POLICY GARDIENT APPROACH

By: Dimple Bohra

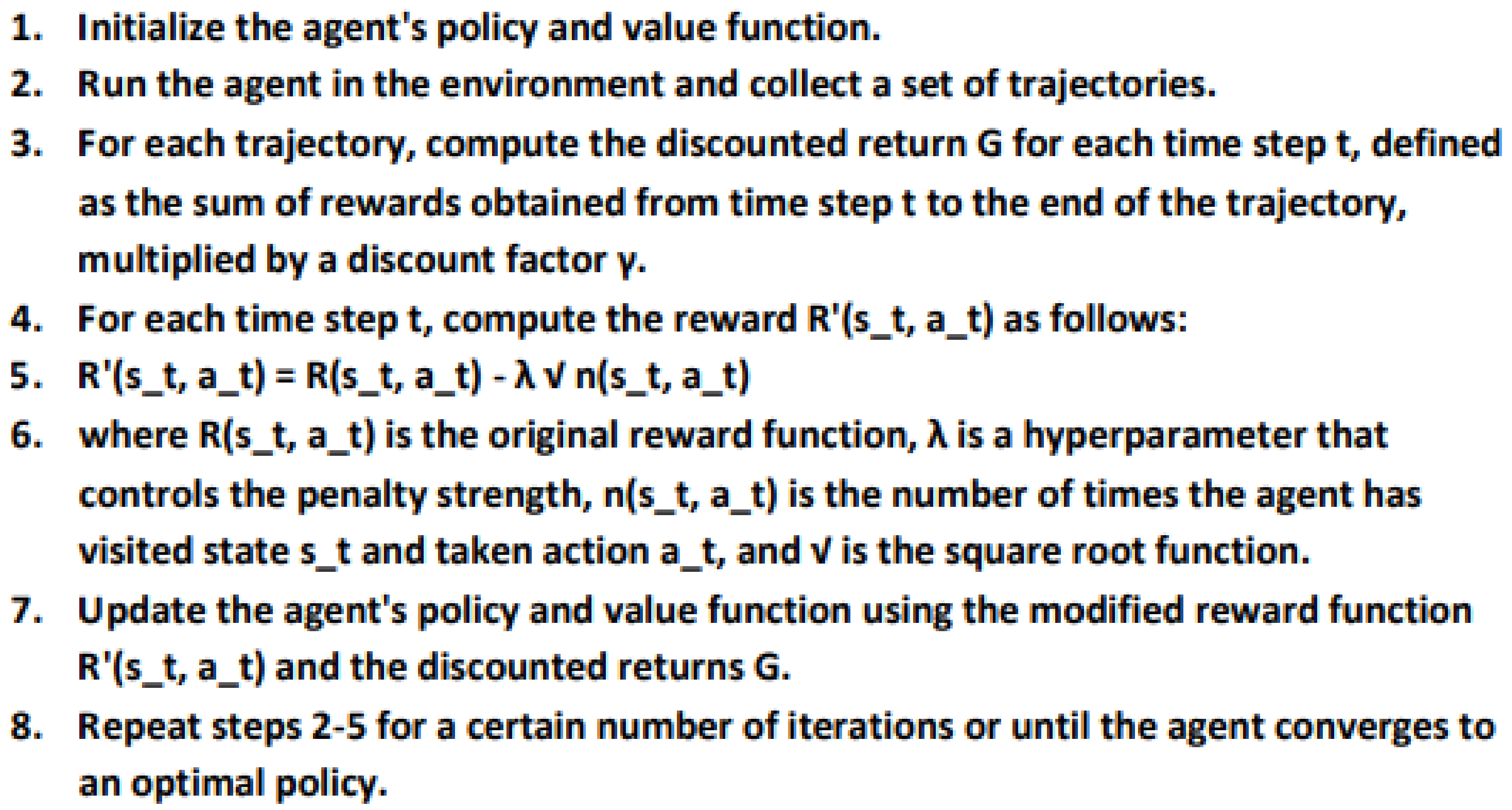
Contents

Reference: chapter 2 from Foundations of Deep Reinforcement Learning: Theory and Practice in Python by Laura Graesser and Wah Loon Keng

Linear Reward Penalty 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 in the linear reward penalty algorithm 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 exploration of new and potentially better policies.
- The linear reward penalty 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 the 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 from time step t to the end of the trajectory, multiplied by a discount factor γ.
4. For each time step t, compute the reward R'(s_t, a_t) as follows:
5. R'(s_t, a_t) = R(s_t, a_t) - λ √ n(s_t, a_t)
6. 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, and √ is the square root function.
7. Update the agent's policy and value function using the modified reward function R'(s_t, a_t) and the discounted returns G.
8. Repeat steps 2-5 for a certain number of iterations or until the agent converges to an optimal policy.

