visualize this by the following figure 12.

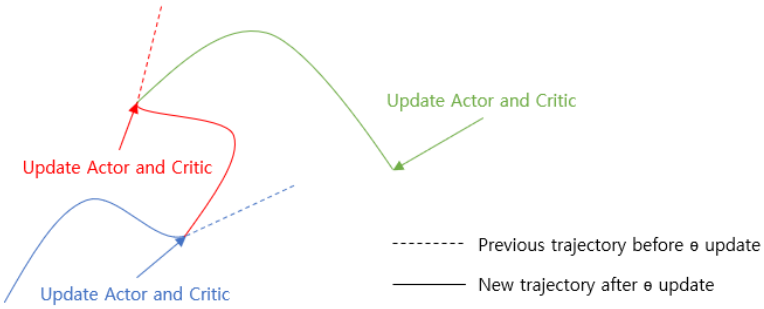

Figure 12: Visualisation of A2C algorithm

At this point, you might wonder how does $p(s_t)$ can be calculated. In fact, $p(s_t)$ can be rewritten as $p(s_t|s_{t-1}, a_{t-1})$. Summing up all the details that we have seen, we get the following equation:

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t,a_t,s_{t+1}} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot (R_t + \gamma V(s_{t+1}) - V(s_t)) \cdot p(s_{t+1}|s_t, a_t) \cdot p_\theta(a_t|s_t) \cdot p(s_t|s_{t-1}, a_{t-1})\, ds_t, a_t, s_{t+1} \tag{92}$$

It is strongly recommended to refer Figure 13 with Algorithm 8. As you might have noticed, A2C algorithm uses N samples to update the parameters θ and w unlike to AC algorithm which uses a single sample. AC algorithm did not take $\sum_{t=0}^{\infty}$ into account. However, A2C algorithm considers $\sum_{t=0}^{\infty}$ and use N samples to update parameters. This is possible because we have used $Q - V$ instead of Q. By subtracting V, it was able to reduce variance; therefore, now we can use more samples. $Q - V$ reduces the variance and using more than one sample increases the variance. They cancel each other in terms of variance.

Advantage Actor-Critic algorithm

Algorithm parameters: actor(policy) parameter θ and critic parameter w;
Algorithm hyper-parameters: maximum step T, discount factor γ, sample size N, learning rate for actor α, and critic β;

Initialize θ and w. Choose maximum step T, discount factor γ, sample size N, learning rate α, and β;
repeat
  Collect N samples (single sample: s_t, a_t, R_t, s_{t+1});
  Actor update: $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \sum_{i=t-N+1}^{t} \ln p_\theta(a_i|s_i) \cdot (R_i + \gamma V_w(s_{i+1}) - V_w(s_i))$;
  Critic update: $w \leftarrow w - \beta \cdot \nabla_w \sum_{i=t-N+1}^{t} (R_i + \gamma V_w(s_{i+1}) - V_w(s_i))^2$;
  Clear the batch (Remove N samples)
until *we find satisfactory parameters:* θ *and* w;

Algorithm 8: Advantage Actor-Critic algorithm

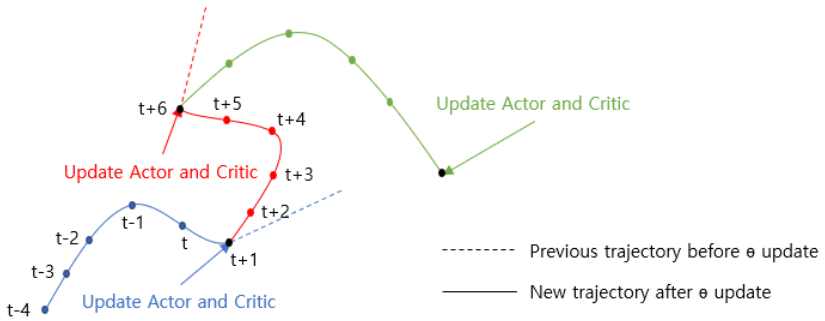

Figure 13: Visualisation of A2C algorithm when $N = 5$. Critic gets updated with $N = 5$ TD-errors

13.4 Asynchronous Advantage Actor Critic (A3C)

There three important aspects in A3C which are (1) correlations between batches, (2) n-step TD, and (3) Entropy. Let's find out what these are.

