SDU

5) Policy-Gradient Methods

Melih Kandemir

University of Southern Denmark
Department of Mathematics and Computer Science (IMADA)

Policy gradients

The bottlenecks of the value-based approach:

- When $\mathcal{A}$ is large or continuous, it is not feasible to extract the (epsilon) greedy policy from the action-value function, i.e. by solving

$$a^* \in \arg\max_a Q(s, a).$$

- When Q is learned as a parametric approximation, a small update on its parameters may make big and unforeseeable changes to the corresponding greedy policy.
- It is easier to describe an intended exploration scheme by directly modeling the policy distribution.

The policy-gradient approach suggests expressing the policy distribution as a parametric function π_θ. For given set of feasible parameters Θ, this way we define a feasible set of policies $\Pi = \{\pi_\theta | \theta \in \Theta\}$. Example:

$$\pi(a|s) := \mathcal{N}(a|f_\theta(s), \exp(g_{\theta'}(s)))$$

for neural networks $f_\theta(s)$ and $g_{\theta'}$ with respective weights θ and θ'.

RL as a policy fitting problem

$$\rho_\pi = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t p_0^\top P_\pi^t = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t p_\pi^t$$

is defined as the γ-**discounted state visitation distribution** (a.k.a. occupancy measure) for a policy π. Then we can express the expected return as

$$\eta(\pi) := \sum_{t=0}^{\infty}\gamma^t p_0^\top P_\pi^t r(s_t,\pi(s_t)) = \sum_{t=0}^{\infty}\sum_{s\in\mathcal{S}}\gamma^t p_0^\top P_\pi^t r(s_t,\pi(s_t))\mathbb{I}(s_t=s)$$

$$= \sum_{s\in\mathcal{S}}\sum_{t=0}^{\infty}\gamma^t p_0^\top P_\pi^t r(s,\pi(s)) = \frac{1}{1-\gamma}\sum_{s\in\mathcal{S}}\rho_\pi(s)r(s,\pi(s)).$$

Hence the reinforcement learning problem can be cast as

$$\pi^* \in \arg\max_\pi \sum_{s\in\mathcal{S}}\rho_\pi(s)r(s,\pi(s)).$$

Policy gradient theorem

For a parameteric policy, we have

$$\theta^* \in \arg\max_\theta \sum_{s\in\mathcal{S}} \rho_{\pi_\theta}(s) r(s, \pi_\theta(s)).$$

Updating θ affects the data distribution via ρ_{π_θ}. Hence performing stochastic gradient ascent on $r(s, \pi_\theta(s))$ using samples collected from ρ_{π_θ} will not maximize the intended objective $\eta(\pi_\theta)$. Let us find its correct gradient then.

Theorem

For any parametric policy π_θ, the following holds:

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s\sim p_\pi^t}\left[\int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da \middle| s_0 = s \right] \qquad \square$$

Remark: This theorem only prescribes the gradient computation for a state s. It does not guarantee a policy improvement!

Proof

$$\nabla_\theta V^{\pi_\theta}(s) = \nabla_\theta \int \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da$$

$$= \int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da + \int \pi_\theta(a|s) \nabla_\theta Q^{\pi_\theta}(s,a) da$$

$$= \int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da$$

$$+ \int \pi_\theta(a|s) \nabla_\theta \Big\{ \sum_{s' \in \mathcal{S}} p(s'|s,a) \Big[r(s,a) + \gamma \int \pi_\theta(a'|s') Q^{\pi_\theta}(s',a') da' \Big] \Big\} da$$

$$= \int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da + \int \pi_\theta(a|s) r(s,a) \Big[\nabla_\theta \cancelto{0}{\sum_{s' \in \mathcal{S}} p(s'|s,a)} \Big] da$$

$$+ \gamma \int \pi_\theta(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a) \nabla_\theta \Big\{ \underbrace{\int \pi_\theta(a'|s') Q^{\pi_\theta}(s',a') da'}_{V^{\pi_\theta}(s')} \Big\} da$$

