## 1.POLICY GRADIENTS

a) For Policy Gradient, our goal is to get the derivative of $V(s;\theta)$ and update $\theta$ with the derivative. The steps are:
1. Sample a trajectory/rollout $\tau = (s_0, a_0, .., s_L)$. Here, since we consider the so-called bandit setting where the trajectory does not matter, we will only consider one action, which can be regarded as $\tau = a_i$
2. Compute $R(\tau) = \sum_{t=0}^{L-1} -r(s_t, a_t)$. Since we doesn't consider the trajectory, and we know that different actions $a_i$ give rise to different rewards $R_i$, we can get that $R(\tau) = R_i$
3. $\theta \leftarrow \theta - \eta R(\tau) \frac{\partial}{\partial \theta} log\pi(\tau)$ , which means $\triangle\theta = \eta R(\tau) \frac{\partial}{\partial \theta} log\pi(\tau) = \eta R_i \frac{\partial}{\partial \theta} log\pi(a_i)$

As we known, we have

$$\pi(a_i) = softmax(\theta_i), i = 1, \ldots, k$$
$$log\pi(a_i) = log(softmax(\theta_i))$$

Thus, our goal now is to get the derivative of the log(Softmax)

We assume $\theta$ as a vector containing the action scores for a single state, that's the output of the network. Thus $\theta_i$ is an element for a certain action $i$ in all actions $k$.
We can rewrite the softmax output as

$$\pi(a_i) = softmax(\theta_i) = p_i = \frac{e^{\theta_i}}{\sum_k e^{\theta_k}}$$

and the log-likelihood as

$$L_i = log(\pi(a_i)) = log(p_{y_i})$$

From chain rule we have

$$\frac{\partial L_i}{\partial \theta_i} = \frac{\partial L_i}{\partial p_i} \frac{\partial p_i}{\partial \theta_i}$$

For the first part,

$$\frac{\partial L_i}{\partial p_i} = \frac{1}{p_i}$$

For the second part,we have to recall the quotient rule for derivatives, let the derivative be represented by the operator $\mathbf{D}$,

$$\frac{f(x)}{g(x)} = \frac{g(x)\mathbf{D}f(x) - f(x)\mathbf{D}g(x)}{g(x)^2}$$

Here, set $\sum_k e^{\theta_k} = \Sigma$, we get,

$$\frac{\partial p_i}{\partial \theta_i} = \frac{\partial}{\partial \theta_i}\left(\frac{e^{\theta_i}}{\sum_k e^{\theta_k}}\right) = \frac{\Sigma \mathbf{D} e^{\theta_i} - e^{\theta_i}\mathbf{D}\Sigma}{\Sigma^2} = \frac{e^{\theta_i}(\Sigma - e^{\theta_i})}{\Sigma^2}$$

Here, $\mathbf{D}\sum = e^{\theta_i}$ because if we take the input array $\theta$ in the softmax function, we're always "looking" or we're always taking the derivative of the i-th element. In this case, the derivative with respect to the $i^{th}$ element will always be 0 in those elements that are non-$i$, but $e^{\theta_i} at i$.

$$\frac{\partial p_i}{\partial f_i} = \frac{e^{\theta_i}(\Sigma - e^{\theta_i})}{\Sigma^2} = \frac{e^{\theta_i}}{\Sigma}\frac{\Sigma - e^{\theta_i}}{\Sigma} = p_i * (1 - p_i)$$

Thus, we know the derivative of the log(Softmax)

$$\frac{\partial L_i}{\partial \theta_i} = \frac{\partial L_i}{\partial p_i}\frac{\partial p_i}{\partial \theta_i} = \frac{1}{p_i}(p_i * (1 - p_i)) = (1 - p_i)$$