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 which 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 good performance in an environment.
- Action probabilities are changed by following the policy gradient, therefore REINFORCE is known as a policy gradient algorithm

REINFORCE Algorithm

- The algorithm needs three components:

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

RL as MDP

An MDP is defined by a 4-tuple S, A, P(.), R(.), where

- S is the set of states.
- A is the set of actions.
- $P(s_{t+1} \mid s_t, a_t)$ is the state transition function of the environment.
- $R(s_t, a_t, s_{t+1})$ is the reward function of the environment.
- One important assumption underlying the reinforcement learning problems is agents do not have access to the transition function, P(st+1 | st, at), or the reward function, R(st, at, st+1).
- The only way an agent can get information about these functions is through the states, actions, and rewards it actually experiences in the environment—that is, the tuples (st, at, rt)

RL as MDP

- To complete the formulation of the problem, we also need to formalize the concept of

an objective which an agent maximizes. First, let's define the return R(τ) using
a trajectory from an episode, τ = (s0, a0, r0), . . . ,(sT , aT , rT):

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

The objective J(τ) is simply the expectation of the returns over many trajectories,

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

RL as MDP

- The discount factor γ ∈ [0, 1] is an important variable which changes the way future rewards are valued. The smaller γ, the less weight is given to rewards in future time steps, making it “shortsighted.” In the extreme case with γ = 0, the objective only considers the initial reward r0

$$R(\tau)_{\gamma=0} = \sum_{t=0}^{T} \gamma^t r_t = r_0$$

- The larger γ, the more weight is given to rewards in future time steps: the objective becomes more “farsighted.” If γ = 1, rewards from every time step are weighted equally

$$R(\tau)_{\gamma=1} = \sum_{t=0}^{T} \gamma^t r_t = \sum_{t=0}^{T} r_t$$

Policy

- A policy π is a function, mapping states to action probabilities, which is used to sample an action a ~ π(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 . $\theta_1 \neq \theta_2$. .
- 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 Function

- 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

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

Policy Gradient

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

- 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 $R_t(\tau)$.

Policy Gradient

- 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

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

$$\big\downarrow$$

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

- 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(τ) is by changing the state and action distributions which, in turn, change the rewards received by an agent

Policy Gradient Derivation

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

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

Policy Gradient Derivation

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

- 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:

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

Policy Gradient Derivation

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),

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

Objective

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

Policy Gradient Derivation

- 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:

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

Policy Gradient Derivation

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

Apply logarithms to both sides

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

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

Policy Gradient Derivation

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

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

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

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

Final Policy Gradient

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

$$\Downarrow$$

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

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

Monte Carlo Sampling

- The REINFORCE algorithm numerically estimates the policy gradient using Monte Carlo sampling.
- Monte Carlo sampling refers to any method that uses random sampling to generate data used to approximate a function.
- In essence, it is just “approximation with random sampling.
- Example: Estimate the value of π (the mathematical constant)—the ratio of a circle’s circumference to its diameter

Monte Carlo Sampling

Estimate the **value of π** (the mathematical constant)

- A Monte Carlo approach to solving this problem is to take a circle of radius r = 1 centered at the origin and inscribe it in a square.
- Their areas are πr^2 and $(2r)^2$, respectively. Hence, the ratio of these areas is simply

$$\frac{\textit{area of circle}}{\textit{area of square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

Monte Carlo Sampling

Numerically, the square has an area of 4, but since we do not yet know π, so the area of the circle is unknown. Let's consider one quadrant.

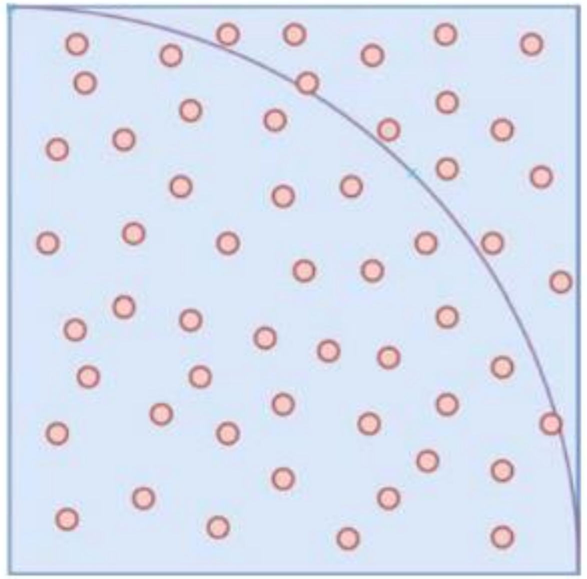


Figure 2.1 Monte Carlo sampling used to estimate π

To obtain an estimation for π, sample many points within the square using a uniformly random distribution. A point (x, y) that lands in the circle has distance less than 1 from the origin—that is, $\sqrt{(x-0)^2+(y-0)^2} \leq 1$

