Policy Gradient Methods

Milan Straka

November 26, 2018

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research, Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Policy Gradient Methods

Instead of predicting expected returns, we could train the method to directly predict the policy

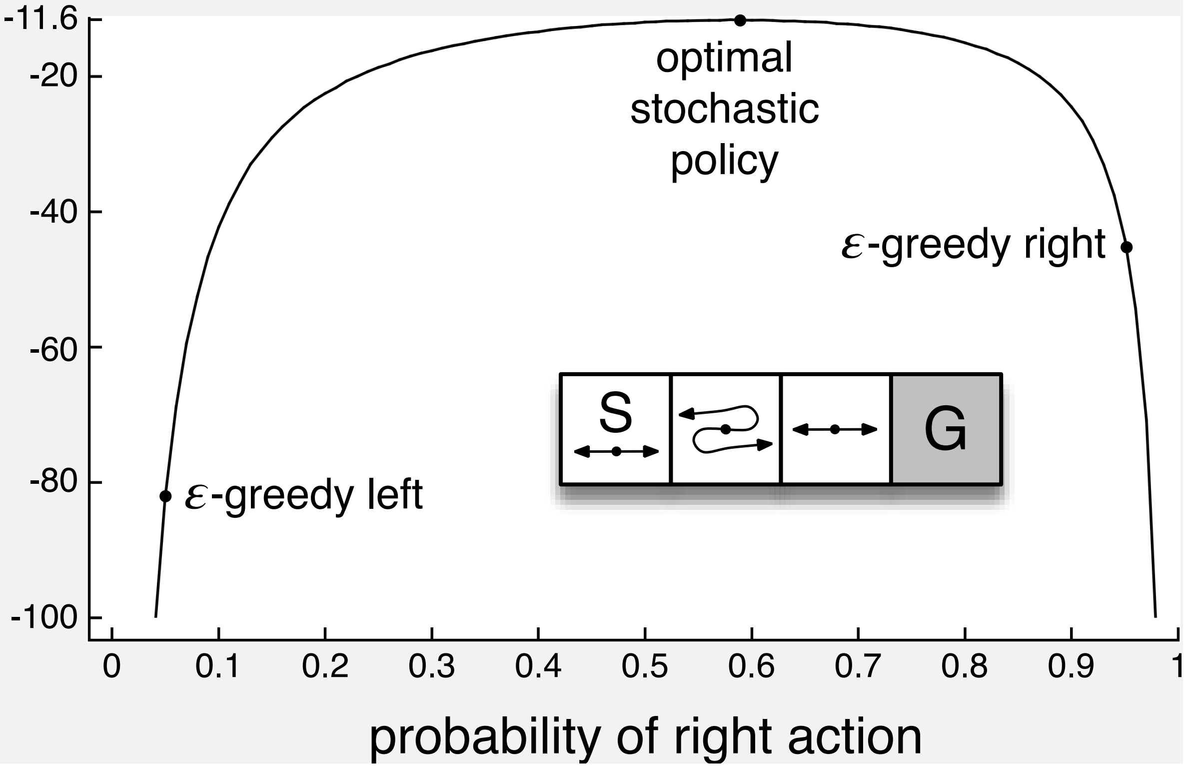
$$\pi(a|s;\boldsymbol\theta).$$

Obtaining the full distribution over all actions would also allow us to sample the actions according to the distribution π instead of just ε-greedy sampling.

However, to train the network, we maximize the expected return $v_\pi(s)$ and to that account we need to compute its *gradient* $\nabla_{\boldsymbol\theta} v_\pi(s)$.

In addition to discarding ε-greedy action selection, policy gradient methods allow producing policies which are by nature stochastic, as in card games with imperfect information, while the action-value methods have no natural way of finding stochastic policies (distributional RL might be of some use though).

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathsf{S})$$



Example 13.1 of "Reinforcement Learning: An Introduction, Second Edition".

Policy Gradient Theorem

Let $\pi(a|s;\boldsymbol\theta)$ be a parametrized policy. We denote the initial state distribution as $h(s)$ and the on-policy distribution under π as $\mu(s)$. Let also $J(\boldsymbol\theta) \stackrel{\text{def}}{=} \mathbb{E}_{h,\pi} v_\pi(s)$.