Proof cont'd

$$\nabla_\theta V^{\pi_\theta}(s)$$
$$= \int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da + \gamma \int \pi_\theta(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a) \nabla_\theta V^{\pi_\theta}(s') da$$
$$= \int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da + \gamma \mathbb{E}_{a \sim \pi(\cdot|s)}[\mathbb{E}_{s' \sim p(\cdot|s,a)}[\nabla_\theta V^{\pi_\theta}(s')|a]].$$

Applying recursion k times yields

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_{t=0}^{k-1} \gamma^t \mathbb{E}_{s \sim p_\pi^t}\left[\int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da \middle| s_0 = s\right]$$
$$+ \gamma^k \mathbb{E}_{s_k \sim p_\pi^k}[\mathbb{E}_{a_k \sim \pi(\cdot|s_k)}[\mathbb{E}_{s'_k \sim p(\cdot|s_k,a_k)}[\nabla_\theta V^{\pi_\theta}(s'_k)|s_k,a_k]]|s_0 = s].$$

Because V^{π_θ} is bounded, so is its gradient. Hence the final term shrinks as $k \to \infty$ $\square$

The REINFORCE trick

Remove the integral by the following small trick

$$\begin{aligned}
\nabla_\theta V^{\pi_\theta}(s) &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\int \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} Q^{\pi_\theta}(s,a) da\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\int \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right]\right].
\end{aligned}$$

Approximate $\mathbb{E}_{s\sim\rho_{\pi_\theta}}$, $\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}$, and Q^{π_θ} by Monte Carlo integration. When Q^{π_θ} term is removed, we retrieve **behavioral cloning**, the straightforward way of doing **imitation learning**.

The REINFORCE algorithm

repeat
  $\mathcal{D} := \emptyset$ ▷ Reset the replay buffer at every episode start
  $s := \texttt{env.reset}()$
  repeat
    $a \sim \pi_\theta(\cdot|s)$
    $r, s' := \texttt{env.step}(a)$
    $\mathcal{D} := \mathcal{D} \cup (s, a, r)$ ▷ Maintain an ordered list
    $s := s'$
  until episode end
  $G := 0$
  for do $t = T - 1 \to 0$ ▷ Traverse list in reverse order
    $(s, a, r) := \mathcal{D}[t]$ ▷ Retrieve t'th tuple
    $G := \gamma G + r$
    $\theta := \theta + \alpha G \nabla_\theta \log \pi_\theta(a|s)$
  end for
until convergence

Actor-critic methods

- REINFORCE works only in the episodic case and can update policy network parameters after a full episode is complete. This Monte Carlo scheme will generate a prohibitively high variance on the gradient estimator.
- One can instead maintain a value predictor alongside the policy function and use it for bootstapping, hence speed up learning and reduce estimator variance. RL algorithms that follow this approach are called **actor-critic** methods.
- The policy function is the **actor**, as the actions are taken based on it, and the value estimate is the **critic** as it estimates the consequences of a taken action.

Baselines

Choose an arbitrary function $b(s)$, called a **baseline**, and subtract it from the action-value function

$$\begin{aligned}
&\mathbb{E}_{s\sim\rho_{\pi_\theta}}\Big[\int \nabla_\theta \pi_\theta(a|s)\Big[Q^{\pi_\theta}(s,a) - b(s)\Big]da\Big] \\
&= \mathbb{E}_{s\sim\rho_{\pi_\theta}}\Big[\int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a)da\Big] - \mathbb{E}\Big[\int \nabla_\theta \pi_\theta(a|s) b(s)da\Big] \\
&= \mathbb{E}_{s\sim\rho_{\pi_\theta}}\Big[\int \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a)da\Big] - \mathbb{E}\Big[\nabla_\theta \int \pi_\theta(a|s)da\, b(s)\Big] \quad (\int \pi_\theta(a|s)da \to 0) \\
&= \mathbb{E}_{s\sim\rho_{\pi_\theta}}\Big[\int \nabla_\theta \pi_\theta(a|s)\Big[Q^{\pi_\theta}(s,a)\Big]da\Big]
\end{aligned}$$

Hence subtracting a baseline does not change the policy gradient, i.e. the mean MC estimator will remain unchanged. But we can exploit $b(s)$ to reduce its variance.