Monte Carlo Sampling

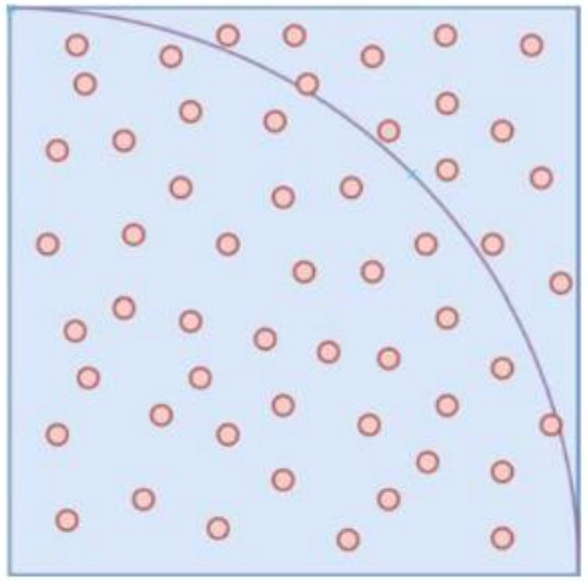


Figure 2.1 Monte Carlo sampling used to estimate π

- Using this, if we count the number of points in the circle, then count the number of points sampled in total, their ratio roughly equals to $\frac{\textit{area of circle}}{\textit{area of square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$
- By iteratively sampling more points and updating the ratio, our estimation will get closer to the precise value.
- Multiplying this ratio by 4 gives us the estimated value of π ≈ 3.14159.

Monte Carlo for REINFORCE

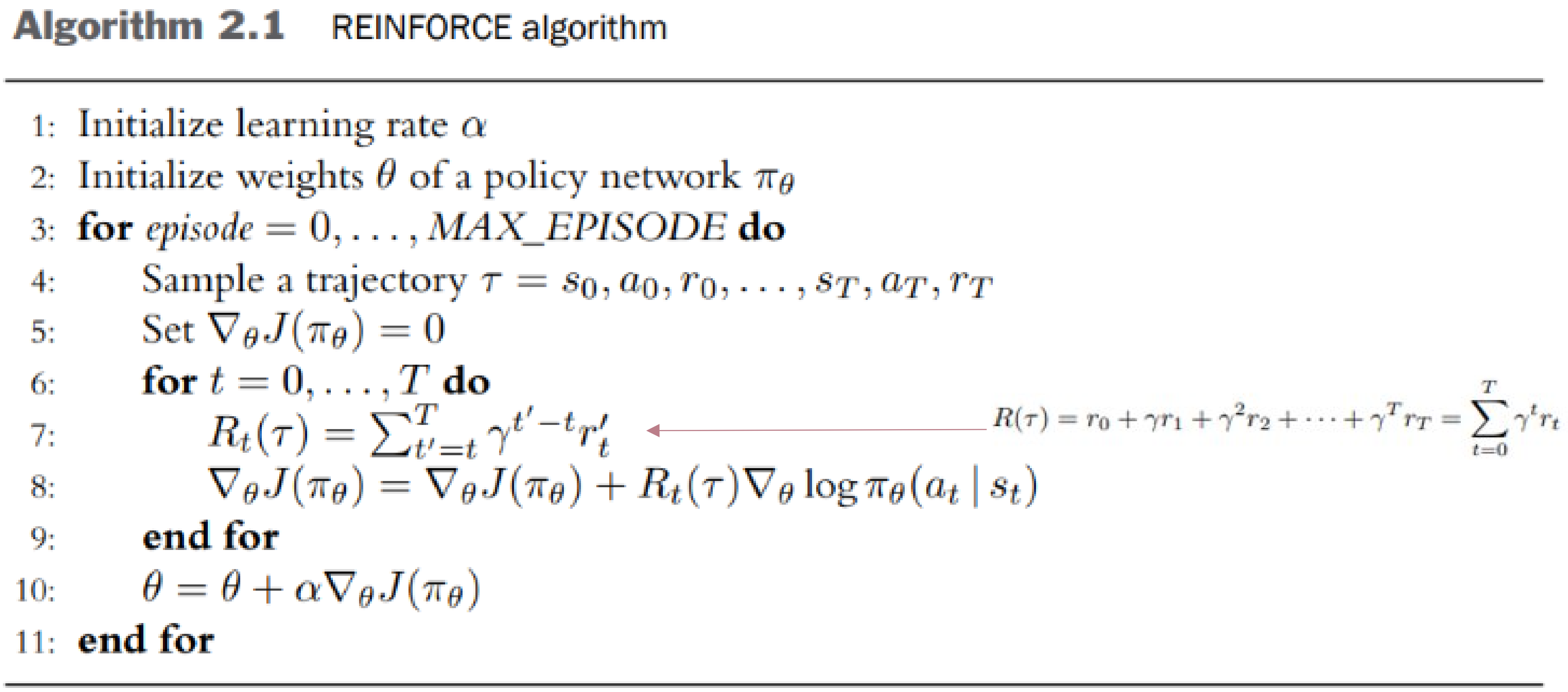
- Monte Carlo can be used to numerically estimate the policy gradient. It is very straightforward. The expectation $E_{\tau\sim\pi\theta}$ implies that as more trajectories τ s are sampled using a policy $\pi\theta$ and averaged, it approaches the actual policy gradient $\nabla\theta J(\pi\theta)$.
- Instead of sampling many trajectories per policy, we can sample just one as shown