Then

$$\nabla_{\boldsymbol\theta} v_\pi(s) \propto \sum_{s'\in\mathcal{S}} P(s \rightarrow \ldots \rightarrow s'|\pi) \sum_{a\in\mathcal{A}} q_\pi(s', a) \nabla_{\boldsymbol\theta} \pi(a|s';\boldsymbol\theta)$$

and

$$\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) \propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} q_\pi(s, a) \nabla_{\boldsymbol\theta} \pi(a|s;\boldsymbol\theta),$$

where $P(s \rightarrow \ldots \rightarrow s'|\pi)$ is probability of transitioning from state s to s' using 0, 1, … steps.

$$\begin{aligned}
\nabla v_\pi(s) &= \nabla \Big[\sum\nolimits_a \pi(a|s;\boldsymbol\theta) q_\pi(s,a) \Big] \\
&= \sum\nolimits_a \Big[\nabla \pi(a|s;\boldsymbol\theta) q_\pi(s,a) + \pi(a|s;\boldsymbol\theta) \nabla q_\pi(s,a) \Big] \\
&= \sum\nolimits_a \Big[\nabla \pi(a|s;\boldsymbol\theta) q_\pi(s,a) + \pi(a|s;\boldsymbol\theta) \nabla \big(\sum\nolimits_{s'} p(s'|s,a)(r + v_\pi(s')) \big) \Big] \\
&= \sum\nolimits_a \Big[\nabla \pi(a|s;\boldsymbol\theta) q_\pi(s,a) + \pi(a|s;\boldsymbol\theta) \big(\sum\nolimits_{s'} p(s'|s,a) \nabla v_\pi(s') \big) \Big]
\end{aligned}$$

We now expand $v_\pi(s')$.

$$\begin{aligned}
&= \sum\nolimits_a \Big[\nabla \pi(a|s;\boldsymbol\theta) q_\pi(s,a) + \pi(a|s;\boldsymbol\theta) \Big(\sum\nolimits_{s'} p(s'|s,a) \Big(\\
&\qquad \sum\nolimits_{a'} \Big[\nabla \pi(a'|s';\boldsymbol\theta) q_\pi(s',a') + \pi(a'|s';\boldsymbol\theta) \Big(\sum\nolimits_{s''} p(s''|s',a') \nabla v_\pi(s'') \Big) \Big) \Big) \Big]
\end{aligned}$$

Continuing to expand all $v_\pi(s'')$*, we obtain the following:*

$$\nabla v_\pi(s) = \sum_{s' \in \mathcal{S}} P(s \to \ldots \to s'|\pi) \sum_{a \in \mathcal{A}} q_\pi(s',a) \nabla_{\boldsymbol\theta} \pi(a|s';\boldsymbol\theta).$$

Proof of Policy Gradient Theorem

Recall that the initial state distribution is $h(s)$ and the on-policy distribution under π is $\mu(s)$. If we let $\eta(s)$ denote the number of time steps spent, on average, in state s in a single episode, we have

$$\eta(s) = h(s) + \sum_{s'} \eta(s') \sum_a \pi(a|s')p(s|s',a).$$

The on-policy distribution is then the normalization of $\eta(s)$:

$$\mu(s) \stackrel{\text{def}}{=} \frac{\eta(s)}{\sum_{s'} \eta(s')}.$$

The last part of the policy gradient theorem follows from the fact that $\mu(s)$ is

$$\mu(s) = \mathbb{E}_{s_0 \sim h(s)} P(s_0 \rightarrow \ldots \rightarrow s|\pi).$$

REINFORCE Algorithm

The REINFORCE algorithm (Williams, 1992) uses directly the policy gradient theorem, maximizing $J(\boldsymbol\theta) \stackrel{\text{def}}{=} \mathbb{E}_{h,\pi} v_\pi(s)$. The loss is defined as

$$\begin{aligned}
-\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) &\propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} q_\pi(s, a) \nabla_{\boldsymbol\theta} \pi(a|s; \boldsymbol\theta) \\
&= \mathbb{E}_{s\sim\mu} \sum_{a\in\mathcal{A}} q_\pi(s, a) \nabla_{\boldsymbol\theta} \pi(a|s; \boldsymbol\theta).
\end{aligned}$$

