Policy Gradient Algorithms

- **Why?**
  - Value functions can be very complex for large problems, while policies have a simpler form. Example: Threshold policies for inventory control.
  - For large problems some form of approximation is needed to represent the value function. In such cases, learning algorithms are not guaranteed to converge. There are very simple examples where value function updates diverge with function approximation.

    Policy gradient methods are well-behaved with function approximation. For e.g. you can view the weights of a neural network as the parameters of the policy and under mild regularity conditions, they can be shown to converge.
  - In many problem domains, you have only partial information about the state. In such cases, value function methods run into a lot of problems. We will visit this, hopefully, later in the semester. Policy gradient methods are "better" behaved even in this scenario.
- **What?**
  - Directly search in policy space
  - Policy depends on some parameters Θ
  - Estimate gradient of performance w.r.t. Θ
  - Update Θ in the direction of gradient
  - Reach a local maximum

Recall:

In Chapter 2 we took a brief look at a policy gradient approach for non-associative problems, like the n-armed bandit task. Here is a brief recap of that.
Computing gradient of performance w.r.t. parameters:

$$\begin{aligned}
\eta(\Theta) &= E(r) \\
&= \sum_a Q^\star(a)\pi(a;\Theta) \\
\nabla\eta(\Theta) &= \sum_a Q^\star(a)\nabla\pi(a;\Theta) \\
&= \sum_a Q^\star(a)\frac{\nabla\pi(a;\Theta)}{\pi(a;\Theta)}\pi(a;\Theta)
\end{aligned}$$

Estimate the gradient from N samples:

$$\hat{\nabla}(\Theta) = \frac{1}{N}\sum_{i=1}^{N} r_i \cdot \underbrace{\frac{\nabla\pi(a_i;\Theta)}{\pi(a_i;\Theta)}}_{\text{LikelihoodRatio}}$$

MC Policy Gradient

- Samples are entire trajectories
  $s_0, a_0, r_1, s_1, a_1, \cdots, s_T$
- Evaluation criterion is the return along the path, instead of immediate rewards
- The gradient estimation equation becomes:

$$\hat{\nabla}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} R_i(s_0) \cdot \frac{\nabla p_i(s_0; \Theta)}{p_i(s_0; \Theta)}$$

  where, $R_i(s_0)$ is the return starting from state s_0 and $p_i(s_0; \Theta)$ is the probability of i-th trajectory, starting from s_0 and using policy given by Θ.

- The "likelihood ratio" in this case evaluates to:

$$\frac{\nabla p_i(s_0; \Theta)}{p_i(s_0; \Theta)} = \sum_{j=0}^{T-1} \frac{\nabla \pi(s_j, a_j; \Theta)}{\pi(s_j, a_j; \Theta)} \tag{1}$$

- Estimate depends on starting state s_0. One way to address this problem is to assume a fixed initial state. Example: Starting board configuration in games like backgammon, chess etc.
- More common assumption is to use the average reward formulation.
- **Recall:**
  - Maximize average reward per time step:

$$\rho^{\pi}(s) = \lim_{N \to \infty} \frac{1}{N} E\left(\sum_{t=0}^{N-1} r_t \mid s_0 = s \right)$$

  - *Unichain assumption:* One set of "recurrent" class of states
  - ρ^{π} is then state independent
  - Recurrent class: Starting from any state in the class, the probability of visiting all the states in the class is 1.

MC Policy Gradient

- Assumption 1: For **every policy** under consideration, the Unichain assumption is satisfied, with **the same set of recurrent states**.
- Pick one recurrent state $i^\star$. Trajectories are defined as starting and ending at this recurrent state.
- Assumption 2: Bounded rewards.

Incremental Update

We can incrementally compute the summation in Equation 1, over one trajectory as follows:

$$\begin{aligned} z_{t+1} &= z_t + \frac{\nabla \pi(s_t, a_t; \Theta)}{\pi(s_t, a_t; \Theta)} \\ R_{t+1} &= R_t + \frac{1}{t+1}\left[r_t - R_t\right] \end{aligned}$$

z_T is known as an *eligibility trace*. Recall the characteristic eligibility term from REINFORCE:

$$\frac{\partial \ln \pi(a_t; \Theta)}{\partial \Theta}.$$

z_T keeps track of this eligibility over time, hence is called a trace.

Simple MC Policy Gradient Algorithm

Algorithm 1 Simple MC Policy Gradient Algorithm

Set $j = 0, R_0 = 0, z_0 = \overline{0}, \Delta_0 = \overline{0}$
for each episode **do**
  for each transition s_t, a_t, r_t, s_{t+1} **do**
    $z_{t+1} = z_t + \frac{\nabla \pi(s_t, a_t; \Theta)}{\pi(s_t, a_t; \Theta)}$
    $R_{t+1} = R_t + \frac{1}{t+1}[r_t - R_t]$
  end for
  $\Delta_{j+1} = \Delta_j + R_T z_T$
  $j = j + 1$
end for
Return Δ_N / N, where N is the number of episodes

Adjust Θ using a simple stochastic gradient ascent rule:

$$\Theta \longleftarrow \Theta + \alpha \frac{\Delta_N}{N}$$

where α is a positive step size parameter.

- The algorithm computes and unbiased estimate of the gradient, but can be very slow due to high variance in the estimates. The variance is related to the "recurrence time" or the episode length. Thus for problems with large state spaces, the variance becomes unacceptably high.
- **Variance reduction techniques:** lead to some bias
  - Truncate summation (eligibility traces)
  - Decay eligibility traces. In this case, the decay rate controls the bias-variance trade off.
  - Actor-Critic methods. These methods use value function estimates to reduce variance. We will look at this in Chapter 6.
  - Employ a set of recurrent states to define episodes, instead of just one $i^\star$.