13.4.1 Correlations between batches

In A2C algorithm, we used batches to update both actor and critic. For example, if you see Figure 13, you can observe that we have used 5 samples to update them ($N = 5$). As we update them with consecutive experiences, it generates correlation between batches. This causes a problem because when we update a neural network, samples (or batches) should not be correlated (i.e. samples should satisfy i.i.d. property unless it is full gradient. Using i.i.d. samples for updating gradient could be considered as an unbiased estimate of full gradient.). A3C algorithm tries to alleviate the correlations between batches by introducing asynchronous (or multiple) agents. The agents does not know the existence of other agents and generates samples independently. Although the agents share the parameter (θ in $p_\theta(a_t|s_t)$), their trajectories are different from each other because they sample an action from the distribution $p_\theta(a_t|s_t)$. We can visualize the above by the following:

Figure 14: Asynchronous agents in A3C algorithm. Different agent (agent ①, ②, and ③) can have different trajectory. It is possible to have different batch size for each agent.

We call each agent as *worker*. Each worker gives either samples or calculated gradient to the center, and the center gives updated parameters to each worker back (See Figure 14). Each worker produces samples and the samples are not correlated. This helps to reduce correlation between samples. We will talk more about the update rule of *workers* later in this section.

13.4.2 *n-step* TD

A2C algorithm used *1-step* TD when updating both actor and critic. A3C algorithm uses *n-step* TD (See section 9 for *n-step* TD). To derive *n-step* in A3C algorithm, we will begin from A2C algorithm. Let's see how *2-step* in A3C algorithm can be formulated. By doing so, we will be able to better understand *n-step* in A3C algorithm by expanding the idea of *2-step* A3C algorithm. The definition of Q could be expressed by the following.

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}:a_\infty} G_t \cdot p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)\, ds_{t+1} : a_\infty \tag{93}$$

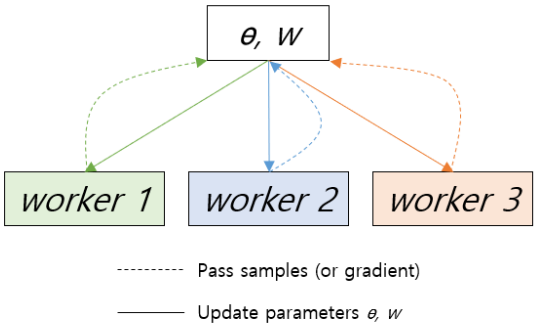

Figure 15: Workers in A3C algorithm

By rewriting $p(s_{t+1}, a_{t+1}, \dots | s_t, a_t)$

$$p(s_{t+1}, a_{t+1}, \dots | s_t, a_t) = p(a_{t+2}, \cdots | s_t, a_t, s_{t+1}, a_{t+1}, s_{t+2}) \cdot p(s_{t+1}, a_{t+1}, s_{t+2} | s_t, a_t) \tag{94}$$

$$= p(a_{t+2}, \cdots | s_t, a_t, s_{t+1}, a_{t+1}, s_{t+2}) \cdot p(s_{t+2} | s_t, a_t, s_{t+1}, a_{t+1}) \cdot p(s_{t+1}, a_{t+1} | s_t, a_t) \tag{95}$$

$$= p(a_{t+2}, \cdots | s_t, a_t, s_{t+1}, a_{t+1}, s_{t+2}) \cdot p(s_{t+2} | s_t, a_t, s_{t+1}, a_{t+1}) \cdot p(a_{t+1} | s_t, a_t, s_{t+1}) \cdot p(s_{t+1} | s_t, a_t) \tag{96}$$

$$= p(a_{t+2}, \cdots | s_{t+2}) \cdot p(s_{t+2} | s_{t+1}, a_{t+1}) \cdot p(a_{t+1} | s_{t+1}) \cdot p(s_{t+1} | s_t, a_t) \tag{97}$$