However, the sum over all actions is problematic. Instead, we rewrite it to an expectation which we can estimate by sampling:

$$-\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) \propto \mathbb{E}_{s\sim\mu} \mathbb{E}_{a\sim\pi} q_\pi(s, a) \nabla_{\boldsymbol\theta} \ln \pi(a|s; \boldsymbol\theta),$$

where we used the fact that

$$\nabla_{\boldsymbol\theta} \ln \pi(a|s; \boldsymbol\theta) = \frac{1}{\pi(a|s; \boldsymbol\theta)} \nabla_{\boldsymbol\theta} \pi(a|s; \boldsymbol\theta).$$

REINFORCE therefore minimizes the loss

$$-\mathbb{E}_{s\sim\mu}\mathbb{E}_{a\sim\pi} q_\pi(s,a)\nabla_{\boldsymbol\theta} \ln \pi(a|s;\boldsymbol\theta),$$

estimating the $q_\pi(s,a)$ by a single sample.

Note that the loss is just a weighted variant of negative log likelihood (NLL), where the sampled actions play a role of gold labels and are weighted according to their return.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol\theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol\theta)$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
        $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha G \nabla \ln \pi(A_t|S_t,\boldsymbol\theta)$

Modification of Algorithm 13.3 of "Reinforcement Learning: An Introduction, Second Edition".

REINFORCE with Baseline

The returns can be arbitrary – better-than-average and worse-than-average returns cannot be recognized from the absolute value of the return.

Hopefully, we can generalize the policy gradient theorem using a baseline $b(s)$ to

$$\nabla_{\boldsymbol\theta} J(\boldsymbol\theta) \propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} \big(q_\pi(s,a) - b(s)\big) \nabla_{\boldsymbol\theta} \pi(a|s;\boldsymbol\theta).$$

The baseline $b(s)$ can be a function or even a random variable, as long as it does not depend on a, because

$$\sum_a b(s) \nabla_{\boldsymbol\theta} \pi(a|s;\boldsymbol\theta) = b(s) \sum_a \nabla_{\boldsymbol\theta} \pi(a|s;\boldsymbol\theta) = b(s) \nabla 1 = 0.$$

A good choice for $b(s)$ is $v_\pi(s)$, which can be shown to minimize variance of the estimator. Such baseline reminds centering of returns, given that

$$v_\pi(s) = \mathbb{E}_{a\sim\pi} q_\pi(s, a).$$

Then, better-than-average returns are positive and worse-than-average returns are negative.

The resulting $q_\pi(s, a) - v_\pi(s)$ function is also called an *advantage function*

$$a_\pi(s, a) \stackrel{\text{def}}{=} q_\pi(s, a) - v_\pi(s).$$

Of course, the $v_\pi(s)$ baseline can be only approximated. If neural networks are used to estimate $\pi(a|s; \boldsymbol\theta)$, then some part of the network is usually shared between the policy and value function estimation, which is trained using mean square error of the predicted and observed return.

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol\theta)$
Input: a differentiable state-value function parameterization $\hat v(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol\theta} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol\theta)$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$ $\quad (G_t)$
        $\delta \leftarrow G - \hat v(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat v(S_t,\mathbf{w})$
        $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha^{\boldsymbol\theta} \delta \nabla \ln \pi(A_t|S_t, \boldsymbol\theta)$

Modification of Algorithm 13.4 of "Reinforcement Learning: An Introduction, Second Edition".

