1. What are the ways of finding action value estimates (Qt(a))?

Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g.: suppose by the *t*-th play, action a had been chosen k_a times, producing rewards then $r_1, r_2, \ldots, r_{k_a}$,

"sample average" $Q_t(a) = \dfrac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$ **OR** $Q_t(a) = \dfrac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)}$

$$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

Sample Average Method

- If k_a = 0, then we define $Q_t(a)$ instead as some default value, such as $Q_1(a) = 0$.
- As $k_a \to \infty$, by the law of large numbers, $Q_t(a)$ converges to $q^*(a)$.
- We call this the sample-average method for estimating action values because each estimate is a simple average of the sample of relevant rewards.

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}},$$

Greedy Action Selection

- The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select at step t one of the greedy actions.
- This greedy action selection method can be written as

$$a_t = a_t^* = \operatorname{argmax}_a Q_t(a)$$

- where argmax_a denotes the value of a, at which the expression that follows is maximized

ε-Greedy Action Selection

The basic idea is to select the action with the highest estimated value (the greedy action) with probability (1 - epsilon) and to select a random action with probability epsilon.

- ε-Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \\ \text{random action with probability } \varepsilon \end{cases}$$

. . . the simplest way to balance exploration and exploitation

2. What is Q estimate update rule? Derive the same.

Incremental Implementation

The average of the first k rewards is $Q_t(a) = \dfrac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$

Can we do this incrementally (without storing all the rewards)?

$$\begin{aligned} Q_{k+1} &= \frac{1}{k}\sum_{i=1}^{k} R_i \\ &= \frac{1}{k}\left(R_k + \sum_{i=1}^{k-1} R_i\right) \\ &= \frac{1}{k}\left(R_k + (k-1)Q_k + Q_k - Q_k\right) \\ &= \frac{1}{k}\left(R_k + kQ_k - Q_k\right) \\ &= Q_k + \frac{1}{k}\left[R_k - Q_k\right], \end{aligned}$$

$$\begin{aligned} Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\ &= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\ &= \frac{1}{n}\left(R_n + (n-1)Q_n\right) \\ &= \frac{1}{n}\left(R_n + nQ_n - Q_n\right) \\ &= Q_n + \frac{1}{n}\left[R_n - Q_n\right], \end{aligned}$$

This is a common form for update rules:

NewEstimate = *OldEstimate* + *StepSize*[*Target* – *OldEstimate*]

3. How to handle non-stationary rewards in the Q estimate update rule?

Tracking a Nonstationary Problem

- In case of Nonstationary, it makes sense to weight recent rewards more heavily than long-past ones.
- One of the most popular ways of doing this is to use a constant step-size parameter.
- For example, the incremental update rule for updating an average Q_k of the k − 1 past rewards is modified to be :

$$Q_{k+1} = Q_k + \alpha\left[R_k - Q_k\right]$$

where the step-size parameter $\alpha \in (0, 1]$ is constant. This results in Q_{k+1} being a weighted average of past rewards and the initial estimate Q_1

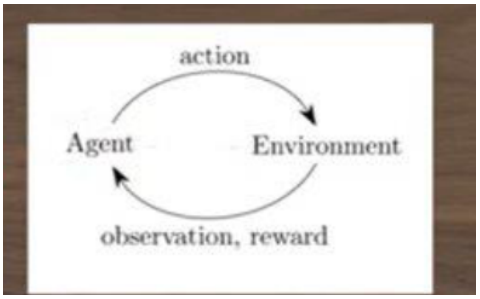
4. Derive exponential/recency-weighted average update rule for Q estimates.

$$\begin{aligned}
Q_{k+1} &= Q_k + \alpha\left[R_k - Q_k\right] \\
&= \alpha R_k + (1-\alpha)Q_k \\
&= \alpha R_k + (1-\alpha)\left[\alpha R_{k-1} + (1-\alpha)Q_{k-1}\right] \\
&= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 Q_{k-1} \\
&= \alpha R_k + (1-\alpha)\alpha R_{k-1} + (1-\alpha)^2 \alpha R_{k-2} + \\
&\qquad \cdots + (1-\alpha)^{k-1}\alpha R_1 + (1-\alpha)^k Q_1 \\
&= (1-\alpha)^k Q_1 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} R_i.
\end{aligned}$$

