Markov Decision Processes

- Finite state space $\mathcal{S}$
- Finite action space $\mathcal{A}$
- In state, take action, enter new state, take action, ...
  $s_1, a_1, s_2, a_2, s_3, a_3, \ldots$
- $\Omega = \{(s_1, a_1, s_2, a_2, \ldots) : s_i \in \mathcal{S}, a_i \in \mathcal{A}\}$
- Basic MDP Assumption:

$$P(S_{n+1} = s' | A_n = a, S_n = s,$$
$$A_{n-1} = a_{n-1}, S_{n-1} = s_{n-1}, \ldots, A_1 = a_1, S_1 = s_1) = P_{ss'}(a)$$

- In RL jargon, $P_{ss'}(a)$ is the "environment."

Policies

- Policy: Rule for choosing actions
  - When choosing an action at step n, can depend on entire past $s_1, a_1, s_2, a_2, \cdots, a_{n-1}, s_n$.
  - Can depend on n.
  - Can be randomized. Probability distribution over $\mathcal{A}$:

$$\pi_n(\cdot \mid \text{past up to } n)$$

- Policy $\pi = (\pi_1, \pi_2, \pi_3, \dots)$
  - Stationary policy:

$$\pi_n(\cdot \mid \text{past up to } n) = \pi(\cdot \mid s_n)$$

  - Deterministic policy:

$$\pi : S \to A \quad \pi(s) \in \mathcal{A}$$

Measure on Sample Space Ω

The environment $P_{ss'}(a)$ and the policy π define a probability measure P_π on Ω. For example

$$\begin{aligned} & P_\pi(S_1 = s_1, A_1 = a_1, S_2 = s_2, A_2 = a_2, S_3 = s_3 \mid S_1 = s_1) \\ = & \pi_1(a_1|s_1) P_{s_1 s_2}(a_1) \pi_2(a_2|s_1, a_1, s_2) P_{s_2 s_3}(a_2) \end{aligned}$$

We write E_π for expectation under policy π

Rewards and Criteria

- $R_n = r(S_n, A_n)$
- Expected reward up to time T:

$$V_\pi = E_\pi \left[\sum_{n=1}^{T} r(S_n, A_n) \mid S_1 = s \right]$$

- Expected discounted reward:

$$V_\pi = E_\pi \left[\sum_{n=1}^{\infty} \beta^n r(S_n, A_n) \mid S_1 = s \right]$$

- Expected Average Reward:

$$V_\pi = E_\pi \left[\liminf_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} r(S_n, A_n) \mid S_1 = s \right]$$

- There exists an optimal stationary deterministic policy for discounted and average reward criteria.
- There are many algorithms for finding the optimal π

Issues

- Environment $P_{ss'}(a)$ may not be known
- Curse of dimensionality
- RL to the rescue

RL

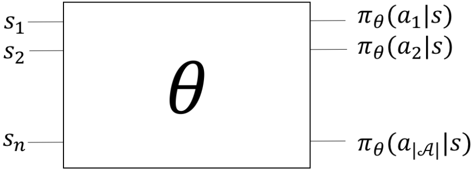
- Many reinforcement learning algorithms are inspired by the Bellman equations for MDPs
- But the Monte Carlo "policy gradient" algorithm does not require the Bellman equation
- So we can jump right into it.

Parameterized Policies: Examples

1. Create features $\Phi_i(s, a)$, $i = 1, \ldots, n$
   Parameters: $\theta_1, \ldots, \theta_n$

$$\pi_\theta(a|s) = \sum_{i=1}^{n} \theta_i \Phi_i(s, a)$$

2. Neural nets: $s = (s_1, \ldots, s_n)$



Goals: Find θ that maximizes $V_{\pi_\theta}(s)$

Policy Gradient Theorem

$$V(\theta) \triangleq E_{\pi_\theta}\left[\sum_{t=1}^{T} r(S_t, A_t) | S_1 = s_1\right]$$

We want to use gradient ascent to maximize $V(\theta)$. To this end, we need to estimate $\nabla_\theta V(\theta)$. Let $Q_t \triangleq \sum_{j=t}^{T} r(S_j, A_j)$

Theorem

$$\nabla_\theta V(\theta) = E_{\pi_\theta}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) Q_t \mid S_1 = s_1\right]$$

Significance: We can estimate $\nabla_\theta V(\theta)$ from episodes drawn from P_{π_θ}.

