Deep Learning

Day 20: Policy Gradient and Actor Critic

CSCI 1470

Eric Ewing

Thursday, 11/13/25

Terminology Review

MDP: Markov Decision Process

<States, Actions, Reward Function, Transition Probabilities, Discount Factor>

Episode: Single run-through of an MDP (from start to terminal state)

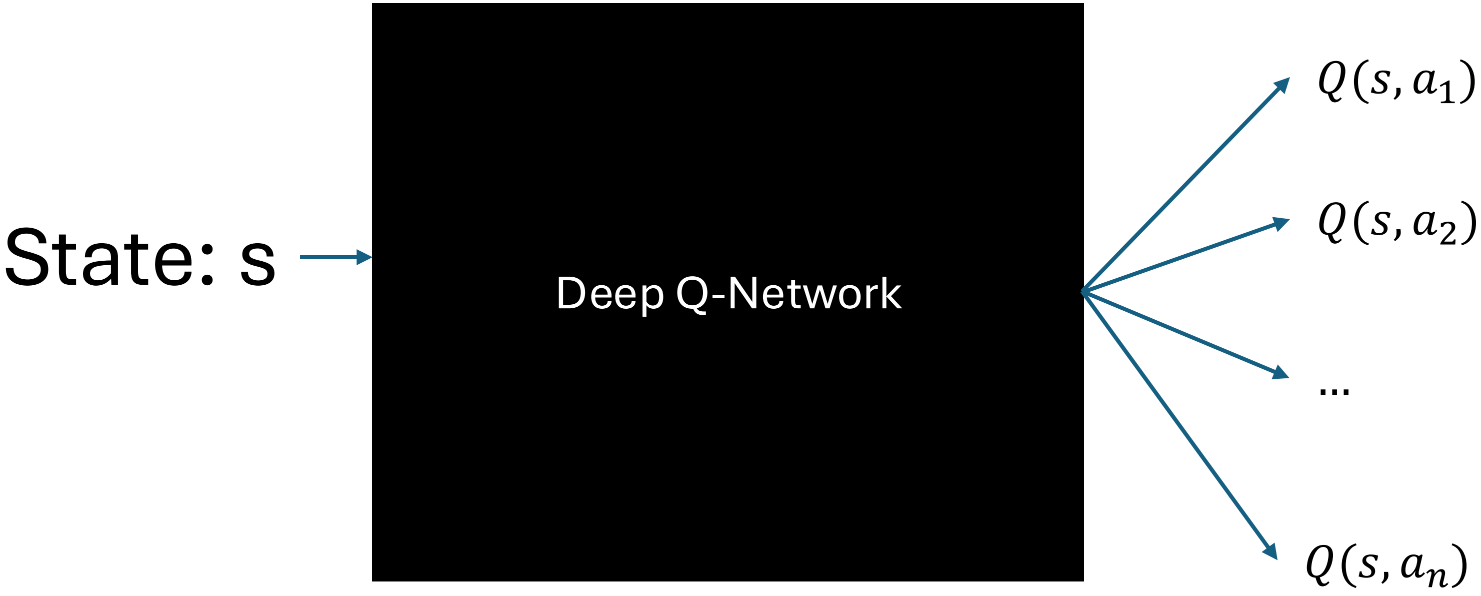
Trajectory: states, actions, and rewards received in an episode

Returns: Total (discounted) rewards of a trajectory

Value: Expected Returns from a specific state

Q-Value: Expected Returns from a specific state when taking a specific action

Deep-Q Network



Deep Q-Networks (DQNs):

1. Take in a state
2. Return Q-values for each action

Deep-Q Learning

Initialize DQN to approximate Q

Maintain estimates of Q(s, a) for all (s, a) pairs

Collect experiences, update Q estimates with:

Compute $L_\theta = \left[r + \gamma \max_{a'} Q_\theta(s', a') \ - Q_\theta(s, a)\right]^2$

update θ with SGD on Loss function

Gymnasium

Gymnasium

Provides a common implementation of MDPs for RL.

A gym environment has a current state that can be updated by calling environment.step(action)

```python
import gymnasium as gym

# Initialise the environment
env = gym.make("LunarLander-v3", render_mode="human")

# Reset the environment to generate the first observation
observation, info = env.reset(seed=42)
for _ in range(1000):
    # this is where you would insert your policy
    action = env.action_space.sample()

    # step (transition) through the environment with the action
    # receiving the next observation, reward and if the episode has terminated or truncated
    observation, reward, terminated, truncated, info = env.step(action)

    # If the episode has ended then we can reset to start a new episode
    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

Implementing DQNs

```
# Compute Q(s_t, a) - the model computes Q(s_t), then we select the
# columns of actions taken. These are the actions which would've been taken
# for each batch state according to policy_net
state_action_values = policy_net(state_batch).gather(1, action_batch)

# Compute V(s_{t+1}) for all next states.
# Expected values of actions for non_final_next_states are computed based
# on the "older" target_net; selecting their best reward with max(1).values
# This is merged based on the mask, such that we'll have either the expected
# state value or 0 in case the state was final.
next_state_values = torch.zeros(BATCH_SIZE, device=device)
with torch.no_grad():
    next_state_values[non_final_mask] = target_net(non_final_next_states).max(1).values
# Compute the expected Q values
expected_state_action_values = (next_state_values * GAMMA) + reward_batch