- sometimes called an exponential, recency-weighted average

Where, Q_t → action-value function at time step t
R → reward
Alpha → step-size parameter

5. What is k-armed Bandit problem? What are the different algorithms to solve k-armed Bandit problem?



Here, You are faced repeatedly with a choice among k different options, or actions. After each choice, you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or time steps. This is the original form of the k-armed bandit problem, so named by analogy to a slot machine, or "one-armed bandit," except that it has k levers instead of one. Each action selection is like a play of one of the slot machine's levers, and the rewards are the payoffs for hitting the jackpot

Different Algorithms to solve k-Armed Bandit Problem:

1. Epsilon-Greedy
2. Upper Confidence Bound (UCB)
3. Thompson Sampling
4. Optimistic Initial Values

6. Write Simple Bandit Algorithm.

The Simple Bandit Algorithm works as follows:

1. Initialize the estimated value of each action to 0.
2. N(a) = 0 i.e., no. of times a is chosen
3. For each time step:
   a. Choose an action with the highest estimated value with probability (1 - epsilon), or choose a random action with probability epsilon. Epsilon is a small positive constant, typically set to 0.1 or less, that controls the exploration-exploitation trade-off.

$$A \leftarrow \begin{cases} \text{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

   b. Take the chosen action and observe the reward.
   c. Increment N(a) by 1.
   d. Update the estimated value of the chosen action as follows:

$$Q(a) = Q(a) + (1/N(a)) * (R - Q(a))$$

   where Q(a) is the estimated value of action a, R is the observed reward

The Simple Bandit Algorithm is a simple yet effective approach to solving the multi-armed bandit problem. By balancing exploration and exploitation, it is able to learn the true reward distribution of the actions over time and converge to the optimal action

7. Explain Greedy action selection algorithm

Greedy Action Selection

- The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select at step t one of the greedy actions.
- This greedy action selection method can be written as

$$a_t = a_t^* = \underset{a}{\operatorname{argmax}} Q_t(a)$$

- where $\arg\max_a$ denotes the value of a, at which the expression that follows is maximized

For example, in a grid-world environment, if the estimated Q-values for a state are 5 (up), 3 (down), 2 (left), and 4 (right), the agent will choose the "up" action because it has the highest Q-value. However, this approach does not consider exploration and may lead to suboptimal actions if the Q-value estimates are inaccurate.

8. Explain e-Greedy action selection algorithm

The basic idea is to select the action with the highest estimated value (the greedy action) with probability (1 - epsilon) and to select a random action with probability epsilon.

- ε-Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability} \quad 1-\varepsilon \\ \text{random action with probability} & \varepsilon \end{cases}$$

A real-life example of an e-greedy action selection algorithm can be seen in online advertising. When a user visits a website, the website can choose to show an ad that has the highest estimated click-through rate (CTR) with a certain probability (1-epsilon). On the other hand, it can show a random ad with probability epsilon. This approach helps to maximize revenue for the website while still exploring different ads to see if they have higher CTRs.

9. Compare e -Greedy and Greedy

The advantage of ε-greedy over greedy methods depends on the task. For example, suppose the reward variance had been larger, say 10. With noisier rewards, it takes more exploration to find the optimal action, and ε-greedy methods should fare even better relative to the greedy method.

On the other hand, if the reward variances were zero, then the greedy method would know the true value of each action after trying it once. In this case, the greedy method might actually perform best because it would soon find the optimal action and then never explore.

10. How to solve Exploration-exploitation dilemma using Optimistic Initial Values?

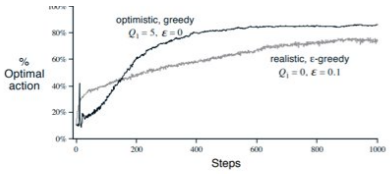
How Optimistic Initial Values works?

- This method is like a little hack that encourages the agent to explore in the beginning.
- We start by assuming all actions are optimum. We do this by assigning all actions an initial value larger than the mean reward we expect to receive after pulling any arm.
- As a result of this, the first estimate of any action will always be less than this optimum initial value. And, even if we're greedy, we'll not visit this action again and rather select another unvisited action.
- This approach encourages the agent to visit all actions multiple times, resulting in early improvement in estimated action values.