Actor-critic methods

Choose the value function as the baseline $V_\psi(s)$ and do REINFORCE trick:

$$\nabla_\theta V^{\pi_\theta}(s) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[\nabla_\theta \log \pi_\theta(a|s)\Big(Q^{\pi_\theta}(s,a) - V_\psi(s)\Big)\right]\right].$$

The corresponding objective for a single sampled sequence is then

$$\theta^*, \psi^* := \arg\max_{\theta,\psi} \sum_{t=0}^{T-1} \Big[G_t - V_\psi(s_t)\Big] \log \pi(a_t|s_t)$$

where G_t is the first-visit MC estimate of s_t. Doing bootstrapping and replacing $G_t - V_\psi(s_t)$ by one-step TD error $\delta_t = r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t)$ via telescoping lemma gives objective

$$\theta^*, \psi^* := \arg\max_{\theta,\psi} \sum_{t=0}^{T-1} \delta_t \log \pi(a_t|s_t).$$

The Advantage Actor-Critic (A2C) Algorithm

See original paper here: [Mnih et al., 2016].

repeat
  $s = \texttt{env.reset}()$
  repeat
    $a \sim \pi_\theta(\cdot|s)$
    $r, s' := \texttt{env.step}(a)$
    $\delta := r + \gamma V_\psi(s') - V_\psi(s)$
    $\theta := \theta + \alpha_a \delta \nabla_\theta \log \pi_\theta(a|s)$
    $\psi := \psi - \alpha_c \delta \nabla_\psi V_\psi(s_t)$
    $s = s'$
  until episode end
until convergence

It is also common to implement the above algorithm with a parametric action-value function $Q_\phi(s, a)$ that is used in the Bellman target calculation. This way, it is possible to estimate the improvement of different policies over the current one via a quantity called an **advantage function** $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$.

Deterministic Policy Gradients

$$\psi_* \leftarrow \arg\min_{\psi} \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \left(r + \gamma \max_{a'} \left\lfloor Q_\psi(s', a') \right\rfloor_{sg} - Q_\psi(s, a) \right)^2$$

where $\lfloor \cdot \rfloor_{sg}$ is called the **stop-gradient** operator that treats its argument as a constant for the optimization problem. When the action space is continuous, the $\max_{a'} Q_\psi(s', a')$ operation is infeasible. Instead we can simply train a separate neural network $\pi_\theta(s)$ to approximate this maximum:

$$\theta^* := \arg\max_{\theta'} \mathbb{E}_{\rho_{\pi_\theta}} [Q_\psi(s, \pi_{\theta'}(s))].$$

Thanks to the stop-gradient operator, ρ_{π_θ} is not affected by the policy update. Hence we can approximate the integral by Monte Carlo. The algorithms that adopt this approach are called **Deterministic Policy Gradient (DPG)** methods. They need to be off-policy as a deterministic policy cannot explore. Common approach is to perturb the policy output with additive noise, i.e. to use $a \leftarrow \pi(s) + \epsilon, \epsilon \sim p(\epsilon)$ during training. The noise is typically a normal distribution with zero mean and scheduled variance.