REINFORCE with Baseline

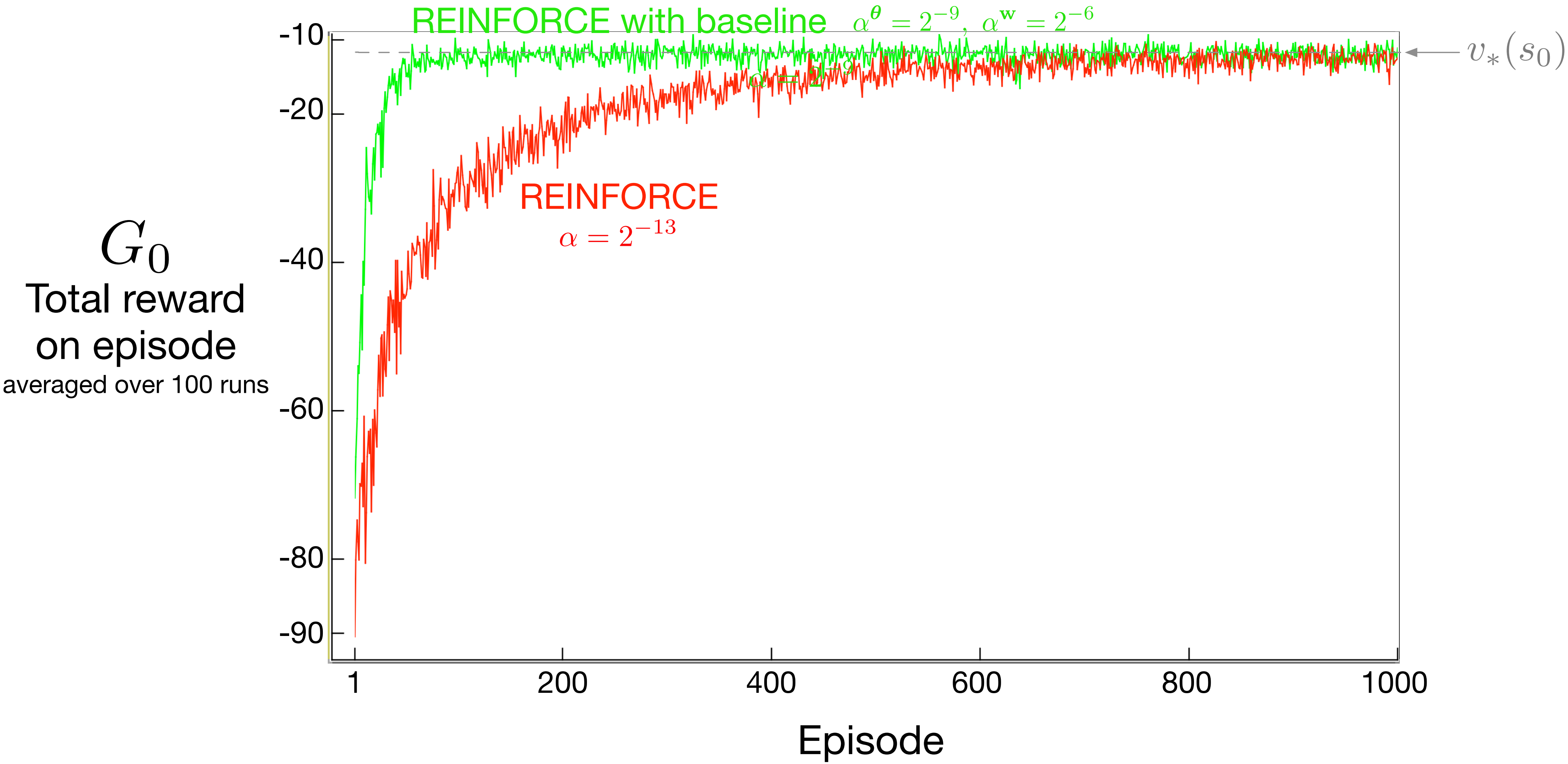


Figure 13.2 of "Reinforcement Learning: An Introduction, Second Edition".

It is possible to combine the policy gradient methods and temporal difference methods, creating a family of algorithms usually called *actor-critic* methods.

The idea is straightforward – instead of estimating the episode return using the whole episode rewards, we can use n-step temporal difference estimation.

Actor-Critic

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol\theta)$
Input: a differentiable state-value function parameterization $\hat v(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol\theta} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)

  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol\theta)$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat v(S',\mathbf{w}) - \hat v(S,\mathbf{w})$ (if S' is terminal, then $\hat v(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat v(S,\mathbf{w})$
    $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha^{\boldsymbol\theta} \delta \nabla \ln \pi(A|S, \boldsymbol\theta)$
    $S \leftarrow S'$

Modification of Algorithm 13.5 of "Reinforcement Learning: An Introduction, Second Edition".