Reinforcement Learning

$V_{pi}(s)$ = Expected Value of reward (with discount factor)

- Measures how good each state is in our environment

$Q_{pi}(s,a)$ = Expected value of reward if we take action a in state s

- Inputs policy pi
- Current state s
- Current action a

Bellman Equation

Optimal Q-function: Q*(s,a) → Q-function for the optimal policy pi*, which gives max possible future reward when taking action a in state s

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Q* encodes the optimal policy: $\pi^*(s) = \arg\max_{a'} Q(s,a')$

Bellman Equation states that Q* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

- Just says that the best policy occurs when we repeatedly take the best action in every state we see

Any Q function that satisfies bellman equation must be Q* (most optimal)

Deep Q-Learning
Idea: Use Bellman Equation as iterative update rule

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q^*.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$
$$\text{Where } r \sim R(s,a), s' \sim P(s,a)$$

Amazing fact: Q_i converges to Q^* as $i \to \infty$

Not tractable, very high branching factor which leads to extremely high numbers of compute

Hack: Train neural network to approximate Q-function, and optimize network with Q-function

For some problems, it might be better to learn a direct mapping from states to actions
Policy Gradients
- Takes a state as input, gives distribution over which action to take in that state
- Objective function: Expected future rewards when following policy pi_{theta}
  $$J(\theta) = \mathbb{E}_{r \sim p_\theta}\left[\sum_{t \geq 0} \gamma^t r_t\right]$$
    - Find the optimal policy by maximizing: $\theta^* = \arg\max_\theta J(\theta)$ **(Use gradient ascent!)**
- We don't actually know gradients through environment dJ/d(theta)
    - The world/simulation is not differentiable from our perspective

- Formulation: assign variable x to represent possible trajectories through environment

Policy Gradients: REINFORCE Algorithm

General formulation: $J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)]$ Want to compute $\frac{\partial J}{\partial \theta}$

$$\frac{\partial J}{\partial \theta} = \frac{\partial}{\partial \theta} \mathbb{E}_{x \sim p_\theta}[f(x)] = \frac{\partial}{\partial \theta} \int_X p_\theta(x) f(x) dx = \int_X f(x) \frac{\partial}{\partial \theta} p_\theta(x) dx$$

$$\frac{\partial}{\partial \theta} \log p_\theta(x) = \frac{1}{p_\theta(x)} \frac{\partial}{\partial \theta} p_\theta(x) \Rightarrow \frac{\partial}{\partial \theta} p_\theta(x) = p_\theta(x) \frac{\partial}{\partial \theta} \log p_\theta(x)$$

$$\frac{\partial J}{\partial \theta} = \int_X f(x) p_\theta(x) \frac{\partial}{\partial \theta} \log p_\theta(x) \, dx = \mathbb{E}_{x \sim p_\theta}\left[f(x) \frac{\partial}{\partial \theta} \log p_\theta(x)\right]$$

Approximate the expectation via sampling!

- Can approximate trajectories by sampling some number of trajectories from policy, which would let us approximate expected value

So, in order to get the gradient of J w.r.t theta to perform these gradient ascent steps, we need to compute d[log(p_{theta}(x)] / d(theta).

Define: Let $x = (s_0, a_0, s_1, a_1, \ldots)$ be the sequence of states and actions we get when following policy π_θ. It's random: $x \sim p_\theta(x)$

$$p_\theta(x) = \prod_{t \geq 0} P(s_{t+1} \mid s_t) \pi_\theta(a_t \mid s_t) \Rightarrow \log p_\theta(x) = \sum_{t \geq 0} (\log P(s_{t+1} | s_t) + \log \pi_\theta(a_t | s_t))$$

$$\frac{\partial}{\partial \theta} \log p_\theta(x) = \sum_{t \geq 0} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t | s_t)$$

Transition probabilities of environment. We can't compute this.

Action probabilities of policy. We can are learning this!

So…. putting it all together:

Expected reward under π_θ:

$$J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)]$$

$$\frac{\partial J}{\partial \theta} = \mathbb{E}_{x \sim p_\theta}\left[f(x) \sum_{t \geq 0} \frac{\partial}{\partial \theta} \log \pi_\theta(a_t | s_t)\right]$$

Sequence of states and actions when following policy π_θ

f(x) - rewards we get by executing the trajectories sampled from the policies
The stuff inside the sigma:

- Gradient of predicted action scores w.r.t model weights. Can obtain these values by back propagating through policy model

Intuition:

- When f(x) is high: Increase the probability of the actions we took
- When f(x) is low: Decrease the probability of the actions we took

Use REINFORCE rule

1. Initialize random weights θ
2. Collect trajectories x and rewards f(x) using policy π_θ
3. Compute dJ/dθ
4. Gradient ascent step on θ
5. GOTO 2

Other models:

Actor-Critic

- Actor: predicts actions (like policy gradient)
- Critic: Predicts future rewards we get from taking those actions (like Q-learning)

Model-Based: Learn a model of the world's state transition function $P(s_{t+1} \mid s_t, a_t)$ and then use planning through the model to make decisions

Imitation Learning: Gather data about how experts perform in the environment, learn function to imitate what they do

Inverse RL: Gather data from experts, learn what reward function they seem to be optimizing, then use RL on that reward function

Adversarial Learning: Learn to fool a discriminator that classifies actions as real/fake

Can use RL algorithms to train more complex neural network architectures