Overestimation bias and double Q learning

Consider a Q-learning step

$$Q(s,a) := Q(s,a) + \alpha[\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Bellman target}} - Q(s,a)]$$

and assume that $Q(s,a) = Q^\pi(s,a) + \varepsilon(s,a)$ with $\mathbb{E}[\varepsilon(s,a)] = 0$ for all s, a. Even under such a well-behaved type of noise that evens out when many updates are performed, the estimated Bellman target overestimates its actual value:

$$\begin{aligned}
\mathbb{E}_\varepsilon \left[r + \gamma \max_{a'} Q(s',a') \right] &= \mathbb{E}_\varepsilon \left[r + \gamma \max_{a'} \left\{ Q^\pi(s',a') + \varepsilon(s',a') \right\} \right] \\
&= r + \gamma \mathbb{E}_\varepsilon \left[\max_{a'} \left\{ Q^\pi(s',a') + \varepsilon(s',a') \right\} \right] \\
&\geq r + \gamma \max_{a'} \mathbb{E}_\varepsilon \left[Q^\pi(s',a') + \varepsilon(s',a') \right] \\
&= r + \gamma \max_{a'} Q^\pi(s',a').
\end{aligned}$$

Double Q-learning

Maintain two Q-tables. Use a randomly chosen one for Bellman target calculation to update the other one. This way of estimating the Bellman target provably underestimates the true value [van Hasselt, 2010].

$Q_0 := 0$ and $Q_2 := 0$
repeat
  $s :=$ `env.reset`(s)
  repeat
    $\pi_b(\cdot|s) :=$ `eps-greedy`$((Q_0(s,.) + Q_1(s,.))/2, \varepsilon)$
    $a \sim \pi_b(\cdot|s)$
    $r, s' :=$ `env.step`(a)
    $i \sim \text{Bernoulli}(0.5)$
    $\delta := r + \gamma \max_{a'} Q_{1-i}(s', a') - Q_i(s, a)$
    $Q_i(s, a) := Q_i(s, a) + \alpha\delta$
    $s := s'$
  until episode end
until convergence

Min-clipping

Bootstrapping causes an accumulation of both underestimation bias and the estimator variance and double Q-learning does not help reduce the estimator variance. To reduce the variance, do the following:

- Do clipped DQL: $y = r + \gamma \min(Q_0(s', \pi(s')), Q_1(s', \pi(s')))$. This reduces variance because both critics are updated and the target calculation formula favors states with consistent values across the critic pairs.
- Use target networks for critics with parameters $\bar{\psi}_i$ that are trained by Polyak averaging $\bar{\psi}_i := (1 - \tau)\bar{\psi}_i + \tau\psi_i$ for $i = 0, 1$.

Because the policy changes too slowly, we have $|Q - y| \approx 0$ and the gradient signal is weak. As a remedy, use target networks for actors with parameters $\bar{\theta}$ that are trained by Polyak averaging $\theta := (1 - \tau)\bar{\theta} + \tau\theta$ and update the policy net parameters θ less frequently than the critic parameters ψ, e.g. every $k-$th update. The eventual double deep Q learning Bellman target is calculated as

$$y = r + \gamma \lfloor \min(Q_{\bar{\psi}_0}(s', \pi_{\bar{\theta}}(s')), Q_{\bar{\psi}_1}(s', \pi_{\bar{\theta}}(s'))) \rfloor_{sg}.$$

Maximum-Entropy Reinforcement Learning

The training objective is to maximize

$$\begin{aligned} J(\pi) &:= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[r(s,a) + \alpha\mathbb{H}[\pi(\cdot|s)]] \\ &= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[r(s,a) - \alpha\mathbb{E}_{a'\sim\pi}[\log\pi(a'|s)]]] \\ &= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[r(s,a)] - \alpha\mathbb{E}_{s\sim\rho_\pi, a, a'\sim\pi}[\log\pi(a'|s)]] \\ &= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[r(s,a)] - \alpha\mathbb{E}_{s\sim\rho_\pi, a'\sim\pi}[\log\pi(a'|s)]] \\ &= \mathbb{E}_{s\sim\rho_\pi, a\sim\pi}[r(s,a) - \alpha\log\pi(a|s)]] \end{aligned}$$

with respect to the policy π for some $\alpha > 0$. The rationale is to seek for the most exploratory policy that maximizes the expected return.