and plugging in the last equation 97 back to the definition of Q, we get the following:

$$Q(s_t, a_t) \triangleq \int_{s_{t+1}:a_\infty} (R_t + \gamma R_{t+1} + \gamma^2 G_{t+2}) \cdot p(a_{t+2}, \cdots | s_{t+2}) \cdot p(s_{t+2} | s_{t+1}, a_{t+1}) \cdot p(a_{t+1} | s_{t+1}) \cdot p(s_{t+1} | s_t, a_t) \, ds_{t+1} : a_\infty \tag{98}$$

$$= E_{s_{t+1}, a_{t+1}, s_{t+2}}[R_t + \gamma R_{t+1} + \gamma^2 V(s_{t+2}) | s_t, a_t] \tag{99}$$

In *2-step* A3C algorithm, the gradient of the objective function is (the following equation is imported from A2C algorithm)

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t, a_t, s_{t+1}} \nabla_\theta \ln p_\theta(a_t | s_t) \cdot (Q - V) \cdot p(s_{t+1} | s_t, a_t) \cdot p_\theta(a_t | s_t) \cdot p(s_t | s_{t-1}, a_{t-1}) \, ds_t, a_t, s_{t+1} \tag{100}$$

where

$$\textit{2-step TD: } Q - V = E_{s_{t+1}, a_{t+1}, s_{t+2}}[R_t + \gamma R_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) | s_t, a_t] \tag{101}$$

The same rules applies to *n-step* TD. For example,

$$\textit{3-step TD: } Q - V = E_{s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, s_{t+3}}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + V(s_{t+3}) - V(s_t) | s_t, a_t] \tag{102}$$

$$\textit{4-step TD: } Q - V = E_{s_{t+1}, a_{t+1}, s_{t+2}, a_{t+2}, s_{t+3}, a_{t+3}, s_{t+4}}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \gamma^4 V(s_{t+4}) - V(s_t) | s_t, a_t] \tag{103}$$

$$\vdots \tag{104}$$

Let's take a simple example when $n = 3$ in A3C algorithm. The algorithm uses *1-step*, *2-step*, and *3-step* TD to update both actor and critic (See Figure 16).

$$\textit{3-step TD: } R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 V_3 - V_0 = A_0 \tag{105}$$

$$\textit{2-step TD: } R_1 + \gamma R_2 + \gamma^2 V_3 - V_1 = A_1 \tag{106}$$

$$\textit{1-step TD: } R_2 + \gamma V_3 - V_2 = A_2 \tag{107}$$

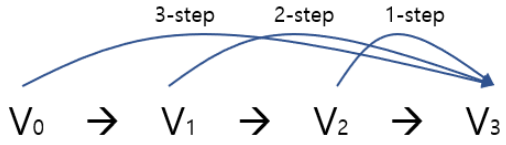

Figure 16: *3-step* TD in A3C algorithm. When we are at $t = 2$, we can acquire V_3

We can define our new actor update function by using advantage function A. n is decided by N in A3C algorithm. When $N = 3$, the objective function and actor update can be formulated by the following, and A_i follows the previous equations 105, 106, 107.

$$\nabla_\theta J_\theta \approx \sum_{t=0}^{\infty} \int_{s_t, a_t, s_{t+1}} \nabla_\theta \ln p_\theta(a_t|s_t) \cdot (Q - V) \cdot p(s_{t+1}|s_t, a_t) \cdot p_\theta(a_t|s_t) \cdot p(s_t|s_{t-1}, a_{t-1})\, ds_t, a_t, s_{t+1} \tag{108}$$

$$\textit{Actor update: } \theta \leftarrow \theta + \alpha \cdot \nabla_\theta \sum_{i=t-2}^{t} \ln p_\theta(a_i|s_i) \cdot A_i \tag{109}$$

It is known that when N is large, it is high variance and low bias. In contrast, when N is small, it is low variance and high bias.