# Compute Huber loss
criterion = nn.SmoothL1Loss()
loss = criterion(state_action_values, expected_state_action_values.unsqueeze(1))
```

Source: Torch DQN tutorial https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

Policies

Why learn Q-values first and turn them into a policy? Why not just learn a policy?

What should the activation function be for the final layer?

How do we train a policy network?

How do we train a policy network?

Need to find an appropriate loss function.

How do we train a policy network?

Need to find an appropriate loss function.

What’s our objective?

How do we train a policy network?

Need to find an appropriate loss function.

What’s our objective?

Find a policy π such that the value of the start state is maximized:

How do we train a policy network?

Need to find an appropriate loss function.

What’s our objective?

Find a policy π such that the value of the start state is maximized:

$$\pi = \mathrm{argmax}_{\pi}\,(V(s_0))$$

How do we train a policy network?

Need to find an appropriate loss function.

What's our objective?

Find a policy π such that the value of the start state is maximized:

$$\pi = \text{argmax}_{\pi} \left(V(s_0)\right)$$

How can we maximize $V(s_0)$?

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Returns of a specific trajectory

Probability of a trajectory occurring

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Returns of a specific trajectory

Probability of a trajectory occurring

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Probability of a trajectory occurring

Returns of a specific trajectory

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

State transition Probability

Probability of taking an action for a given state

Let $J(\theta)$ be our objective function:

$$J(\theta) = V(s_0)$$

$$J(\theta) = \mathbb{E}[G_0]$$

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Sum over all possible trajectories

Probability of a trajectory occurring

Returns of a specific trajectory

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

State transition Probability

Probability of taking an action for a given state

Log-Derivative Trick

We can rewrite the derivative of a function using the derivative of the natural log function:

$$\nabla \ln f(x) = \frac{\nabla \mathrm{f}(x)}{f(x)}$$

$$\nabla f(x) = f(x) \nabla \ln \mathrm{f}(\mathrm{x})$$

When applied to $\Pr(\tau|\theta)$:

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t) + \ln \pi_\theta(a_t|s_t)$$

Log Probability Trick

$$\Pr(\tau|\theta) = \Pi_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

$$\nabla_\theta \Pr(\tau|\theta) = \Pr(\tau|\theta)\, \nabla_\theta \ln \Pr(\tau|\theta)$$

This gradient term is what we want to calculate

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Log of product -> sum of logs

$$\nabla_\theta \ln \Pr(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T} \ln P(s_{t+1}|s_t, a_t) + \ln \pi_\theta(a_t|s_t)$$

Derivative of sum -> sum of derivative

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Gradient of a trajectory

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln P(s_{t+1}|s_t, a_t) + \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

State transition function
does not depend on θ!

$$\nabla_\theta \ln \Pr(\tau|\theta) = \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)$$

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$

Take the gradient

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$

Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$

Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$

Log-Derivative Trick

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$ Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$ Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$ Log-Derivative Trick

$$\nabla_\theta J(\theta) = \sum_{\tau} \left[\Pr(\tau|\theta) G(\tau) \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)\ \right]$$ Gradient of a Trajectory

Policy Gradient Derivation

Putting it all back together:

$$J(\theta) = \sum_{\tau} \Pr(\tau|\theta)\, G(\tau)$$ Our Objective

$$\nabla_\theta J(\theta) = \sum_{\tau} \nabla_\theta \Pr(\tau|\theta)\, G(\tau)$$ Take the gradient

$$\nabla_\theta J(\theta) = \sum_{\tau} \Pr(\tau|\theta) G(\tau) \nabla_\theta \ln \Pr(\tau|\theta)$$ Log-Derivative Trick

$$\nabla_\theta J(\theta) = \sum_{\tau} [\Pr(\tau|\theta) G(\tau) \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)\ ]$$ Gradient of a Trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$ Convert back to Expectation

Policy Gradient

Bigger step if better returns

Direction to move in to increase probability of trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

We will never be able to sum over all possible trajectories…

How do we get around this?

Policy Gradient

Bigger step if better returns

Direction to move in to increase probability of trajectory

$$\nabla_\theta J(\theta) = \mathbb{E}[G_0 \sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

We will never be able to sum over all possible trajectories...

How do we get around this?

Sampling!

1. Collect n trajectories following policy π_θ
2. $\Pr(\tau|\theta) = 1/n$ for each trajectory
3. Calculate the total return for each trajectory $G(\tau)$

Reward-To-Go Policy Gradient

You can also do the policy gradient derivation such that the gradient does not depend on G_0, but on G_t

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Or

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} Q(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

REINFORCE (Policy Gradient Learning)

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Source: Sutton and Barto, Reinforcement Learning: An Introduction

REINFORCE (Policy Gradient Learning)

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Why is the update adding the gradient instead of subtracting?

Source: Sutton and Barto, Reinforcement Learning: An Introduction

REINFORCE (Policy Gradient Learning)

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

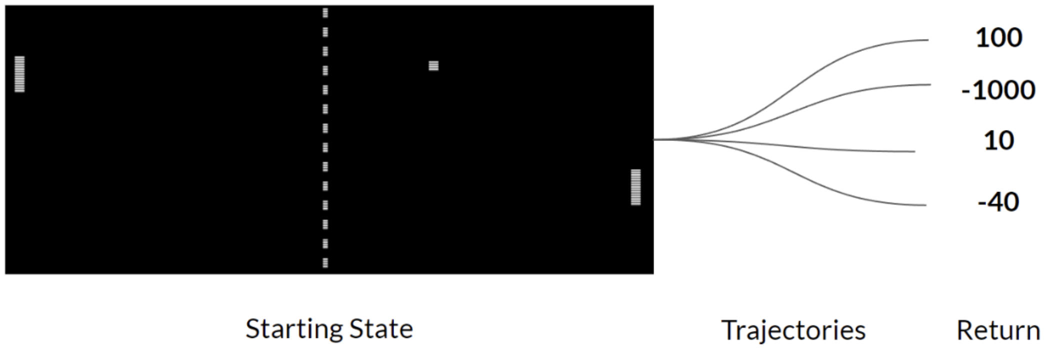
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Why is the update adding the gradient instead of subtracting?

When π is based on a softmax, $\nabla_\theta \ln \pi_\theta(a|s)$ is actually easy to compute by hand using log rules and the fact that $\ln e^x = x$

Source: Sutton and Barto, Reinforcement Learning: An Introduction

Variance of REINFORCE

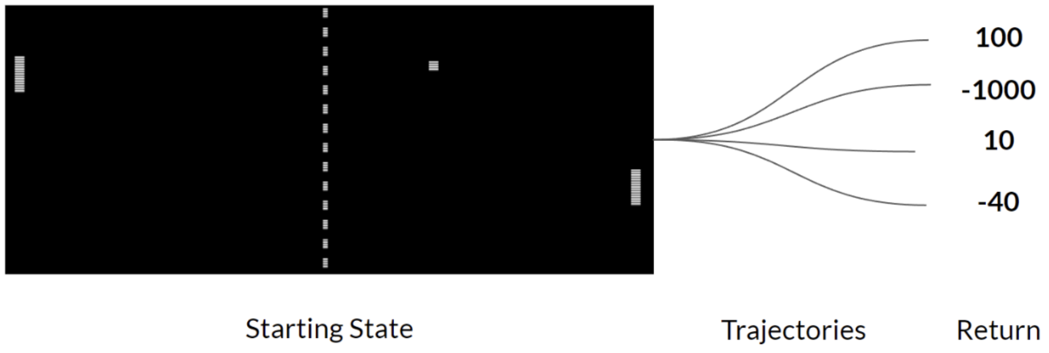


REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Variance of REINFORCE

REINFORCE has **high** variance



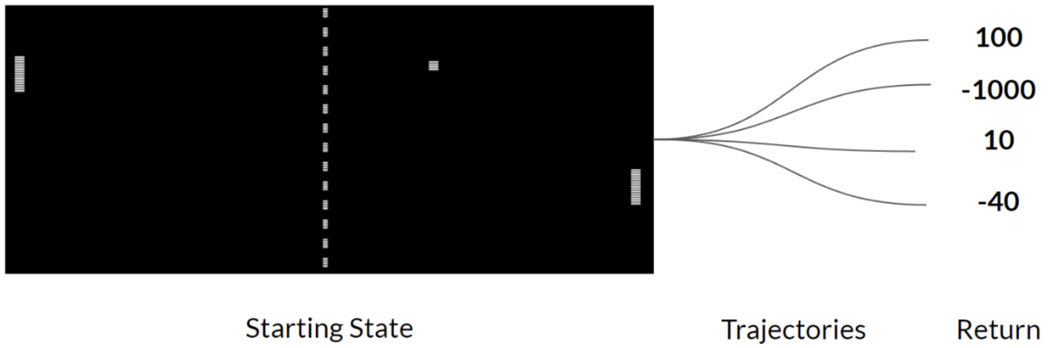
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Variance of REINFORCE

REINFORCE has **high** variance

It depends heavily on the returns of a single episode



REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

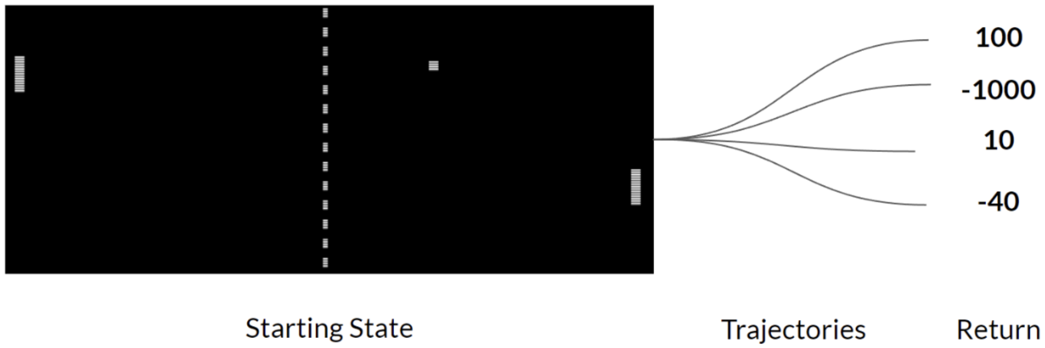
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Variance of REINFORCE

REINFORCE has **high** variance

It depends heavily on the returns of a single episode