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

$$\Big\downarrow$$

$$\boxed{\nabla_\theta J(\pi_\theta) \approx \sum_{t=0}^{T} R_t(\tau)\nabla_\theta \log \pi_\theta(a_t \mid s_t)}$$

REINFORCE Algorithm



Algorithm 2.1 REINFORCE algorithm

```
1:  Initialize learning rate $\alpha$
2:  Initialize weights $\theta$ of a policy network $\pi_\theta$
3:  for $episode = 0, \ldots, MAX\_EPISODE$ do
4:      Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
5:      Set $\nabla_\theta J(\pi_\theta) = 0$
6:      for $t = 0, \ldots, T$ do
7:          $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$
8:          $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
9:      end for
10:     $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
11: end for
```

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

On Policy and Off Policy RL Algorithms

- An algorithm is **on-policy** if it learns on the policy—that is, 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: REINFORCE, SARSA, Actor-critic

On Policy and Off Policy RL Algorithms

- In contrast, an algorithm is **off-policy** if it does not have this requirement.
- 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.
- Example: DQN

Improving REINFORCE

- Our formulation of the REINFORCE algorithm estimates the policy gradient using Monte Carlo sampling with a single trajectory. This is an unbiased estimate of the policy gradient, but one disadvantage of this approach is that it has a high variance.
- High Variance because the returns can vary significantly from trajectory to trajectory. This is due to three factors.
- First, actions have some randomness because they are sampled from a probability distribution.
- Second, the starting state may vary per episode.
- Third, the environment transition function may be stochastic

Improving REINFORCE

- One way to reduce the variance of the estimate is to modify the returns by subtracting a suitable action-independent baseline, as shown below:

$$\nabla_\theta J(\pi_\theta) \approx \sum_{t=0}^{T} \left(R_t(\tau) - b(s_t)\right) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

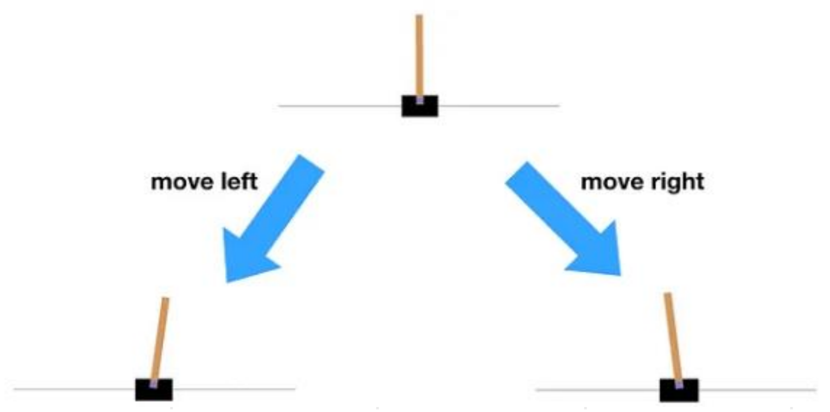
- One option for the baseline is the value function V^π.
- An alternative is to use the mean returns over the trajectory. Let $b = \frac{1}{T} \sum_{t=0}^{T} R_t(\tau)$
- Note that this is a constant baseline per trajectory that does not vary with state s_t. It has the effect of centering the returns for each trajectory around 0. For each trajectory, on average, the best 50% of the actions will be encouraged, and the others discouraged

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.

Example : CartPole

- CartPole, also known as inverted pendulum, is a game in which you try to balance the pole as long as possible. It is assumed that at the tip of the pole, there is an object which makes it unstable and very likely to fall over. The goal of this task is to move the cart left and right so that the pole can stand (within a certain angle) as long as possible.



Example : Lunar lander

- This environment deals with the problem of landing a lander on a landing pad.
- Initially agent does not know how to control and land a rocket, but with time it learns from its mistakes and start to improve its performance and in the end learns to fully control the rocket and perfectly lands it.

THANK YOU!!