Sketch of Proof for $T = 2$

$$
\begin{aligned}
\nabla_\theta E_{\pi_\theta}\left[r(S_1, A_1) \mid S_1 = s_1\right] =& \nabla_\theta \sum_{a \in \mathcal{A}} r(s_1, a)\pi(a|s_1) \\
=& \sum_{a \in \mathcal{A}} r(s_1, a)\nabla_\theta \pi(a|s_1) \\
=& \sum_{a \in \mathcal{A}} r(s_1, a)\nabla_\theta \log \pi(a|s_1) \cdot \pi(a|s_1) \\
=& E_{\pi_\theta}\left[r(S_1, A_1)\nabla_\theta \log \pi(A_1|S_1) \mid S_1 = s_1\right]
\end{aligned}
$$

Similar calculations show that

$$
\begin{aligned}
&\nabla_\theta E_{\pi_\theta}\left[r(S_2, A_2) \mid S_1 = s_1\right] \\
=&E_{\pi_\theta}\left[r(S_2, A_2)\nabla_\theta\left(\log\pi(A_1|S_1) + \log\pi(A_2|S_2)\right) \mid S_1 = s_1\right]
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&\nabla_\theta E\left[r(S_1, A_1) + r(S_2, A_2)|S_1 = s_1\right] \\
=&E_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(A_1|S_1)(r(S_1, A_1) + r(S_2, A_2)) \mid S_1 = s_1\right] \\
&+ E_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(A_2|S_2) r(S_2, A_2) \mid S_1 = s_1\right] \\
=&E_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(A_1|S_1) Q_1 \mid S_1 = s_1\right] \\
+&E_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(A_2|S_2) Q_2 \mid S_1 = s_1\right]
\end{aligned}
$$

Monte Carlo Policy Gradient

Algorithm 1 REINFORCE

function $\mathrm{REINFORCE}(\theta)$
  Use π to generate episode $s_1, a_1, s_2, a_2, \ldots, s_T, a_T$.
  for $t = 1, 2, \ldots, T$ **do**
    $q_t = \sum_{j=t}^{T} r(s_j, a_j)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) q_t$
  return θ

Interpretation of $\nabla_\theta \log \pi_\theta(a_t|s_t) q_t$? Move θ towards the direction that gives more of the observed a_t when in state s_t. If the return q_t is big, then make the stepsize big.

Notes:

1. Use back propagation to calculate $\nabla_\theta \log \pi_\theta(a_t|s_t)$
2. Model free; breaks curse of dimensionality
3. $\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t) Q_t$: unbiased estimator for $\nabla_\theta V(\theta)$.
4. But it can have a huge variance. Actor-critic to the rescue
5. Under various conditions, converges w.p.1 to a local optimum for $V(\theta)$

Deep-Reinforcement Learning: Pong to Pixels

- Andrej Karpathy blog
- Action is either UP or DOWN. Reward = +1 if get ball past opponent; -1 if miss the ball; 0 otherwise.
- Neural network: Input is pixel values; one hidden layer with 200 units; output is probability of going UP.
- For each fixed θ, run pong 100 times. For each state/action visited, label it as +1, -1, or 0 depending on the outcome of the episode. For each of these state/actions, we also know $\nabla_\theta \log \pi_\theta(a_t|s_t)$ from backprob.
- Update θ using REINFORCE; then run another 100 episodes
- 130-line Python script using OpenAI Gym's ATARI 2600 Pong
- After 8.000 episodes, able to beat built in AI.