We can reduce variance by collecting more than one trajectory



REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

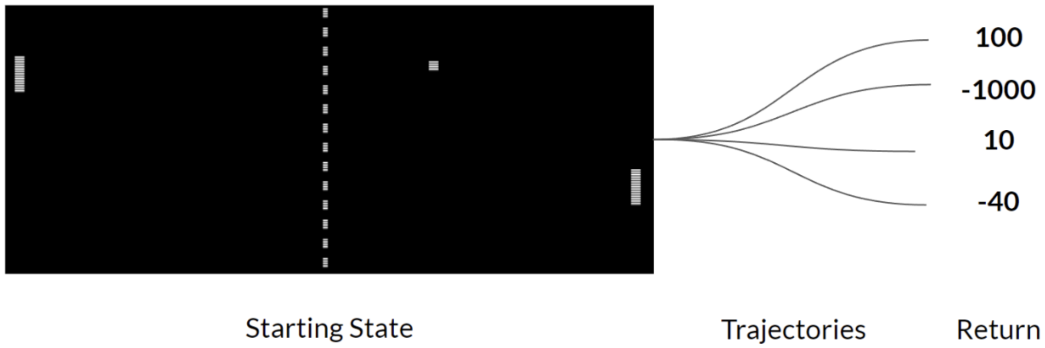
Variance of REINFORCE

REINFORCE has **high** variance

It depends heavily on the returns of a single episode

We can reduce variance by collecting more than one trajectory

Or…



REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Baseline Functions

Baseline Functions

Subtracting a baseline function from G_t does not change the expected gradient

Baseline Functions

Subtracting a *baseline* function from G_t does not change the expected gradient

A baseline function $b(s)$ is any function that depends only on the state (not on actions)

Baseline Functions

Subtracting a baseline function from G_t does not change the expected gradient

A baseline function $b(s)$ is any function that depends only on the state (not on actions)

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T}(G_t - b(s))\,\nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Baseline Functions

Subtracting a baseline function from G_t does not change the expected gradient

A baseline function $b(s)$ is any function that depends only on the state (not on actions)

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T}(G_t - b(s))\,\nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Baseline functions can reduce the variance of the gradient estimate

Baseline Functions

Subtracting a baseline function from G_t does not change the expected gradient

A baseline function $b(s)$ is any function that depends only on the state (not on actions)

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T}(G_t - b(s))\,\nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Baseline functions can reduce the variance of the gradient estimate

The value function V(s) is the ideal baseline function

Derivation of REINFORCE w/ Baseline Function

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} (G_t - b(s))\, \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

First, let's show that the gradient estimate is unbiased. We see that with the baseline, we can distribute and rearrange and get:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'}\right) - \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\right]$$

Due to linearity of expectation, all we need to show is that for any single time t, the gradient of $\log \pi_\theta(a_t|s_t)$ multiplied with $b(s_t)$ is zero. This is true because

$$\begin{aligned}
\mathbb{E}_{\tau\sim\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\Big] &= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[\mathbb{E}_{s_{t+1:T},a_{t:T-1}}[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)]\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t)\cdot \underbrace{\mathbb{E}_{s_{t+1:T},a_{t:T-1}}[\nabla_\theta \log \pi_\theta(a_t|s_t)]}_{E}\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t)\cdot \mathbb{E}_{a_t}[\nabla_\theta \log \pi_\theta(a_t|s_t)]\Big] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\Big[b(s_t)\cdot 0\Big] = 0
\end{aligned}$$

$$\mathbb{E}_{a_t}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\Big] = \int \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \pi_\theta(a_t|s_t) da_t = \nabla_\theta \int \pi_\theta(a_t|s_t) da_t = \nabla_\theta \cdot 1 = 0$$

Derivation: https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/

REINFORCE with Baseline

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

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

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

