1 Policy Gradient Methods

In this section we take an approach that is different to the action-value methods that we have considered previously. We continue the function approximation scheme, but attempt to learn a *parameterised policy* $\pi(a|s, \boldsymbol{\theta})$ where $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ is the policy's *parameter vector*. Our methods might also learn a value function, but the policy will provide a probability distribution of possible actions without directly consulting the value function as we did previously.

We will learn the policy parameter by *policy gradient methods*. These are gradient methods based on some scalar performance measure $J(\boldsymbol{\theta})$. In particular, performance is maximised by *gradient ascent* using some stochastic estimate of J, $\widehat{\nabla J(\boldsymbol{\theta}_t)}$, whose expectation approximates $\mathbb{E}[\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)]$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta} + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}.$$

Methods that also learn a value function are called *actor-critic* methods. Actor is in reference to the learn policy, while critic is in reference to the learned (usually state-) value function.

1.1 Policy Approximation and its Advantages

In policy gradient methods, the policy can be parameterised by any differentiable function of the parameter $\boldsymbol{\theta}$. We generally require $\pi \in (0, 1)$ to be defined on the open interval to ensure exploration.

For action-spaces that are discrete and not too large, it is common to learn a preference function $h(s, a, \boldsymbol{\theta}) \in R$ and then take a soft-max to get the policy

$$\pi(a|s, \boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}.$$

We call this type of parameterisation *soft-max in action preferences*. (Note the homomorphism: preferences add, while probabilities multiply.) We can learn the preferences any way we like, be it linear or using a deep learning.

Some advantages of policy parameterisation:

- Action-value methods, such as ϵ-greedy action selection, can result give situations in which an arbitrarily small change in the action-values completely changes the policy.
- The soft-max method will approach a deterministic policy over time. If we used action-values then these would approach their (finite) true values, leading to finite probabilities (with the soft-max). Action preferences do not necessarily converge, but instead are driven to produce an optimal stochastic policy.
- In some problems, the best policy may be stochastic. Action-value methods have no natural way of approximating this, whereas it is embedded in this scheme.
- Often the most important reason for choosing a policy based learning method is that policy parameterisation provides a good way to inject prior knowledge into the system.

1.2 The Policy Gradient Theorem

The episodic and continuing cases of the policy gradient theorem have different proofs (since they use different formulations of the expected reward). Here we will first focus on the episodic case, but the results carry over the same.

In the episodic case the performance function is the true value of the start state under the current policy

$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s_0).$$

In the following we assume no discounting ($\gamma = 1$), but this can be inserted by making the requisite changes (see exercises).

The success of the *policy gradient theorem* is that it gives a gradient of the performance function that does not include derivatives of the state distribution. The result for the episodic case is as follows and is derived in the box shown below

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}).$$

Proof of the Policy Gradient Theorem (episodic case)

With just elementary calculus and re-arranging of terms, we can prove the policy gradient theorem from first principles. To keep the notation simple, we leave it implicit in all cases that π is a function of $\boldsymbol{\theta}$, and all gradients are also implicitly with respect to $\boldsymbol{\theta}$. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \text{for all } s \in \mathcal{S} && \text{(Exercise 3.18)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] && \text{(product rule of calculus)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r\,|\,s,a) \big(r + v_\pi(s')\big) \Big] \\
& && \text{(Exercise 3.19 and Equation 3.2)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'\,|\,s,a) \nabla v_\pi(s') \Big] && \text{(Eq. 3.4)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'\,|\,s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} \big[\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''\,|\,s',a') \nabla v_\pi(s'') \big] \Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

after repeated unrolling, where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π. It is then immediate that

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(box page 199)} \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Eq. 9.3)} \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Q.E.D.)}
\end{aligned}
$$

1.3 REINFORCE: Monte Carlo Policy Gradient

We now attempt to learn a policy by stochastic gradient ascent on the performance function. To begin, the policy gradient theorem can be stated as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\sum_{a} q_{\pi}(S_t, a)\nabla_{\boldsymbol{\theta}}\pi(a|S_t, \boldsymbol{\theta})\right].$$

The *all-actions* method simply samples this expectation to give the update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \sum_{a} \hat{q}(S_t, a, \boldsymbol{w})\nabla_{\boldsymbol{\theta}}\pi(a|S_t, \boldsymbol{\theta}).$$

The classical REINFORCE algorithm involves only A_t, rather than a sum over all actions. We proceed

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} &= \mathbb{E}_{\pi}\left[\sum_{a} \pi(a|S_t, \boldsymbol{\theta}) q_{\pi}(S_t, a)\frac{\nabla_{\boldsymbol{\theta}}\pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})}\right] && (1)\\
&= \mathbb{E}_{\pi}\left[q_{\pi}(S_t, A_t)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}\right] && (2)\\
&= \mathbb{E}_{\pi}\left[G_t\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}\right], && (3)
\end{aligned}$$

which yields the REINFORCE update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta}). \tag{4}$$

This update moves the parameter vector in the direction of increasing the probability of the action taken proportional to the return and inversely proportional to the probability of the action. It uses the complete return from time t, so in this sense is a Monte Carlo algorithm. We refer to the quantity

$$\nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$$

as the *eligibility vector*. Pseudocode is given in the box below (complete with discounting). Convergence to a local optimum is guaranteed under the standard stochastic approximation conditions for decreasing α. However, since it is a Monte Carlo method, it will likely have high variance which will slow learning.