Optimistic Initial Values

- The downside is that the initial estimates become, in effect, a set of parameters that must be picked by the user, if only to set them all to zero.
- The upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected.
- Initial action values can also be used as a simple way of encouraging exploration

Optimistic Initial Values



- Figure shows the performance on the 10-armed bandit testbed of a greedy method using Q1(a) = +5, for all a.
- For comparison, also shown is an ε-greedy method with Q1(a) = 0. Initially, the optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time.
- We call this technique for encouraging exploration optimistic initial values

11. Define confidence and uncertainty in UCB algorithm.

Confidence refers to the agent's belief or level of certainty about the estimated value of a state-action pair. A higher level of confidence indicates that the agent has more trust in its estimated values.

Uncertainty, on the other hand, represents the lack of knowledge or ambiguity about the true values of state-action pairs. Higher uncertainty suggests that the agent is less sure about the actual values and needs to gather more information to improve its knowledge.

12. Explain UCB action selection algorithm

Upper-Confidence-Bound Action Selection

- Considering the working of Greedy and ε-Greedy algorithm, it would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.
- One effective way of doing this is to select actions as

$$A_t = \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- where ln*t* denotes the natural logarithm of t (the number that e ≈ 2.71828 would have to be raised to in order to equal t), and the number c > 0 controls the degree of exploration. If Nt(a) = 0, then a is considered to be a maximizing action.

Upper-Confidence-Bound Action Selection

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- The idea of this upper confidence bound (UCB) action selection is that the square-root term is a measure of the uncertainty or variance in the estimate of a's value. The quantity being max'ed over is thus a sort of upper bound on the possible true value of action a, with the c parameter determining the confidence level.
- Each time a is selected the uncertainty is presumably reduced; $N_t(a)$ is incremented and, as it appears in the denominator of the uncertainty term, the term is decreased.

UCB Action Selection

- As shown, UCB generally performs better than ε-greedy action selection, except in the first n plays, when it selects randomly among the as-yet unplayed actions.
- UCB with c = 1 would perform even better but would not show the prominent spike in performance on the 11th play.

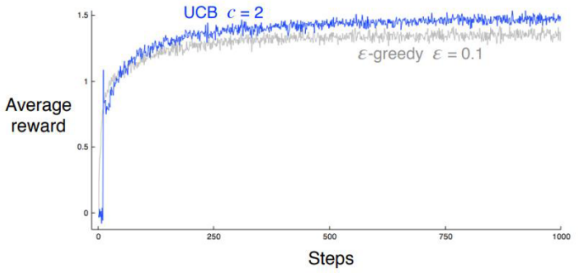


Figure: Average performance of UCB action selection on the 10-armed testbed.

- UCB will often perform well, as shown here, but is more difficult than ε-greedy to extend beyond bandits to the more general reinforcement learning.
- One difficulty is in dealing with nonstationary problems; something more complex than the methods discussed so far would be needed.
- Another difficulty is dealing with large state spaces

13. Compare UCB action selection and e –Greedy.

UCB is more exploratory because it selects actions based on an upper confidence bound that encourages the agent to try out less-frequently-selected actions that may have higher rewards.

E-greedy is more exploitative because it mostly selects the action with the highest estimated value, but it still has some exploration through the occasional selection of random action.

UCB assumes that the rewards are normally distributed, while e-greedy does not make any assumptions about the distribution of rewards.

14. Write the Thompson sampling algorithm (for beta priors)

Algorithm 1: Thompson Sampling using Beta priors

For each arm $i = 1, \ldots, N$ set $S_i = 0, F_i = 0$.
foreach $t = 1, 2, \ldots,$ **do**
    For each arm $i = 1, \ldots, N$, sample $\theta_i(t)$ from the Beta$(S_i + 1, F_i + 1)$ distribution.
    Play arm $i(t) := \arg\max_i \theta_i(t)$ and observe reward r_t.
    If $r_t = 1$, then $S_{i(t)} = S_{i(t)} + 1$, else $F_{i(t)} = F_{i(t)} + 1$.
end

15. Define reward and Penalty. What is the significance of introducing a Penalty?

Reward: The feedback signal received by the agent indicating how well it did in the environment
Penalty: The penalty term encourages the agent to explore and learn about the environment by reducing the reward for actions that have been taken frequently in the past.