Pseudocode uses SGD, but you can just as easily use any other optimizer (e.g., Adam)

Source: Sutton and Barto Chapter 13

REINFORCE with Baseline

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

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

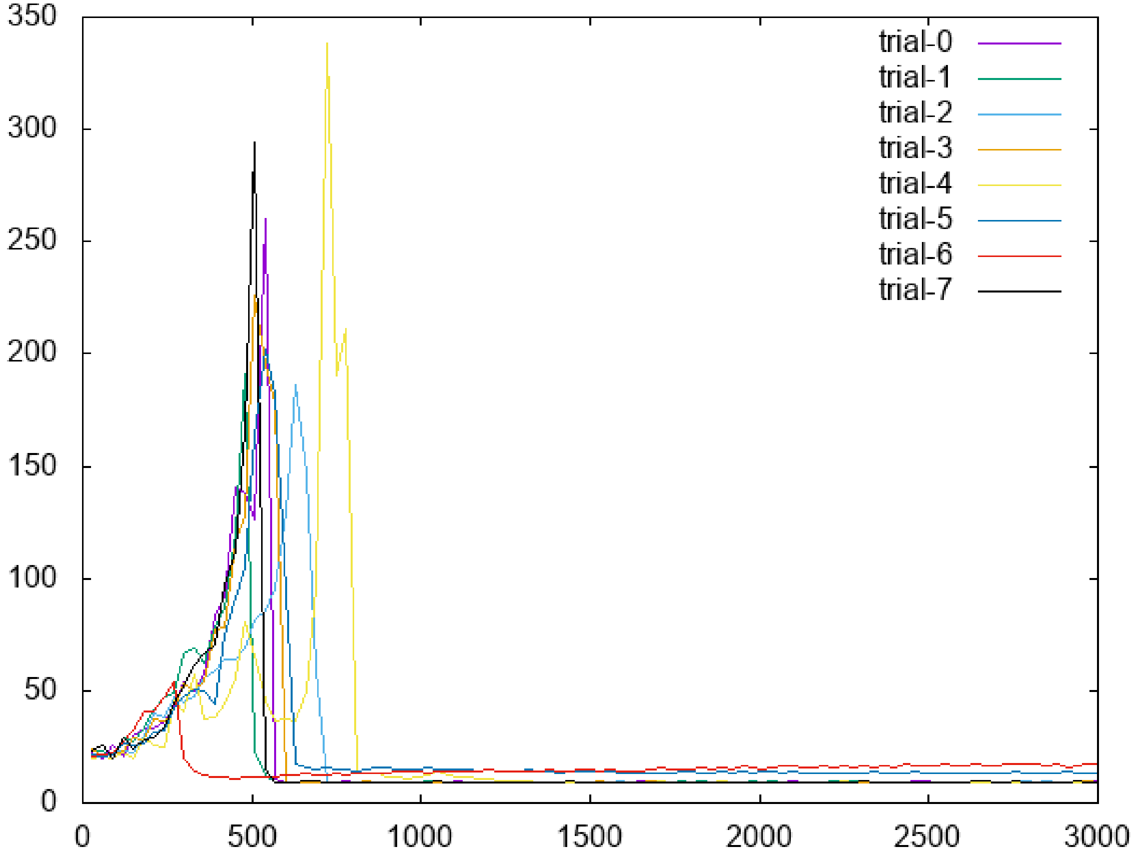
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G -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

Gradient of L $= \frac{1}{2}\delta$^2

Pseudocode uses SGD, but you can just as easily use any other optimizer (e.g., Adam)

Source: Sutton and Barto Chapter 13

Key Idea for RL: Variance is the enemy



Cartpole

Policy Collapse: https://stats.stackexchange.com/questions/252685/policy-gradient-reward-collapse

Let’s do an example with Multi-Arm Bandits

What’s a one-armed bandit?

Multi-Arm Bandits

Multi-Arm Bandits

Single-armed bandit

Multi-Arm Bandits

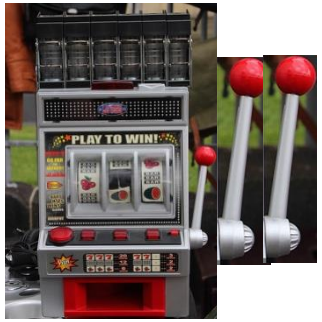
Single-armed bandit

Multi-Arm Bandits

Single-armed bandit

Multi-Arm Bandits

Single-armed bandit

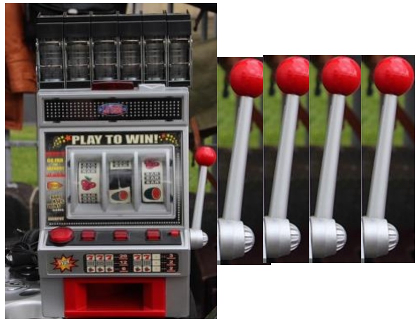


Multi-Arm Bandits

Single-armed bandit

Multi-Arm Bandits

Single-armed bandit

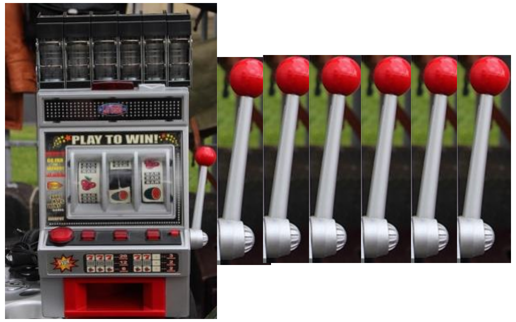


Multi-Arm Bandits

Single-armed bandit

Multi-Arm Bandits

Single-armed bandit



Multi-Arm Bandits

Single-armed bandit

Multi-Arm Bandits

When an arm is pulled, the rewards are random.

Each arm returns a reward with (different) unknown mean and variance

Single-armed bandit

Multi-Arm Bandits

When an arm is pulled, the rewards are random.

Each arm returns a reward with (different) unknown mean and variance

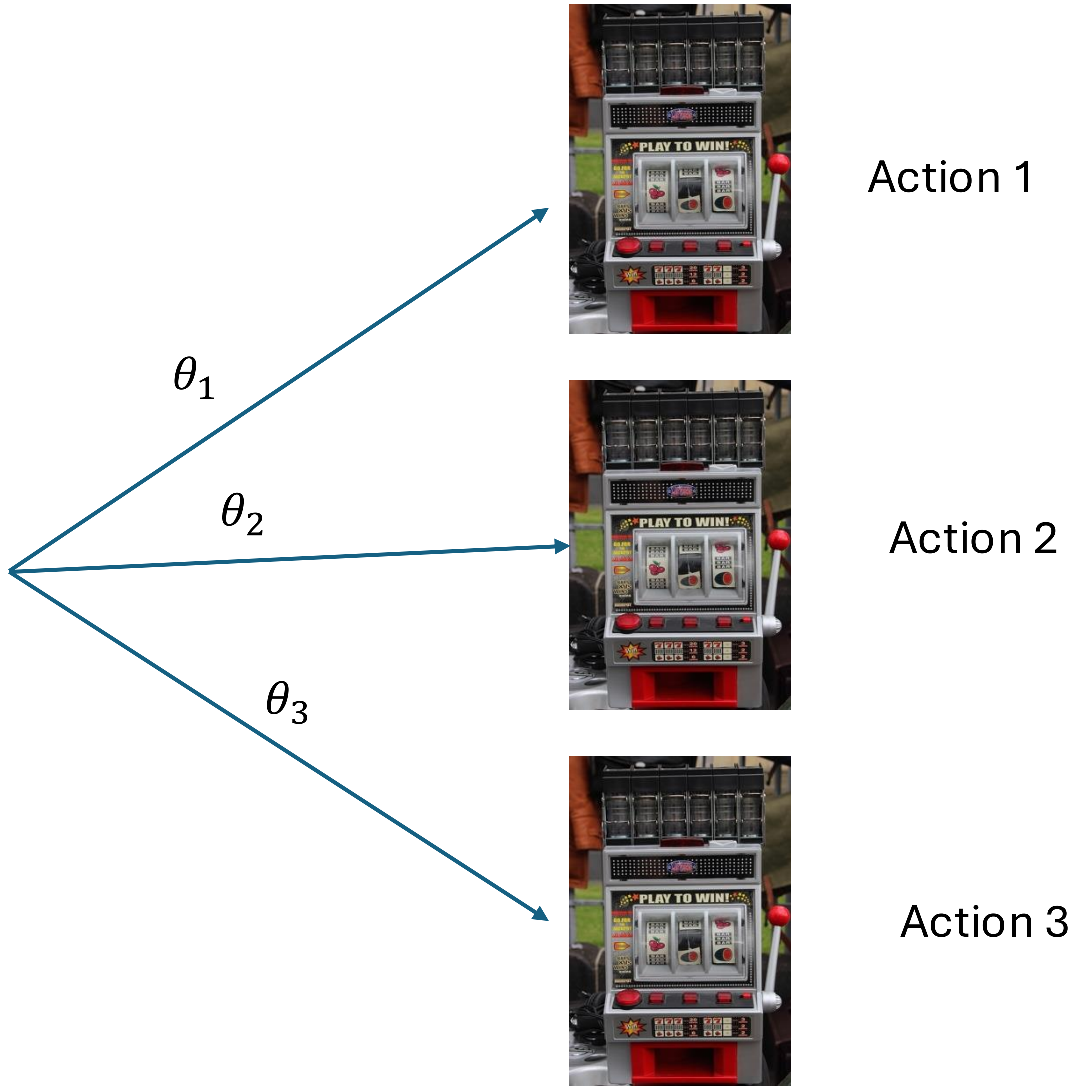
Single-armed bandit

Bandit Problems are essentially MDPs with a single state.

Useful testbed for a number of algorithms and very useful for theory

Policy Gradient on Multi-Arm Bandits

Maintain parameter for each action, θ_i

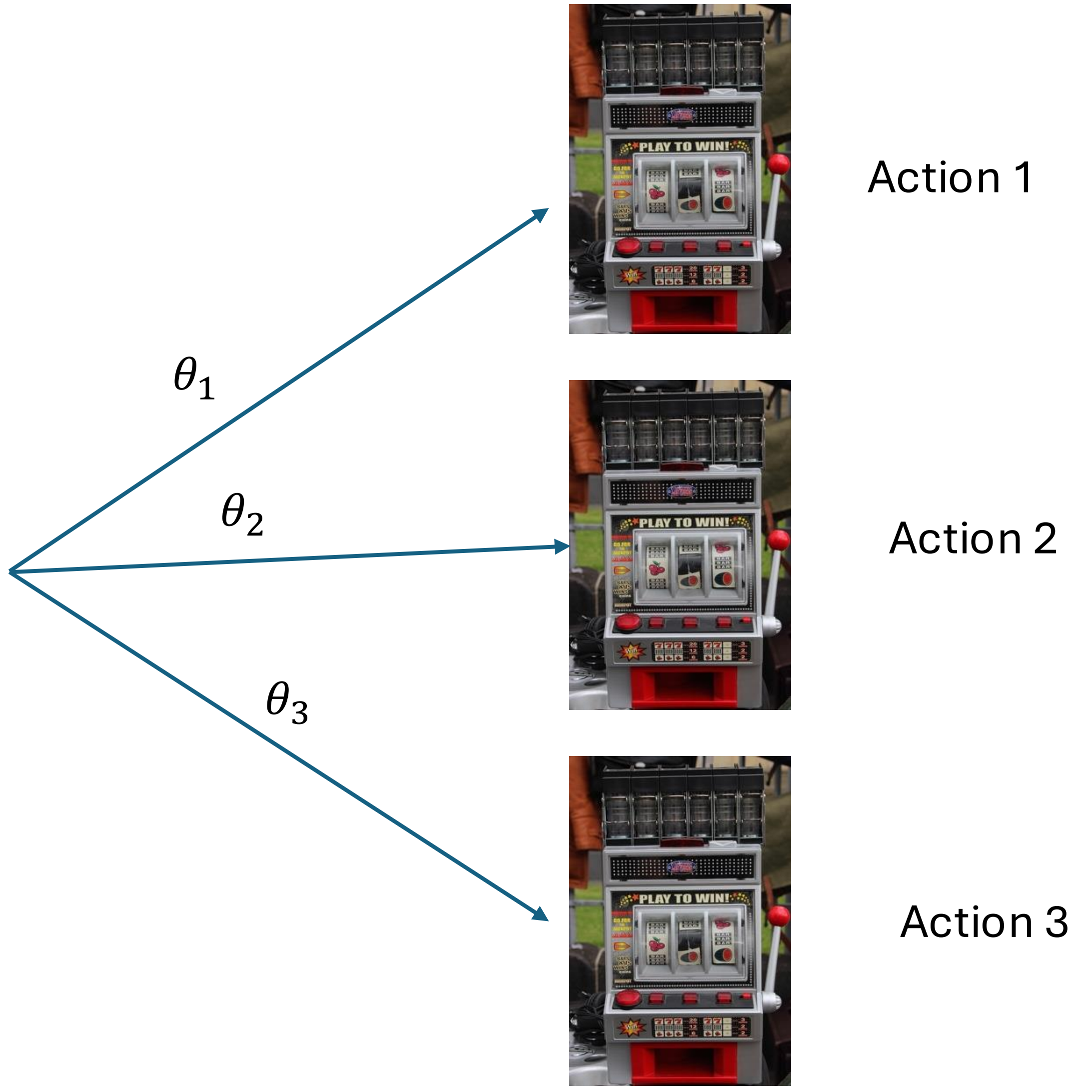


Policy Gradient on Multi-Arm Bandits

Maintain parameter for each action, θ_i

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

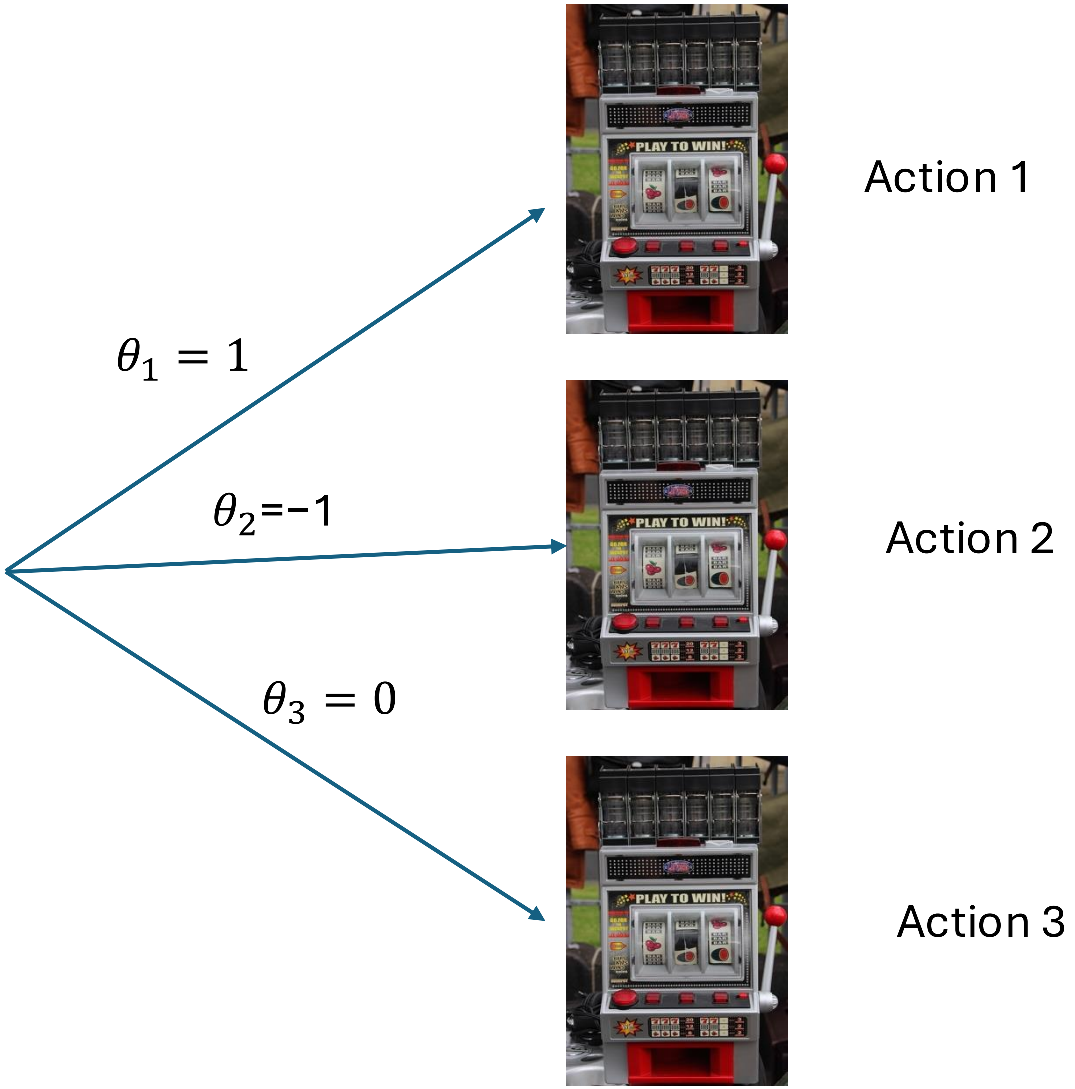


Policy Gradient on Multi-Arm Bandits

Maintain parameter for each action, θ_i

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$



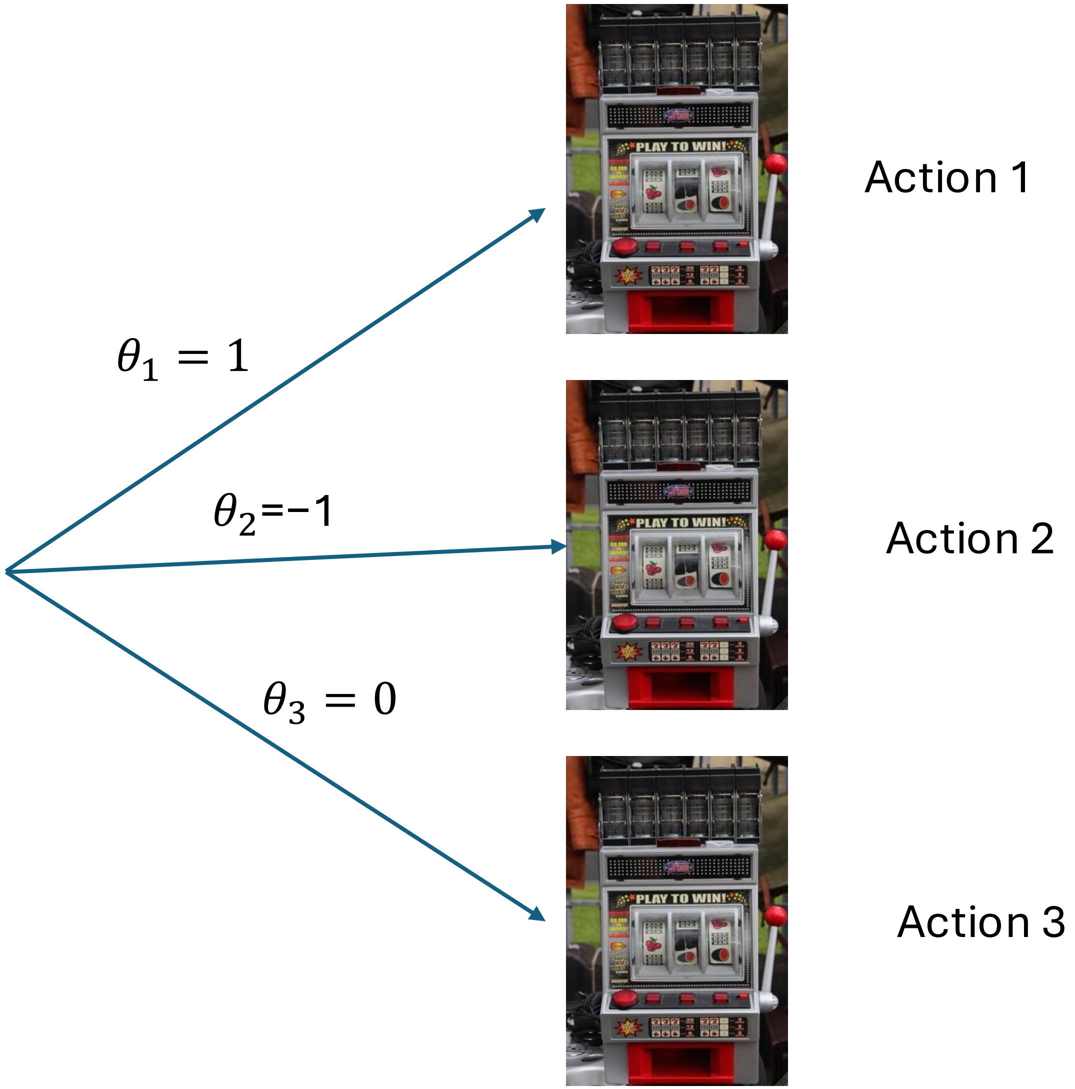
Policy Gradient on Multi-Arm Bandits

Maintain parameter for each action, θ_i

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\pi_\theta(a_1) = \frac{e}{e + \frac{1}{e} + 1} = 0.66$$



Policy Gradient on Multi-Arm Bandits

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

T

Policy Gradient on Multi-Arm Bandits

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

Take 5 actions according to π_θ:

$$\tau = (a_1, 3), (a_2, -1), (a_3, 2), (a_1, 4), (a_3, 1)$$

T

Policy Gradient on Multi-Arm Bandits

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

Take 1 action according to π_θ:

$$\tau = (a_1, 3)$$

Policy Gradient (without states): $\nabla_\theta J(\theta) = G_t^T \nabla_\theta \ln \pi_\theta(a \quad)$

Take Action according to Softmax:

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a\quad)$$

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:

Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a\quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

What is $\nabla_\theta \ln \pi_\theta(a_1)$?

$$\nabla_\theta \ln \pi_\theta(a_1) = \nabla_\theta \ln \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

What is $\nabla_\theta \ln \pi_\theta(a_1)$?

