M2177.0043 Introduction to Deep Learning

Lecture 21: Reinforcement learning[1]

Hyun Oh Song[1]

[1] Dept. of Computer Science and Engineering, Seoul National University

June 9, 2020

[1] Many slides and figures adapted David Silver and Justin Johnson

Last time

- Reinforcement learning
- Convergence
- Deep Q-network

Outline

Policy-based reinforcement learning

Policy-based reinforcement learning

- So far, we looked at approximating the value or action-value function using parameters θ,

$$v_\theta(s) \approx v(s), q_\theta(s,a) \approx q(s,a)$$

- A policy was generated from the value function
- However, this q-function can be very complicated to fit perfectly for every state and action pair.
- Policy-based methods directly parameterize the policy

$$\pi_\theta(s,a) = P(a \mid s; \theta)$$

Reinforcement learning objective

Let π denote a stochastic policy $\pi : S \times A \to [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right], \text{where}$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t | s_t), s_{t+1} \sim P(s_{t+1} | s_t, a_t).$$

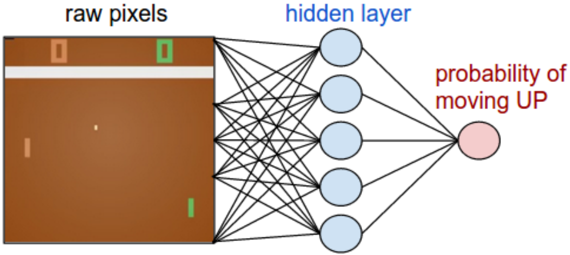


Figure: Example policy network of Atari 2600 Pong.

Value functions

State-action value function Q_π, the value function V_π, and the advantage function A_π can be defined as:

$$
\begin{aligned}
Q_\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}, a_{t+1}, \cdots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+1})\right], \\
V_\pi(s_t) &= \mathbb{E}_{a_t, s_{t+1}, \cdots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+1})\right], \\
A_\pi(s, a) &= Q_\pi(s, a) - V_\pi(s), \text{where} \\
&\quad a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geqslant 0.
\end{aligned}
$$

Objective decomposition

Policy gradient method uses the gradient of the objective function $\eta(\pi_\theta)$ to update the parametrized policy. The objective function can be shown as:

$$
\begin{aligned}
\eta(\pi_\theta) &= \mathbb{E}_{s_0,a_0,\cdots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \\
&= \sum_{s_0}\rho_0(s_0)\sum_{a_0}\pi_\theta(a_0|s_0)\sum_{s_1}P(s_1|s_0,a_0)\cdots\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \\
&= \underbrace{\sum_{s_0}\sum_{a_0}\cdots}_{\sum_\tau}\underbrace{\rho_0(s_0)\prod_{t=0}^{\infty}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t,a_t)}_{P(\tau;\theta)}\underbrace{\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right]}_{R(\tau)} \\
&= \sum_{\tau}P(\tau;\theta)R(\tau)
\end{aligned}
$$

Policy gradient derivation

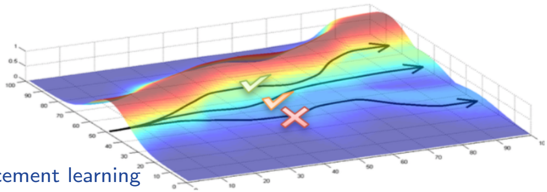
Then, the gradient of the objective function is

$$\begin{aligned}
\nabla_\theta \eta(\pi_\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau) \qquad\qquad \text{(Log-Derivative trick)} \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \left[\sum_{t=0}^{\infty} (\log \pi_\theta(a_t|s_t) + \cancel{\log P(s_{t+1}|s_t, a_t)}) \right] R(\tau) \\
&= \mathbb{E}_{s_0, a_0 \cdots} \left[\left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=0}^{\infty} \gamma^t r(s_t) \right) \right] \\
&\approx \frac{1}{m} \sum_{i=1}^{m} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \right) \left(\sum_{t=0}^{T} \gamma^t r(s_t^i) \right) \right]
\end{aligned}$$

Intuition of policy gradient

$$
\begin{aligned}
\nabla_\theta \eta(\pi_\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&\approx \frac{1}{m} \sum_{i=1}^{m} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left(\sum_{t=0}^{T} \gamma^t r(s_t^i) \right) \right]
\end{aligned}
$$

- Increase probability of paths with positive reward.
- Decrease probability of paths with negative reward.



Expected Grad-Log-Prob Lemma[2]

- EGLP lemma is used extensively throughout the theory of policy gradients.

Lemma 1 (EGLP Lemma)

Suppose that P_θ is a parameterized probability distribution over a random variable x. Then

$$\mathbb{E}_{x \sim P_\theta}[\nabla_\theta \log P_\theta(x)] = 0.$$

Proof.

$$\int_x P_\theta(x) = 1 \implies \nabla_\theta \int_x P_\theta(x) = \nabla_\theta 1 = 0$$

$$0 = \nabla_\theta \int_x P_\theta(x) = \int_x \nabla_\theta P_\theta(x) = \int_x P_\theta(x) \nabla_\theta \log P_\theta(x) = \mathbb{E}_{x \sim P_\theta}[\nabla_\theta \log P_\theta(x)]$$

∎

[2] Taken from https://spinningup.openai.com/

Reward to go policy gradient[3]

- Recall the policy gradient

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

  Taking a step with this gradient pushed up the log-probabilities of each action in proportion to $R(\tau)$, the sum of *all rewards ever obtained*.