16. What is LRP algorithm?

- The linear reward penalty algorithm is a reinforcement learning algorithm that is used to train agents in tasks where the reward function is sparse or misleading.
- The penalty term encourages the agent to explore and learn about the environment by reducing the reward for actions that have been taken frequently in the past.
- This helps to avoid getting stuck in suboptimal policies and promotes the exploration of new and potentially better policies.

- The LRP algorithm is a simple and effective way to enhance exploration in reinforcement learning and has been shown to work well in a variety of tasks, including robotics, game playing, and natural language processing.

LRP Algorithm:

1. Initialize the agent's policy and value function
2. Run the agent in an environment and collect a set of trajectories.
3. For each trajectory, compute the discounted return G for each time step t, defined as the sum of rewards obtained at time step t to the end of the trajectory, multiplied by the discount factor γ
4. For each time step t, compute the reward R'(s_t, a_t) as follows:

   $$R'(s\_t, a\_t) = R(s\_t, a\_t) - \lambda\sqrt{n(s\_t, a\_t)}$$

   Where, R(s_t, a_t) is the original reward function, λ is a hyperparameter that controls the penalty strength, n(s_t, a_t) is the number of times the agent has visited state s_t and taken action a_t
5. Update the agent's policy and value function using the modified reward function R'(s_t, a_t) and the discount return G.
6. Repeat steps 2-5 for a certain number of iterations or until the agent converges to an optimal policy.
7. For each time step t, compute the reward R'(s_t, a_t) as follows:

   $$R'(s\_t, a\_t) = R(s\_t, a\_t) - \lambda\sqrt{n(s\_t, a\_t)}$$

   Where, R(s_t, a_t) is the original reward function, λ is a hyperparameter that controls the penalty strength, n(s_t, a_t) is the number of times the agent has visited state s_t and taken action a_t
8. Update the agent's policy and value function using the modified reward function R'(s_t, a_t) and the discount return G.
9. Repeat steps 2-5 for a certain number of iterations or until the agent converges to an optimal policy.

17. What are the important components of REINFORCE algorithm?

The algorithm needs 3 components:

1. A parametrized policy
2. An objective to be maximized
3. A method for updating the policy parameters

1. Policy

- A policy π is a function, mapping states to action probabilities, which is used to sample an action $a \sim \pi(s)$. In REINFORCE, an agent learns a policy and uses this to act in an environment.
- A good policy is one which maximizes the cumulative discounted rewards. The key idea of the algorithm is to learn a good policy, and this means doing function

approximation. Neural networks are powerful and flexible function approximators, so we can represent a policy using a deep neural network consisting of learnable parameters θ. This is often referred to as a policy network πθ. We say that the policy is parametrized by θ.

- Each specific set of values of the parameters of the policy network represents a particular policy. To see why, consider θ1=θ2. For any given state s, different policy networks may output different sets of action probabilities, that is $\pi_{\theta 1}(s) \neq \pi_{\theta 2}(s)$. The mappings from states to action probabilities are different so we say that πθ1 and πθ2 are different policies. A single neural network is therefore capable of representing many different policies.
- Formulated in this way, the process of learning a good policy corresponds to searching for a good set of values for θ. For this reason, it is important that the policy network is differentiable.

The Objective Function

- An agent acting in an environment generates a trajectory,
- which contains a sequence of rewards along with the states and actions. A trajectory is denoted as : τ = s0, a0, r0, . . . , sT , aT , rT .
- The return of a trajectory Rt(τ) is defined as a discounted sum of rewards from time step t to the end of a trajectory.

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$$

- The objective is the expected return over all complete trajectories generated by an agent.