13.4.3 Entropy

The algorithm introduces entropy for actor and this does not apply to critic. This enables the actor to take various actions and allows exploration. The idea is straightforward. We add entropy of policy in actor update process.

$$\textit{Actor update: } \theta \leftarrow \theta + \alpha \cdot \nabla_\theta \sum_{i=t-N+1}^{t} (\ln p_\theta(a_i|s_i) \cdot A_i + \lambda \cdot H_i(p_\theta(a_i|s_i))) \tag{110}$$

λ adjusts the effect of entropy. Assuming the policy is Gaussian distribution, the standard deviation of the policy is proportional to entropy. That is, the larger the standard deviation, the more diverse the actor can perform (See (B) for more details). We can visualize this by the following diagram. Actor update function now considers two terms:

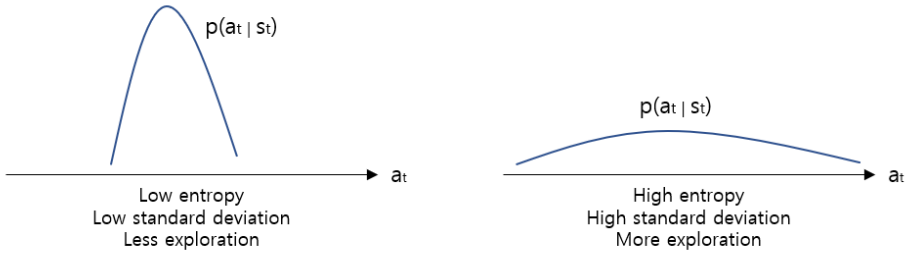

Figure 17: Relationship between entropy, standard deviation, and exploration

$\ln p_\theta(a_t|s_t) \cdot A_i$ and $H_i(p_\theta(a_i|s_i))$. When λ is large, the actor focuses on exploration. When it is small, the actor focuses on the first term.

13.4.4 Final algorithm

Asynchronous Advantage Actor-Critic algorithm

Algorithm parameters: actor(policy) parameter θ and critic parameter w;
Algorithm hyper-parameters: maximum step T, discount factor γ, sample size N, learning rate for actor α, and critic β;

Initialize θ and w. Choose maximum step T, discount factor γ, sample size N, learning rate α, and β;
For an asynchronous worker,
repeat
  Collect N samples (single sample: s_t, a_t, R_t, s_{t+1});
  Actor update: $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \sum_{i=t-N+1}^{t} (\ln p_\theta(a_i|s_i) \cdot A_i + \lambda \cdot H_i(p_\theta(a_i|s_i)))$;
  Critic update: $w \leftarrow w - \beta \cdot \nabla_w \sum_{i=t-N+1}^{t} {A_i}^2$;
  Clear the batch (Remove N samples)
until *we find satisfactory parameters:* θ *and* w;

Algorithm 9: Asynchronous Advantage Actor-Critic algorithm

13.4.5 Update rule of *workers*

There are two ways to update parameters θ and w. The first option is called **gradient parallelism**. In case of gradient parallelism, each worker calculates gradient of actor and critic using N samples. Afterwards, the worker passes the gradients to the center (global network). The global network then updates its own θ and w. The updated θ and w get copied into each worker back. This method advantageous when expanding the number of workers because the gradients are calculated by the workers.

The second option is called **data parallelism**. In case of data parallelism, the global network calculates gradients when N samples are given from each worker. A single worker generate N samples and passes the generated samples to the global network. The worker is to just generate samples. Everything else is the global network's job. The global network updates its gradients and copies its gradients into each worker back. The global network calculates the gradients and updates the parameters.

14 Additional Details

(A) Simplification of the policy gradient equation:

We will show how the equation 61 could be simplified into 62

$$(R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots) \cdot (\nabla_\theta \sum_{t=0}^{\infty} \ln p_\theta(a_t|s_t)) \tag{111}$$

We can expand the above equation to the following:

$$(R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots) \cdot (\nabla_\theta \ln p_\theta(a_0|s_0) + \nabla_\theta \ln p_\theta(a_1|s_1) + \nabla_\theta \ln p_\theta(a_2|s_2) + \cdots) \tag{112}$$

It turns out that when we multiply the reward and policy, and if the index of the reward is smaller than the index of the policy, then the multiplied term becomes 0. That is, the following holds when p is smaller than q.