- However, agents should really only reinforce actions on the basis of their *consequences*. Rewards obtained before taking an action have no bearing on how good the action was: only rewards that come after

- It turns out that this intuition shows up in math, and we can prove that policy gradient can also be equivalently expressed by (proof left as an exercise)

[3] Taken from https://spinningup.openai.com/

- $$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

- In this form, actions are only reinforced based on rewards obtained after they are taken.

- We will call this form the "reward-to-go-policy gradient", because the sum of rewards after a point in a trajectory,

  $$\widehat{R_t} = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

  is called the "reward-to-go" from that point, and this gradient expression depends on the reward-to-go from state-action pairs.

Baselines in Policy Gradients[4]

- An immediate consequence of the EGLP lemma is that for any function b which only depends on state,

$$\mathbb{E}_{a_t \sim \pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)\right] = 0$$

- This allows us to add or subtract any number of terms like this from our expression for the policy gradient without changing it in expectation

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

- Any function b used in this way is called a *baseline*

[4] Taken from https://spinningup.openai.com/

- The most common choice of baseline is the on-policy value function $V^{\pi}(s_t)$. Recall that this is the average return an agent gets if it starts in state s_t and then acts according to policy π for the rest of its life.

- Empirically, the choice $b(s_t) = V^{\pi}(s_t)$ has the desirable effect of reducing variance in the sample estimate for the policy gradient.

- This results in faster and more stable policy learning. It is also appealing from a conceptual angle: it encodes the intuition that if an agent gets what it expected, it should "feel" neutral about it.

- In practice, $V^{\pi}(s_t)$ cannot be computed exactly, so it has to be approximated. This is usually done with a neural network, $V_{\phi}(s_t)$, which is updated concurrently with the policy (so that the value network always approximates the value function of the most recent policy).

- The simplest method for learning V_{ϕ}, used in most implementations of policy optimization algorithms (including VPG, TRPO, PPO, and A2C), is to minimize a mean-squared-error objective:

$$\phi_k = \arg\min_{\phi} \mathbb{E}_{s_t, \widehat{R}_t \sim \pi_k} \left[\left(V_{\phi}(s_t) - \widehat{R}_t \right)^2 \right],$$

  where π_k is the policy at epoch k. This is done with one or more steps of gradient descent, starting from the previous value parameters ϕ_{k-1}.

Other forms of policy gradient[5]

- What we have seen so far is that the policy gradient has the general form

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

where Φ_t could be any of

$$\Phi_t = R(\tau), \text{ or } \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}), \text{ or } \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

- All of these choices lead to the same expected value for the policy gradient, despite having different variances. It turns out that there are two more valid choices of weights Φ_t which are important to know.

[5] Taken from https://spinningup.openai.com/

Advantage function baseline[6]

1. **On-Policy Action-Value Function** The choice $\Phi_t = Q^{\pi_\theta}(s_t, a_t)$ is also valid.
2. **The Advantage Function** The *advantage function* $A^\pi(s, a)$ corresponding to a policy π describes how much better it is to take a specific action a in state s, over randomly selecting an action according to the current policy $\pi(\cdot \mid s)$, assuming you act according to π forever after. Mathematically the advantage function is defined by

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

   This choice $\Phi_t = A^{\pi_\theta}(s_t, a_t)$ is also valid.

[6] Taken from https://spinningup.openai.com/

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

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

(Law of iterated expectation)

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

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \mathbb{E}_{\tau_{:t}\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\underbrace{\mathbb{E}_{\tau_{t:}\sim\pi_\theta}\left[\widehat{R}_t \mid s_t, a_t\right]}_{=Q_\theta^\pi(s_t,a_t)}\right]$$

- Define $\tau_{:t} = (s_0, a_0, \ldots, s_t, a_t)$ as the trajectory up to time t, and $\tau_{t:}$ as the remainder of the trajectory after that.
- Note in the last step, we use Markov assumption. Conditioning on the entirety of the past up to time t is equal to conditioning on the last time step

Policy gradient update

Thus, in the policy gradient algorithm, parameters are updated with step size α as:

$$\theta_{new} = \theta_{old} + \alpha \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{old}}$$

Limitations of policy gradient

- Sample efficiency is poor.
  - Each environment sample is discarded after only one gradient step.

- Hard to choose appropriate step size.

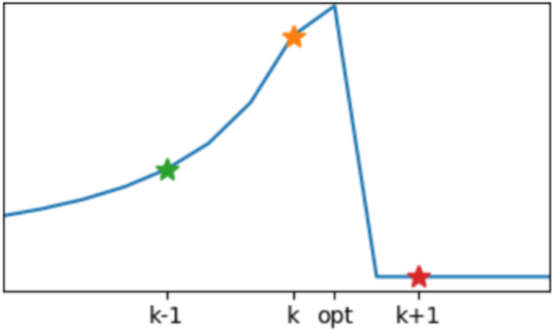


Figure: x-axis: policy parameter, y-axis: objective. A bad step (k→k+1) can lead to performance collapse, which may be hard to recover from.

Trust-region policy optimization (TRPO)[7]

- In the policy gradient algorithm, parameters are updated with step size α as:

$$\theta_{new} = \theta_{old} + \alpha \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{old}}$$

- In TRPO, constrain the stepsize

$$\mathbb{E}_s \left[D_{KL}(\pi_{\theta_{old}}(\cdot \mid s) \parallel \pi_{\theta_{new}}(\cdot \mid s)) \right] \leqslant \delta$$

[7] Schulman et al. "Trust region policy optimization" ICML2015