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

Policy Gradient

- The policy provides a way for an agent to act, and the objective provides a target to maximize. The final component of the algorithm is the policy gradient.
- Formally, we say a policy gradient algorithm solves the following problem

$$\max_{\theta} J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

- To maximize the objective, we perform gradient ascent on the policy parameters θ.
- To improve on the objective, compute the gradient and use it to update the parameters as

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

- The term $\pi_\theta(a_t \mid s_t)$ is the probability of the action taken by the agent at time step t.
- The action is sampled from the policy, $a_t \sim \pi_\theta(s_t)$.
- The right-hand side of the equation states that the gradient of the log probability of the action with respect to θ is multiplied by return $R_t(\tau)$.

- The policy gradient is the mechanism by which action probabilities produced by the policy are changed. If the return $R_t(\tau) > 0$, then the probability of the action $\pi_\theta(a_t \mid s_t)$ is increased; conversely, if the return $R_t(\tau) < 0$, then the probability of the action $\pi_\theta(a_t \mid s_t)$ is decreased.
- Over the course of many updates $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ the policy will learn to produce actions which result in high $R_t(\tau)$.

18. Explain the objective function in REINFORCE algorithm?

The Objective Function

- An agent acting in an environment generates a trajectory,
- which contains a sequence of rewards along with the states and actions. A trajectory is denoted as : τ = s0, a0, r0, . . . , sT , aT , rT .
- The return of a trajectory Rt(τ) is defined as a discounted sum of rewards from time step t to the end of a trajectory.

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$$

- The objective is the expected return over all complete trajectories generated by an agent.

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

19. Define Policy Gradient and derive the same

Policy Gradient

- The policy provides a way for an agent to act, and the objective provides a target to maximize. The final component of the algorithm is the policy gradient.
- Formally, we say a policy gradient algorithm solves the following problem

$$\max_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

- To maximize the objective, we perform gradient ascent on the policy parameters θ.
- To improve on the objective, compute the gradient and use it to update the parameters as

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

- The term $\pi_\theta(a_t \mid s_t)$ is the probability of the action taken by the agent at time step t.
- The action is sampled from the policy, $a_t \sim \pi_\theta(s_t)$.
- The right-hand side of the equation states that the gradient of the log probability of the action with respect to θ is multiplied by return $R_t(\tau)$.

- The policy gradient is the mechanism by which action probabilities produced by the policy are changed. If the return $R_t(\tau) > 0$, then the probability of the action $\pi_\theta(a_t \mid s_t)$ is increased; conversely, if the return $R_t(\tau) < 0$, then the probability of the action $\pi_\theta(a_t \mid s_t)$ is decreased.
- Over the course of many updates $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ the policy will learn to produce actions which result in high $R_t(\tau)$.

Policy Gradient Derivation

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

- This presents a problem because we cannot differentiate $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ with respect to θ.
- The rewards r_t are generated by an unknown reward function $R(s_t, a_t, s_{t+1})$ which cannot be differentiated.
- The only way for the policy variables θ to influence $R(\tau)$ is by changing the state and action distributions which, in turn, change the rewards received by an agent

- the gradient of the expectation can be rewritten as follows:

$$
\begin{aligned}
&\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] \\
&= \nabla_\theta \int dx\ f(x)p(x\,|\,\theta) && (\textit{definition of expectation}) \\
&= \int dx\ \nabla_\theta \big(p(x\,|\,\theta)f(x)\big) && (\textit{bring in } \nabla_\theta) \\
&= \int dx\ \big(f(x)\nabla_\theta p(x\,|\,\theta) + p(x\,|\,\theta)\nabla_\theta f(x)\big) && (\textit{chain-rule}) \\
&= \int dx\ f(x)\nabla_\theta p(x\,|\,\theta) && (\nabla_\theta f(x) = 0) \\
&= \int dx\ f(x)p(x\,|\,\theta)\frac{\nabla_\theta p(x\,|\,\theta)}{p(x\,|\,\theta)} && \left(\textit{multiply } \frac{p(x\,|\,\theta)}{p(x\,|\,\theta)}\right) \\
&= \int dx\ f(x)p(x\,|\,\theta)\nabla_\theta \log p(x\,|\,\theta) && (\textit{substitute Equation 2.14}) \\
&= \mathbb{E}_x[f(x)\nabla_\theta \log p(x\,|\,\theta)] && (\textit{definition of expectation})
\end{aligned}
$$

$$
\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] = \int dx\ f(x)\nabla_\theta p(x\,|\,\theta)
$$