It means that

$$\frac{\partial}{\partial \theta} log\pi(a_i) = 1 - \pi(a_i)$$

Thus, we have

$$\triangle\theta = \eta R_i \frac{\partial}{\partial\theta} log\pi(a_i) = \eta R_i(1-\pi(a_i))$$

b) As we known from above, we can say that $\triangle\theta$ is proportional to the $E(reward - loss)$, which is the expected reward loss if action $a_i$ doesn't happen. Here the $1-\pi(a_i)$ represents the predicted possibility of action $a_i$ doesn't be chose, and $R_i$ is the reward that we should received if we choose action $a_i$.
Also, reward $R$ is proportional to the prediction loss. When $R$ converges, the loss also converges, then once loss converges to a limited range, then the network will make little change on $\theta$ and then finish training.
For example, if choosing action $a_i$ will actually result in big reward $R_i$, which means $a_i$ is a good move, however the network predicts that the possibility that we should choose action $a_i$ is small, which is $1-\pi(a_i)$ is large, then the $\triangle\theta$ would be large and parameter $\theta$ still need to be trained. And after many rounds, when agent reach the same state, the network now predicts that the possibility that we should choose action $a_i$ is large, which is $1-\pi(a_i)$ is small, then the $\triangle\theta$ would be small and parameter $\theta$ will be slightly adjusted.

## 2.DESIGNING REWARDS IN Q-LEARNING

a) We declare a reward of +2 for reaching the goal, -1 for running into a monster, and 0 for every other move.
b) We declare a reward of +1.5 for reaching the goal, -1.5 for running into a monster, and -0.5 for every other move.

As we know, the formula of discounted return is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

if a constant offset $O$ is added to all rewards, then the discounted return becomes:

$$G_t = (R_{t+1} - O) + \gamma(R_{t+2} - O) + \gamma^2(R_{t+3} - O) + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. - \sum_{k=0}^{\infty} \gamma^k O$$

As we can see from the second reward formula, if we add an offset to rewards, the sum of the offset to current reward would be $\sum_{k=0}^{\infty} \gamma^k O$. Once the number of total movement increases, which means the $k$ increases, the total offset would increase, resulting in less reward. Therefore, we can know the importance of the negative rewards on every other move: it incentivizes the agent to get done as quickly as possible because it's constantly losing points when it play this game. If the agent want to maximize the rewards, it not only need to avoid monster and reach the goal, but also need to move as fewer times as possible.

Let's look into the expected discounted return $R_{(\tau)}$ of two strategy. Assume that $r_1$ is the reward for the agent reaching the goal, $r_2$ is the reward for the agent running into a monster, and $r_3$ is the reward for every other move. $s_{t1}$ is the state that after agent choose action $a_{t1}$ resulting in reaching the goal, $s_{t2}$ is the state that after agent choose action $a_{t2}$ resulting in running into a monster, $s_{t3}$ is the state that after agent choose action $a_{t3}$ resulting in an empty move.
The agent should try to minimize the negative reward:

$$minimize R(\tau) = \sum -r(s_t, a_t) = \sum -r_1(s_{t1}, a_{t1}) + \sum -r_2(s_{t2}, a_{t2}) + \sum -r_3(s_{t3}, a_{t3})$$

For strategy a, $r_3(s_{t3}, a_{t3}) = 0$, which means that for the agent, the total number of empty move has no impact on its discounted return. Thus, agent might make more extra moves.
For strategy b, $r_3(s_{t3}, a_{t3}) = -0.5$, which means that for the agent, the total number of empty move has great impact on its discounted return and agent will try to make as less empty move as possible and reach to the goal faster, comparing to strategy a.

Thus, we know that strategy b is better than strategy a.

For strategy a), it is actually a sparse reward problem. Under this rewards strategy, agent only get positive reward when it reach the goal and negative reward when it run into a monster. And there is no feedback in other states(or 'empty' move). Thus, there is no efficient feedback to lead the agent to move in the correct direction, so it is difficult for the agent to reach the goal only by random exploration. Therefore, in this case, the agent would take more time to explore and find a way to reach the goal. There might have the case that since there is no penalty for every extra 'empty' move, the agent might spin in the same place and cannot figure out the path to the goal, when the agent is far away from the goal and the monster. Or in other words, the agent would be stuck in the local optimal.

For strategy b), there is negative reward for every other move, which will encourage the agent to reach the goal without running into monster as soon as possible so as to avoid accumulating penalties for every other move it makes. Thus, it will avoid the case that mentioned above in strategy a.

In conclusion, strategy b) is better as it sets negative reward for every other move.