$$\nabla_\theta \ln \pi_\theta(a_1) = \nabla_\theta \ln \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$= \nabla_\theta \left[\ln e^{\theta_1} - \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)\right]$$
$$= \nabla_\theta \theta_1 - \nabla_\theta \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)$$

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

What is the shape of $\nabla_\theta J(\theta)$?

What is $\nabla_\theta \ln \pi_\theta(a_1)$?

$$\nabla_\theta \ln \pi_\theta(a_1) = \nabla_\theta \ln \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$= \nabla_\theta \left[\ln e^{\theta_1} - \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)\right]$$
$$= \nabla_\theta \theta_1 - \nabla_\theta \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)$$

Take Action according to Softmax:

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

What is the shape of $\nabla_\theta J(\theta)$?

What is $\nabla_\theta \ln \pi_\theta(a_1)$?

$$\nabla_\theta \ln \pi_\theta(a_1) = \nabla_\theta \ln \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$= \nabla_\theta \left[\ln e^{\theta_1} - \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)\right]$$
$$= \nabla_\theta \theta_1 - \nabla_\theta \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)$$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}} \\ \cdots \\ \cdots \end{bmatrix}$$

Take Action according to Softmax:

$$\pi_\theta(a_1) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

$$\theta_1 = 1, \pi_\theta(a_1) = 0.66$$
$$\theta_2 = -1, \pi_\theta(a_2) = 0.09$$
$$\theta_3 = 0, \pi_\theta(a_3) = 0.25$$

$$\nabla_\theta J(\theta) = G_t \nabla_\theta \ln \pi_\theta(a \quad)$$

$$\tau = (a_1, 3)$$

At timestep 1, a_1 got a reward of 3:
Need to compute: $G_t \nabla_\theta \ln \pi_\theta(a_1)$

$$G_t \nabla_\theta \ln \pi_\theta(a_1) = 3 \cdot \begin{bmatrix} 0.34 \\ -0.09 \\ -0.25 \end{bmatrix}$$

What is G_0?
Bandits is not a sequential problem (no future rewards to worry about), so it's just the observed reward for taking the first action, a_1.

$$G_t = 3$$

What is the shape of $\nabla_\theta J(\theta)$?

What is $\nabla_\theta \ln \pi_\theta(a_1)$?

$$\nabla_\theta \ln \pi_\theta(a_1) = \nabla_\theta \ln \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}}$$

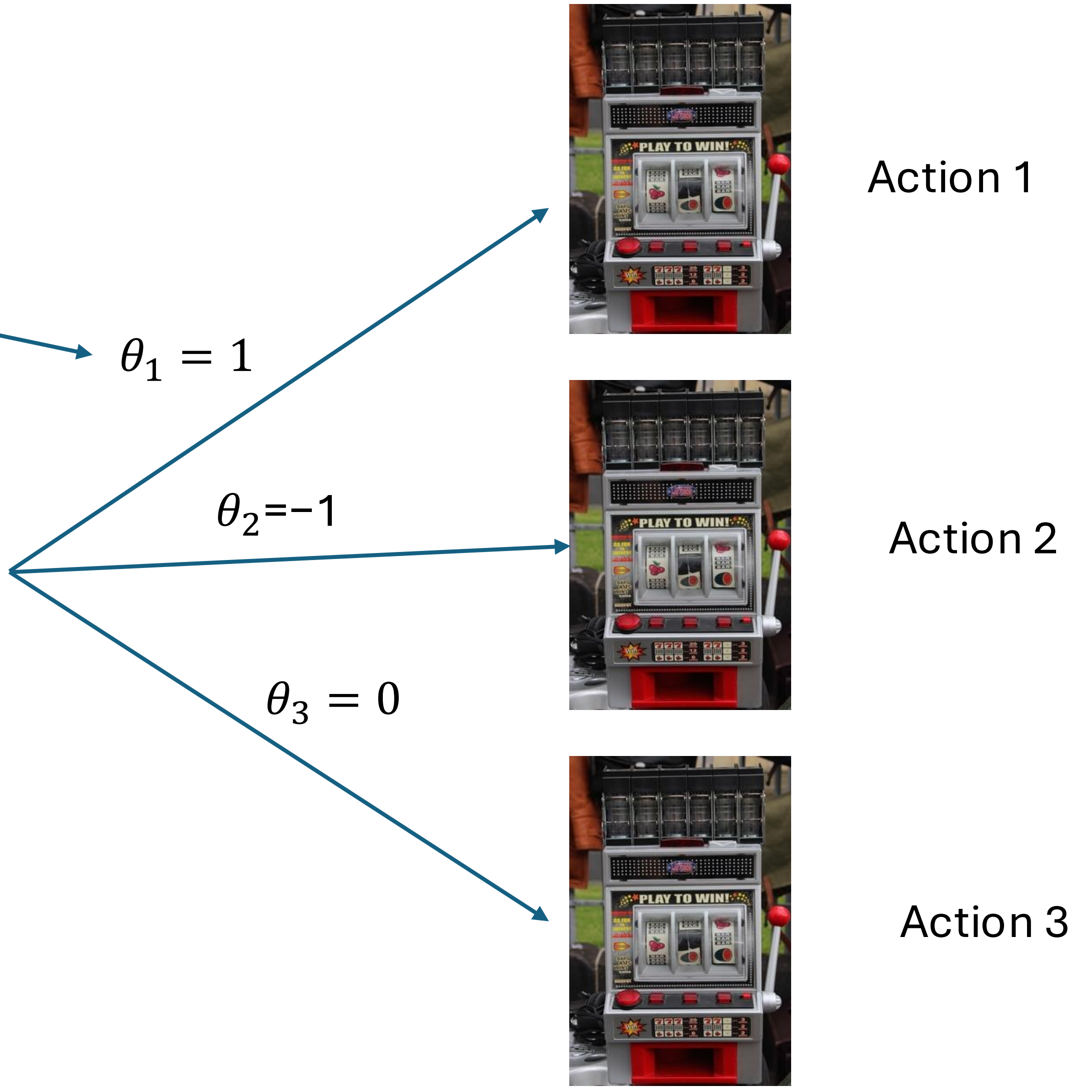
$$= \nabla_\theta \left[\ln e^{\theta_1} - \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)\right]$$
$$= \nabla_\theta \theta_1 - \nabla_\theta \ln\left(e^{\theta_1} + e^{\theta_2} + e^{\theta_3}\right)$$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2} + e^{\theta_3}} \\ \cdots \\ \cdots \end{bmatrix}$$

REINFORCE

What if θ is the output of a neural network?

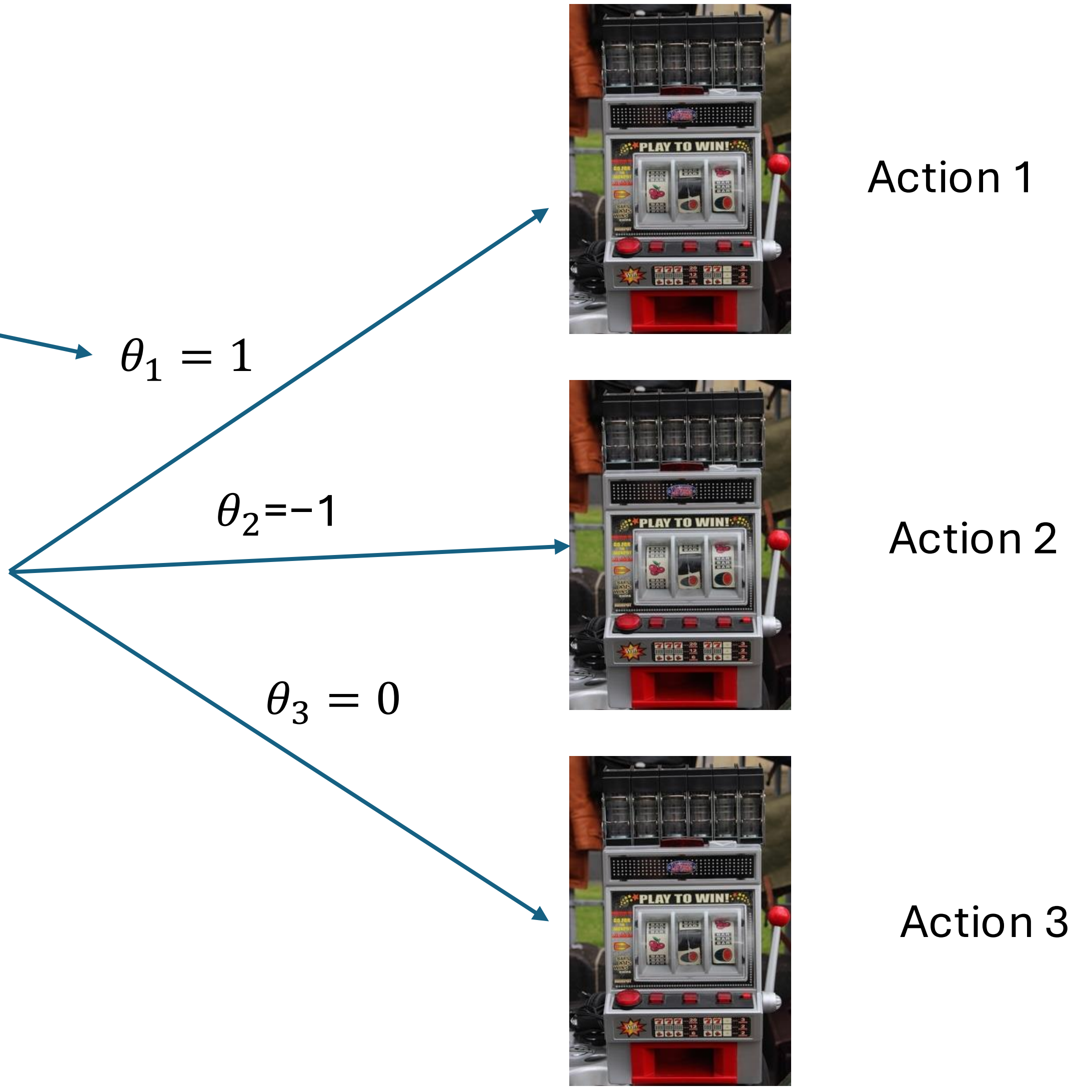
How to compute: $G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)$?



REINFORCE

What if θ is the output of a neural network?

How to compute:
$G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)$?



Compute $\ln \pi_\theta(a_t|s_t)$ in gradient tape context.

REINFORCE

What if θ is the output of a neural network?

How to compute:
$G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)$?

$\theta_1 = 1$

θ_2=−1

$\theta_3 = 0$

Action 1

Action 2

Action 3

Compute $\ln \pi_\theta(a_t|s_t)$ in gradient tape context.

But also, remember, you have to perform gradient ASCENT.
If an optimizer minimizes by default, you can use $-\nabla_\theta J(\theta)$

REINFORCE Variance

If we could calculate $\nabla_\theta J(\theta)$ exactly (not just for single trajectory/sample), then Policy Gradient would be a great algorithm! (with some minor flaws)