- Above Equation has solved our initial problem since we can take the gradient of p(x | θ), but f(x) is a black-box function which cannot be integrated.
- To deal with this, we need to convert the equation into an expectation so that it can be estimated through sampling.
- So, we first multiplied equation identically by p(x|θ)/p(x|θ) in the next step.
- The resulting fraction $\frac{\nabla_\theta p(x|\theta)}{p(x|\theta)}$ can be rewritten with the log-derivative trick as follows:

$$
\nabla_\theta \log p(x\,|\,\theta) = \frac{\nabla_\theta p(x\,|\,\theta)}{p(x\,|\,\theta)}
$$

It should be apparent that this identity can be applied to our objective. By substituting x = τ, f(x) = R(τ), p(x | θ) = p(τ | θ),

$$
\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \quad \Rightarrow \quad \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log p(\tau\,|\,\theta)]
$$

Objective

$$
\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log p(x\,|\,\theta)]
$$

- Observe that the trajectory τ is just a sequence of interleaved events, a_t and s_{t+1}, sampled, respectively, from the agent's action probability $\pi_\theta(a_t \mid s_t)$ and the environment's transition probability $p(s_{t+1} \mid s_t, a_t)$.
- Since the probabilities are conditionally independent, the probability of the entire trajectory is the product of the individual probabilities as shown below:

$$p(\tau \mid \theta) = \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$

$$p(\tau \mid \theta) = \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$

Apply logarithms to both sides

$$\log p(\tau \mid \theta) = \log \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$

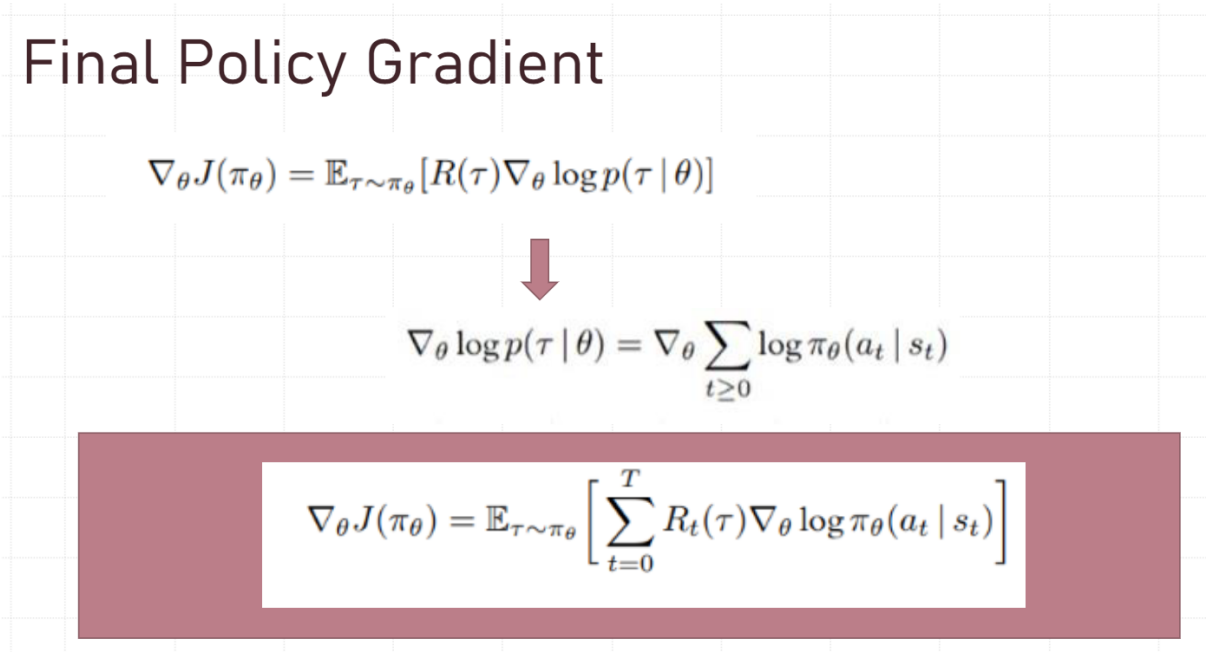
$$\log p(\tau \mid \theta) = \sum_{t \geq 0} \left(\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t) \right)$$