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 $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

1.4 REINFORCE with Baseline

The policy gradient theorem can be generalised to incorporate a comparison to a *baseline* value $b(s)$ for each state

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \left(q_\pi(s,a) - b(s)\right) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}). \tag{5}$$

The baseline can be a random variable, as long as it doesn't depend on a. The update rule then becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t \nabla_{\boldsymbol{\theta}} - b(S_t)\right) \log \pi(A_t|S_t,\boldsymbol{\theta}). \tag{6}$$

The idea of the baseline is to reduce variance – by construction it has no impact on the expected update.

A natural choice for the baseline is a learned state-value function $\hat{v}(S_t, \boldsymbol{w})$. Pseudocode for Monte Carlo REINFORCE with this baseline (also learned by MC estimation) is given in the box below.

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 $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

This algorithm has two step sizes $\alpha^{\boldsymbol{\theta}}$ and $\alpha^{\boldsymbol{w}}$. Choosing the step size for the value estimates is relatively easy, for instance in the linear case we have the rule of thumb $\alpha^{\boldsymbol{w}} = 1/\mathbb{E}\left[||\nabla_{\boldsymbol{w}} \hat{v}(S_t,\boldsymbol{w})||^2_\mu\right]$. It is much less clear how to set the step size for the policy parameters.

1.5 Actor-Critic Methods

Although REINFORCE with baseline can use an estimated value function, it is not an actor-critic method because it does not incorporate value estimates through bootstrapping.

We present here a one-step actor-critic method that is an analog of TD(0), Sarsa(0) and Q-learning. We replace the full return of REINFORCE with a bootstrapped one-step return:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{v}(S_t,\boldsymbol{w})\right) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{7}$$

$$= \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \gamma \hat{v}(S_{t+1},\boldsymbol{w}) - \hat{v}(S_t,\boldsymbol{w})\right) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{8}$$

$$= \boldsymbol{\theta}_t + \alpha \delta_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \tag{9}$$

with δ_t as the one-step TD error. The natural method to learn the state-value function in this case would be semi-gradient TD(0). Pseudocode is given in the boxes below for this algorithm and a

sister algorithm using eligibility traces.

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)
  $I \leftarrow 1$
  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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Actor–Critic with Eligibility Traces (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: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; 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)
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
  $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
  $I \leftarrow 1$
  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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

1.6 Policy Gradient for Continuing Problems

For continuing problems we need a different formulation. We choose as our performance measure the average rate of reward per time step:

$$J(\boldsymbol{\theta}) = r(\pi) = \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi] \tag{10}$$

$$= \lim_{t\to\infty} \mathbb{E}[R_t | S_0, A_{0:t-1} \sim \pi] \tag{11}$$

$$= \sum_s \mu(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r, \tag{12}$$

where μ is the steady distribution under π, $\mu(s) = \lim_{t\to\infty} P(S_t = s | A_{0:t} \sim \pi)$ which we assume to exist and be independent of S_0 (ergodicity). Recall that this is the distribution that is invariant under action selections according to π:

$$\sum_s \mu(s) \sum_a \pi(a|s, \boldsymbol{\theta}) p(s'|s, a) = \mu(s').$$

We also define the values with respect to the differential return:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots .$$

With these changes the policy gradient theorem remains true (proof given in the book). The forward and backward view equations also remain the same. Pseudocode for the actor-critic algorithm in the continuing case is given below.

Actor–Critic with Eligibility Traces (continuing), 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: $\lambda^{\mathbf{w}} \in [0, 1]$, $\lambda^{\boldsymbol{\theta}} \in [0, 1]$, $\alpha^{\mathbf{w}} > 0$, $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\bar{R}} > 0$
Initialize $\bar{R} \in \mathbb{R}$ (e.g., to 0)
Initialize state-value weights $\mathbf{w} \in \mathbb{R}^d$ and policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
Initialize $S \in \mathcal{S}$ (e.g., to s_0)

$\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
$\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
Loop forever (for each time step):
  $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
  Take action A, observe S', R
  $\delta \leftarrow R - \bar{R} + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}} \delta$
  $\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S, \mathbf{w})$
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + \nabla \ln \pi(A|S, \boldsymbol{\theta})$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
  $S \leftarrow S'$

1.7 Policy Parameterisation for Continuous Actions

We simply define a continuous (parameterised) probability distribution over the actions, then do the gradient ascent as above.