1 Policy Gradient Methods

Action-valued method: learns the values of actions, then select actions based on estimated action values. If no action-value estimate, then no policy.

On the contrary, a **parameterized policy** can select actions w/o consulting a value function (a value function may still be used to learn the policy parameter, but isn't required for action selection).

Policy gradient method: Let $J(\boldsymbol{\theta})$ be the scalar performance measure w.r.t policy parameter. We want to maximize performance (so gradient ascent, note the **plus** sign):

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

where $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure w.r.t θ_t. PG methods not necessarily learn an approximate value function.

actor-critic method: learn approximations of both policy (actor) and value functions (critic).

- episodic case: performance = the value of the start state under the parameterized policy $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(S_0)$
- continuing case: performance = average reward rate

1.1 Policy Approximation and its Advantages

In PG methods, policy can be parameterized in any way, as long as the policy $\pi(a|s,\theta)$ is differentiable w.r.t parameters.

Definition 1.1 ***soft-max in action preferences***: *action preference* $h(s,a,\boldsymbol{\theta})$ *can be parameterized arbitrarily.*

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

For example, the preference can be linear $h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{x}(s,a)$

Advantages of PG:

- Action-value methods have no natural way of finding stochastic optimal policies, whereas PG methods can
- policy may be simpler to approximate
- policy parameterization is sometimes a good way to impart prior knowledge
- action probabilities change smoothly
- stronger convergence guarantees for PG than action-value methods

1.2 The Policy Gradient Theorem

Definition 1.2 ***policy gradient theorem***

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

In the episodic case, the constant of proportionality is the average length of an episode, and in the continuing case it is 1 (which is equality actually). μ is the distribution under π.

1.3 REINFORCE: Monte Carlo Policy Gradient

The RHS of the PG theorem is a sum over states weighted by how often the states occur under the target policy π.

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\
&= \mathbb{E}_\pi[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta})] \\
&= \mathbb{E}_\pi[\sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}] \\
&= \mathbb{E}_\pi[q_\pi(S_t,A_t) \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}] \\
&= \mathbb{E}_\pi[G_t \frac{\nabla \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})}]
\end{aligned}$$

Therefore, we have the REINFORCE update:

$$\begin{aligned}\boldsymbol{\theta_{t+1}} &\doteq \boldsymbol{\theta_t} + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta_t})}{\pi(A_t|S_t, \boldsymbol{\theta_t})} \\ &= \boldsymbol{\theta} + \alpha G_t \nabla ln \pi(A_t|S_t, \boldsymbol{\theta})\end{aligned}$$

The vector is the direction in parameter space that most increases the probability of repeating the action A_t on future visits to state S_t. The update increases the parameter vector in this direction proportional to the return (because it causes the parameter to move most in the directions that favor actions that yield the highest return), and inversely proportional to the action probability (because we want to prevent same actions to not be constantly selected, resulting in local optimum).

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 $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, ..., T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla ln \pi(A_t|S_t, \boldsymbol{\theta})$

1.4 REINFORCE with Baseline

We include an arbitrary baseline $b(s)$ (as long as it doesn't vary with a) to (1) strictly generalize REINFORCE; (2) reduce variances w/o introducing bias; (3) dynamically adjust to different states for action selection, (and thus faster learning).

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla \pi(a|s, \boldsymbol{\theta})$$

Therefore, the update rule is

$$\begin{aligned}\boldsymbol{\theta_{t+1}} &\doteq \boldsymbol{\theta_t} + \alpha(G_t - b(S_t))\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta_t})}{\pi(A_t|S_t,\boldsymbol{\theta_t})} \\ &= \boldsymbol{\theta} + \alpha(G_t - b(S_t))\nabla ln\pi(A_t|S_t,\boldsymbol{\theta})\end{aligned}$$

A natural choice for the baseline is an estimate of the state value, $\hat{v}(S_t, \boldsymbol{w})$.