$$\log p(\tau \mid \theta) = \log \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$

$$\log p(\tau \mid \theta) = \sum_{t \geq 0} \left(\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t) \right)$$

$$\nabla_\theta \log p(\tau \mid \theta) = \nabla_\theta \sum_{t \geq 0} \left(\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t) \right)$$

$$\nabla_\theta \log p(\tau \mid \theta) = \nabla_\theta \sum_{t \geq 0} \log \pi_\theta(a_t \mid s_t)$$



20. Write REINFORCE algorithm

The key idea is that during learning, actions that resulted in good outcomes should become more probable - these actions are positively reinforced. Conversely, actions that resulted in bad outcomes should become less probable. If learning is successful, over the course of many iterations action probabilities produced by the policy shift to distribution that results in a good performance in an environment. Action probabilities are changed by following the policy gradient, therefore REINFORCE is known as a policy gradient algorithm.

The steps involved in the implementation of REINFORCE would be as follows:

1. Initialize a Random Policy (a NN that takes the state as input and returns the probability of actions)
2. Use the policy to play N steps of the game — record action probabilities-from policy, reward from the environment, action — sampled by agent
3. Calculate the discounted reward for each step by backpropagation
4. Calculate the expected reward G
5. Adjust weights of Policy (back-propagate error in NN) to increase G
6. Repeat step 2

Algorithm 2.1 REINFORCE algorithm

Initialize learning rate α
Initialize weights θ of a policy network π_θ
for *episode* $= 0, \ldots,$ *MAX_EPISODE* **do**
  Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
  Set $\nabla_\theta J(\pi_\theta) = 0$
  for $t = 0, \ldots, T$ **do**
    $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$ ← $R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$
    $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
  end for
  $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

21. Define Off-Policy and On-Policy RL algorithms and give example of each

An algorithm is on-policy if it learns about the policy-that is, and training can only utilize data generated from the current policy π. This implies that as training iterates through versions of policies, π1, π2, π3, …, each training iteration only uses the current policy at that time to generate training data. As a result, all the data must be discarded after training since it becomes unusable. This makes on-policy methods sample-inefficient–they require more training data. Examples include REINFORCE, SARSA, and Actor-critic.

In contrast, an algorithm is off-policy if it does not have this screenshot. Any data collected can be reused in training. Consequently, off-policy methods are more sample-efficient, but this may require much more memory to store the data. Examples include DQN.

22. Write REINFORCE algorithm and Explain how baseline improves REINFORCE algorithm?

Algorithm 2.1 REINFORCE algorithm

Initialize learning rate α
Initialize weights θ of a policy network π_θ
for *episode* $= 0, \ldots,$ *MAX_EPISODE* **do**
  Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
  Set $\nabla_\theta J(\pi_\theta) = 0$
  for $t = 0, \ldots, T$ **do**
    $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$ ← $R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t$
    $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
  end for
  $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

How Baseline improves REINFORCE

- Consider the case where all the rewards for an environment are negative.
- Without a baseline, even when an agent produces a very good action, it gets discouraged because the returns are always negative.
- Over time, this can still result in good policies since worse actions will get discouraged even more, thus indirectly increasing the probabilities of better actions.
- However, it can lead to slower learning because probability adjustments can only be made in one direction.
- The converse happens for environments where all the rewards are positive.
- Learning is more effective when we can both increase and decrease the action probabilities.
- This requires having both positive and negative returns.

23. You have a task which is to show relative ads to target users. Which algorithm should you use for this task?

A real-life example of an e-greedy action selection algorithm can be seen in online advertising. When a user visits a website, the website can choose to show an ad that has the highest estimated click-through rate (CTR) with a certain probability (1-epsilon). On the other hand, it can show a random ad with probability epsilon. This approach helps to maximize revenue for the website while still exploring different ads to see if they have higher CTRs.