Results on Cartpole

Image Source: https://medium.com/@fork.tree.ai/understanding-baseline-techniques-for-reinforce-53a1e2279b57

Actor-Critic Methods

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem...

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Methods learn an approximation of G_t

Variance of Returns is always a problem…

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

Variance of Returns is always a problem…

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

Variance of Returns is always a problem…

$V(s_t) = \mathbb{E}[G_t]$ as well, but technically it's not a critic function. Critic functions critique actions.

Actor-Critic Methods

REINFORCE uses the return for a trajectory G_t:

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Variance of Returns is always a problem...

Actor-Critic Methods learn an approximation of G_t

$$Q^\pi(s_t, a_t) = \mathbb{E}[G_t]$$

$V(s_t) = \mathbb{E}[G_t]$ as well, but technically it's not a critic function. Critic functions critique actions.

$$\nabla_\theta J(\theta) = \mathbb{E}[\sum_{t=0}^{T} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)]$$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Critic: Scores the action

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Actor (policy): Takes actions

Critic: Scores the action

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Policy network has parameters θ
Q network has parameters w

Initialize $\pi_{\theta}, Q_w, \alpha_{\theta}, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward *r*

Sample next action $a' \sim \pi_{\theta}(a|s)$

update $\theta \leftarrow \theta + \alpha_{\theta} Q_w(s,a) \nabla_{\theta} \ln \pi_{\theta}(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s',a') - Q_w(s,a)$

update $w \leftarrow w + \alpha_w \delta \nabla_w Q_w(s,a)$

$a \leftarrow a', s \leftarrow s'$

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Policy network has parameters θ
Q network has parameters w

Initialize $\pi_{\theta}, Q_w, \alpha_{\theta}, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward r

Sample next action $a' \sim \pi_{\theta}(a|s)$

update $\theta \leftarrow \theta + \alpha_{\theta} Q_w(s,a) \nabla_{\theta} \ln \pi_{\theta}(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s', a') - Q_w(s,a)$

update $w \leftarrow w + \alpha_w \delta \nabla_w Q_w(s,a)$

$a \leftarrow a', s \leftarrow s'$

Like Q-learning and REINFORCE at the same time

Variations on a Theme...

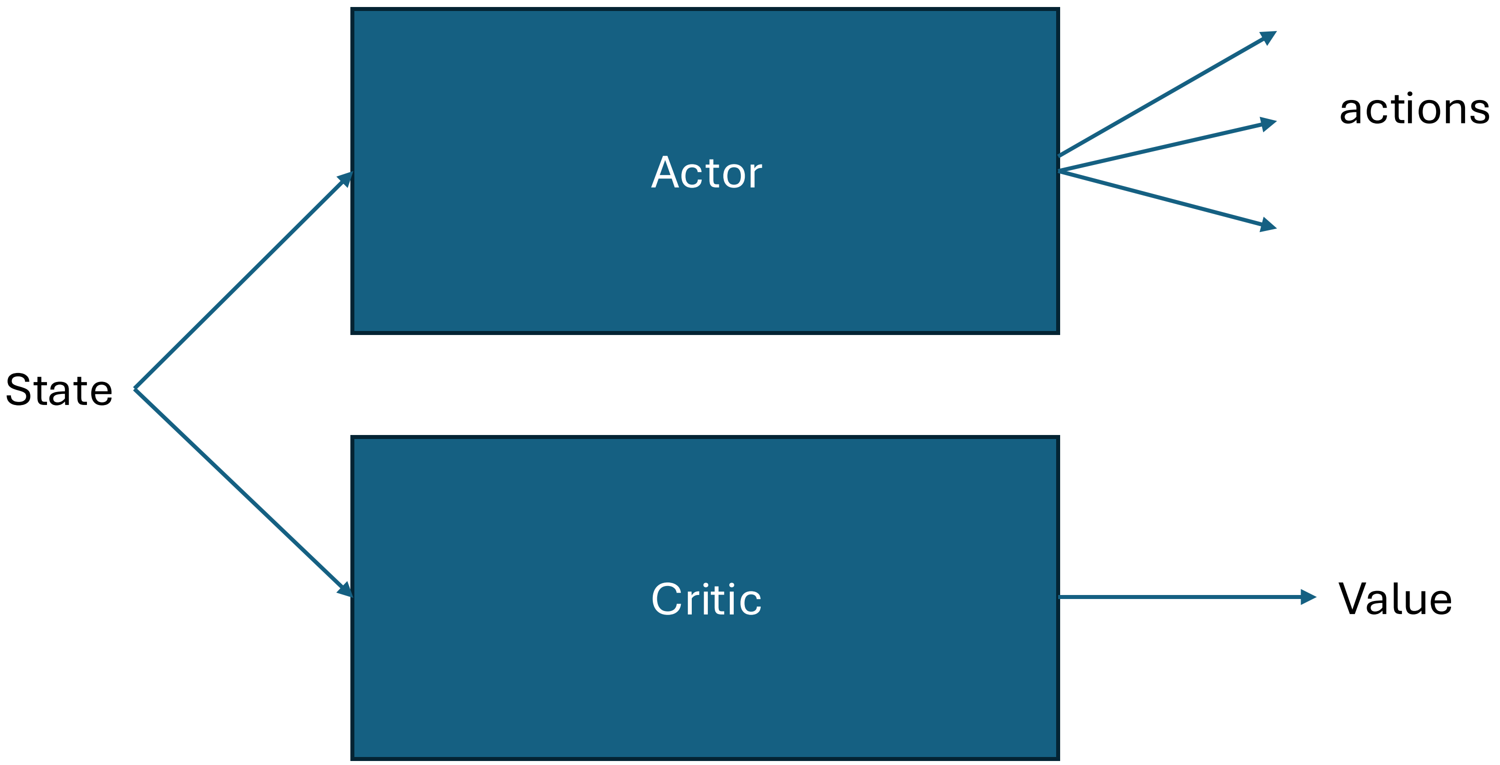
How to estimate $J(\theta)$

(Wikipedia uses R_t instead of G_t)

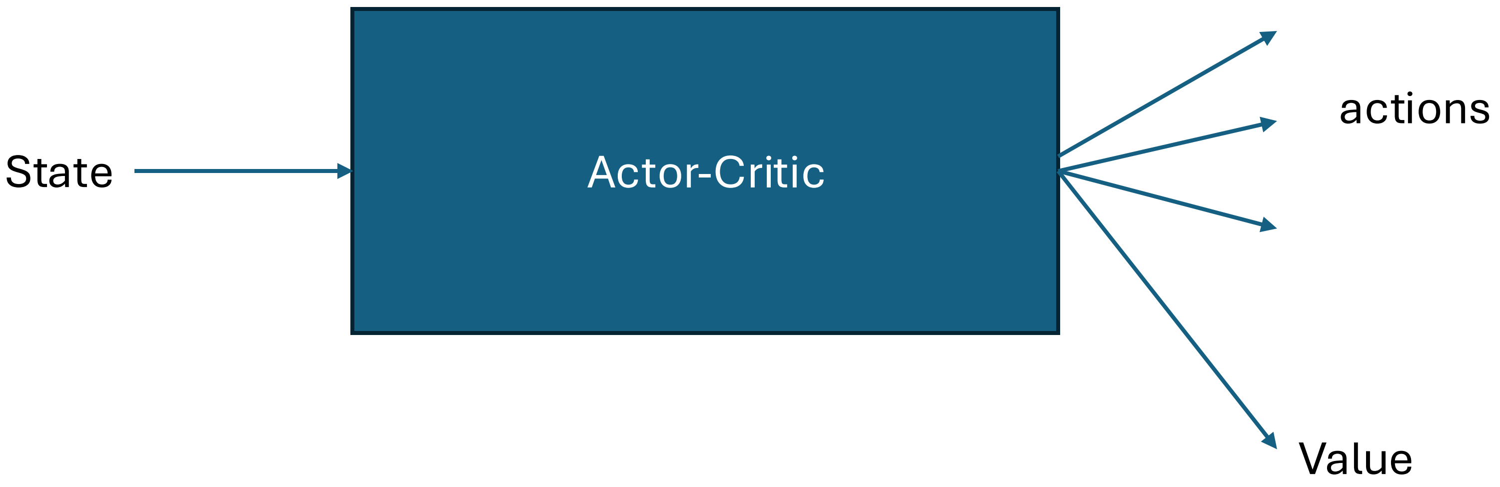
- $\sum_{0 \le i \le T} (\gamma^i R_i)$.
- $\gamma^j \sum_{j \le i \le T} (\gamma^{i-j} R_i)$: the **REINFORCE** algorithm.
- $\gamma^j \sum_{j \le i \le T} (\gamma^{i-j} R_i) - b(S_j)$: the **REINFORCE with baseline** algorithm. Here b is an arbitrary function.
- $\gamma^j \left(R_j + \gamma V^{\pi_\theta}(S_{j+1}) - V^{\pi_\theta}(S_j)\right)$: TD(1) learning.
- $\gamma^j Q^{\pi_\theta}(S_j, A_j)$.
- $\gamma^j A^{\pi_\theta}(S_j, A_j)$: **Advantage Actor-Critic (A2C)**.[3]
- $\gamma^j \left(R_j + \gamma R_{j+1} + \gamma^2 V^{\pi_\theta}(S_{j+2}) - V^{\pi_\theta}(S_j)\right)$: TD(2) learning.
- $\gamma^j \left(\sum_{k=0}^{n-1} \gamma^k R_{j+k} + \gamma^n V^{\pi_\theta}(S_{j+n}) - V^{\pi_\theta}(S_j)\right)$: TD(n) learning.
- $\gamma^j \sum_{n=1}^{\infty} \frac{\lambda^{n-1}}{1-\lambda} \cdot \left(\sum_{k=0}^{n-1} \gamma^k R_{j+k} + \gamma^n V^{\pi_\theta}(S_{j+n}) - V^{\pi_\theta}(S_j)\right)$: TD(λ) learning, also known as **GAE (generalized advantage estimate)**.[4] This is obtained by an exponentially decaying sum of the TD(n) learning terms.