Policy evaluation with a soft Bellman target

Let us build an actor-critic algorithm for this objective following the principles above. Let us first define a soft Bellman operator to be able to define the Bellman error, that is the build the critic $T^\pi Q^\pi(s,a) := r + \gamma E_{s'}[V(s')]$ with

$$V(s) = E_{a\sim\pi}[Q^\pi(s,a) - \alpha \log \pi(a|s)].$$

Then the Bellman target is

$$T^\pi Q^\pi(s,a) = r + \gamma E_{s',a'}[Q^\pi(s',a') - \alpha \log \pi(a'|s')].$$

Applying the principles introduced above, we attain the following Bellman target:

$$y = r + \gamma \lfloor \mathbb{E}_{a'\sim\pi}[\min(Q_{\bar{\psi}_1}(s', \pi_{\bar{\theta}}(s')), Q_{\bar{\psi}_2}(s',a')) - \alpha \log \pi_\theta(a'|s')] \rfloor_{sg}.$$

Then the optimization problem for the critics can be expressed as

$$\psi_i \leftarrow \arg\min_\psi \mathbb{E}_{(s,a,r,s')\in D}\Big[\Big(r + \gamma \lfloor \mathbb{E}_{a'\sim\pi}[\min(Q_{\bar{\psi}_1}(s', \pi_{\bar{\theta}}(s')), Q_{\bar{\psi}_2}(s',a')) - \alpha \log \pi_\theta(a'|s')] \rfloor_{sg} - Q_\psi(s,a)\Big)^2\Big], \quad i = 1,2$$

Training a soft actor

The target policy distribution can be derived from the learned action-value function as follows:

$$\bar{\pi}(\cdot|s) = \frac{e^{Q_\psi(s,\cdot)/\alpha}}{Z}.$$

where $Z = \int e^{Q_\psi(s,a)/\alpha} da$ is the normalization constant. Then the policy network is trained to minimize its Kullback-Leibler divergence to this target policy

$$\begin{aligned} D_{KL}(\pi_\theta(\cdot|s)||\bar{\pi}(\cdot|s)) &= \mathbb{E}_{a\sim\pi_\theta}\left[\log \pi_\theta(a|\pi) - \log \bar{\pi}(a|s)\right] \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[\log \pi_\theta(a|\pi) - Q_\psi(s,\cdot)/\alpha + Z\right] \\ &\propto \mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a|\pi) - Q_\psi(s,\cdot)\right]. \end{aligned}$$

It is common to choose $Q_\psi(s,a) := \min(Q_{\psi_1}(s,a), Q_{\psi_2}(s,a))$. Then the optimization problem can be expressed as

$$\theta_* \leftarrow \arg\min_\theta \mathbb{E}_{s\sim D}[\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\alpha \log \pi_\theta(a|\pi) - \min(Q_{\psi_1}(s,a), Q_{\psi_2}(s,a))]].$$

Reparameterization

The expectation on the actions is taken with respect to a distribution that depends on θ. Since the gradient of a random sample does not exist, we need to find an alternative expression for the data generating process that takes a sample from a distribution the parameters of which do not depend on θ and transforms this sample by a deterministic function later:

$$a \sim \pi_\theta(\cdot|s) \stackrel{\mathrm{d}}{=} \varepsilon \sim p(\varepsilon), a = f_\theta(s, \varepsilon).$$

This technique is known in the literature as the **reparameterization trick**. It is implemented in PyTorch with the `rsample()` function. Replacing the sampling process of a this way, we get an optimization problem that can be solved by gradient descent

$$\theta_* \leftarrow \arg\min_\theta \mathbb{E}_{s\sim D}[\mathbb{E}_{\varepsilon\sim p}[\alpha \log \pi_\theta(f_\theta(s,\varepsilon)|\pi) - \min(Q_{\psi_1}(s, f_\theta(s,\varepsilon)), Q_{\psi_2}(s, f_\theta(s,\varepsilon))]].$$