$$\int_\tau \gamma^p R_p \cdot \nabla_\theta \ln p_\theta(a_q|s_q) \cdot p_\theta(\tau)\, d\tau = 0 \tag{113}$$

For instance,

$$\int_\tau R_0 \cdot \nabla_\theta \ln p_\theta(a_1|s_1) \cdot p_\theta(\tau)\, d\tau \tag{114}$$

$$= \int_\tau R_0 \cdot \nabla_\theta \ln p_\theta(a_1|s_1) \cdot p_\theta(a_1|\tau_{-a_1})\, p(\tau_{-a_1})\, d\tau \tag{115}$$

$$= \int_\tau R_0 \cdot \nabla_\theta \ln p_\theta(a_1|s_1) \cdot p_\theta(a_1|s_1)\, p(\tau_{-a_1})\, d\tau \tag{116}$$

$$= \int_{\tau_{-a_1}} R_0 \int_{a_1} \nabla_\theta \ln p_\theta(a_1|s_1)\, p_\theta(a_1|s_1)\, da_1\, p(\tau_{-a_1})\, d\tau_{-a_1} \tag{117}$$

$$= \int_{\tau_{-a_1}} R_0 \int_{a_1} \frac{\nabla_\theta p_\theta(a_1|s_1)}{p_\theta(a_1|s_1)}\, p_\theta(a_1|s_1)\, da_1\, p(\tau_{-a_1})\, d\tau_{-a_1} \tag{118}$$

$$= \int_{\tau_{-a_1}} R_0 \int_{a_1} \nabla_\theta p_\theta(a_1|s_1)\, da_1\, p(\tau_{-a_1})\, d\tau_{-a_1} \tag{119}$$

$$= \int_{\tau_{-a_1}} R_0\, (\nabla_\theta \int_{a_1} p_\theta(a_1|s_1)\, da_1)\, p(\tau_{-a_1})\, d\tau_{-a_1} \tag{120}$$

$$= \int_{\tau_{-a_1}} R_0\, (\nabla_\theta 1)\, p(\tau_{-a_1})\, d\tau_{-a_1} \tag{121}$$

$$= \int_{\tau_{-a_1}} R_0 \cdot 0 \cdot\ p(\tau_{-a_1})\, d\tau_{-a_1} \tag{122}$$

$$= 0 \tag{123}$$

Therefore, we can rewrite the equation 111 to the following:

$$\sum_{t=0}^{\infty} (\nabla_\theta \ln p_\theta(a_t|s_t) \cdot (\sum_{k=t}^{\infty} \gamma^k \cdot R_k)) \tag{124}$$

(B) Entropy and Gaussian distribution:

We will show why standard deviation of the policy is proportional to entropy. Entropy H of $p(x)$ can be defined by the following assuming that $p(x)$ follows Gaussian distribution.

$$H(p(x)) \triangleq -\int p(x) \ln p(x) dx \tag{125}$$

$$where\ p(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{126}$$

$$H(p(x)) = -\int p(x) \ln p(x) dx \tag{127}$$

$$= -\int p(x) \ln \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) dx \tag{128}$$

$$= \int p(x) \cdot \ln \sqrt{2\pi\sigma^2} dx + \int p(x) \cdot \frac{(x-\mu)^2}{2\sigma^2} dx \tag{129}$$

$$= \frac{1}{2}(1 + \ln 2\pi\sigma^2) \tag{130}$$

15 Check your understanding

1. How many polices there are when there are 7 discrete states and 2 actions per state?
   $\rightarrow 2^7$. Generally, the number of polices is $|A|^{|S|}$.

2. Is the optimal policy for a MDP always unique?
   $\rightarrow$ No.

3.

References

[1] Raghavendra Selvan. "Bayesian tracking of multiple point targets using Expectation Maximization". PhD thesis. July 2015.

[2] *Understanding the role of the discount factor in reinforcement learning*. `https://stats.stackexchange.com/questions/221402/understanding-the-role-of-the-discount-factor-in-reinforcement-learning`. 2016.