Source: https://en.wikipedia.org/wiki/Actor-critic_algorithm

Actor-Critic Networks



Actor-Critic Networks



Just make sure you use the correct activation function for the different outputs

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Q-Learning uses:

$$\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

Actor-Critic Uses:

$$\delta = r + \gamma Q(s', a') \ - Q\ (s, a)$$

Deep Q-Learning Revisited

Compute TD-Error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

Loss Function: $L = \delta^2$

Update model with Gradient Descent

Q-Learning uses:

$$\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$$

Actor-Critic Uses:

$$\delta = r + \gamma Q(s', a') \ - Q\ (s, a)$$

Q-Learning is learning *Optimal Q-values*

Actor-Critic is learning the Q-values for following a specific policy Q^{π}

On-Policy Vs. Off-Policy Learning

RL algorithms collect experiences and learn from these experiences

On-Policy Algorithms have to collect experiences with the policy they are learning

Off-Policy Algorithms can use **any** policy to collect experiences

DQNs Are Off-Policy

In Q-Learning, we typically collect experiences using an ϵ-greedy policy in training

With probability ϵ, take random action.

Else, take action $\mathrm{argmax}_a\, Q(s, a)$

At test time, we should take the best actions, not the ϵ-greedy actions

$$\mathrm{argmax}_a\, Q(s, a)$$

DQNs Are Off-Policy

In Q-Learning, we typically collect experiences using an ϵ-greedy policy in training

With probability ϵ, take random action.

Else, take action $\text{argmax}_a\ Q(s, a)$

At test time, we should take the best actions, not the ϵ-greedy actions

$$\text{argmax}_a\ Q(s, a)$$

These are different policies! DQNs can be trained with any data collection policy at training time

Actor-Critic Algorithm: Learn Q^{π} and $\pi(a|s)$

Initialize $\pi_\theta, Q_w, \alpha_\theta, \alpha_w$

Repeat forever:

Take action a, get new state s' and reward r

Sample next action $a' \sim \pi_\theta(a|s)$

On Policy: Have to take actions according to π_θ

update $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \ln \pi_\theta(a|s)$

Calculate TD Error: $\delta = r + \gamma Q_w(s', a') - Q_w(s,a)$

update $w \leftarrow w + \alpha_w \delta \nabla_w Q_w(s,a)$

$a \leftarrow a', s \leftarrow s'$

On-Policy vs Off-Policy Learning

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
    - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
    - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
    - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
    - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

Disadvantages of Off-Policy Learning:

On-Policy vs Off-Policy Learning

Advantage of On-Policy Learning:

- More sample efficient, tend to converge faster

Disadvantages of On-Policy Learning:

- Can get stuck in local minima
  - Is there a simple policy that performs ok? Would small changes to that policy cause returns to go down temporarily?
  - How do balance exploration in our policy?

Advantages of Off-Policy Learning:

- Can learn from any policy
- More likely to learn an optimal policy

Disadvantages of Off-Policy Learning:

- Slower…

For the Record: On-Policy Q-Learning (SARSA)

There is an On-Policy Q-learning algorithm:

$$\delta = r + \gamma Q(s', a') \; - Q(s, a)$$

Why is it called SARSA?

$$\delta = \gamma Q(s', a') + r \; - Q\ (s, a)$$