REINFORCE with baseline (episodic) for estimating π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\boldsymbol{w})$
Algorithm parameter: step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\boldsymbol{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weight $\boldsymbol{w} \in \mathbb{R}^d$
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, ..., T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{v}(S_t,\boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta\nabla\hat{v}(S_t,\boldsymbol{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\gamma^t\delta\nabla ln\pi(A_t|S_t,\boldsymbol{\theta})$

1.5 Actor-Critic Methods

why go from baseline methods to AC? In REINFORCE with baseline, the learned $\hat{v}(s)$ estimates only the first state of each state transition. It is made prior to the transition's action and thus cannot be used to assess that action. In AC methods, the $\hat{v}(s)$ is also applied to the second state of the transition (which forms a one-step return), so we can assessing that action. This way of assessing actions is called a *critic*.

Note that one-step return introduces bias, but it is often superior to the actual return for lower variance and easier computation. We can fix the bias with n-step returns and eligibility traces.

Definition 1.3 ***one-step AC*** *replace the full return of REINFORCE with the*

one-step return. The baseline of REINFORCE may be a $\hat{v}(s)$, but AC must use $\hat{v}(s)$ as the baseline.

$$\begin{aligned}\boldsymbol{\theta_{t+1}} &\doteq \boldsymbol{\theta_t} + \alpha(\underbrace{G_{t:t+1}}_{\textit{one-step return}} - \hat{v}(S_t, \boldsymbol{w}))\frac{\nabla\pi(A_t|S_t, \boldsymbol{\theta_t})}{\pi(A_t|S_t, \boldsymbol{\theta_t})} \\ &= \boldsymbol{\theta_t} + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}))\frac{\nabla\pi(A_t|S_t, \boldsymbol{\theta_t})}{\pi(A_t|S_t, \boldsymbol{\theta_t})}\end{aligned}$$

Since we are using one-step return, the one-step AC is full online and incremental.

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \boldsymbol{w})$
Algorithm parameter: step size $\alpha^{\boldsymbol{\theta}} > 0, \alpha^{\boldsymbol{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weight $\boldsymbol{w} \in \mathbb{R}^d$
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S != $terminal$ (for each time step):
    $A \sim \pi(\cdot|S, \theta)$
    Take action A, observe S', R
    $\delta \leftarrow R + \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta\nabla\hat{v}(S_t, \boldsymbol{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla ln\pi(A_t|S_t, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

AC with Eligibility Traces (episodic) is available in the book. We just declare eligibility trace vectors for both $\boldsymbol{\theta}$ and $\boldsymbol{w}$, and a trace-decay rates λ. We decay the the γ for both ET vectors at each update.

1.6 Policy Gradient for Continuing Problems

Recall in chap 10.3, we introduce the average reward for continuing problems and a steady-state distribution $\mu(s)$.

$$\sum_s \mu(s) \sum_a \pi(a|s, \boldsymbol{\theta}) p(s'|s, a) = \mu(s') \quad \forall s' \in \S$$

The pseudocode of AC with ET (continuing) is similar to AC with ET (episodic), except that we (1) replace reward with (reward - mean reward), and the mean reward is update as $\overline{R} \leftarrow \overline{R} + \alpha^{\overline{R}}\delta$; (2) remove γ decay.

1.7 Policy Parameterization for Continuous Actions

Instead of computing learned probabilities for each of the many actions, we instead learn statistics of the probability distribution.

Recall the probability density function is:

$$p(s) \doteq \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

To use PDF in policy parameterization

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} exp(-\frac{(a-\mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2})$$

where we often declare that

$$\mu(s, \boldsymbol{\theta}) \doteq \boldsymbol{\theta}_\mu^T \boldsymbol{x}_\mu(s)$$

$$\sigma(s, \boldsymbol{\theta}) \doteq exp(\boldsymbol{\theta}_\sigma^T \boldsymbol{x}_\sigma(s)) > 0$$

1.8 Summary

- In contrary to action-value methods, policy gradient methods perform action selections w/o consulting action-value estimates

- Advantages of learn&store policy parameters
  - learn specific probabilities for taking the actions
  - learn appropriate levels of exploration
  - approach deterministic policies asymptotically
  - can handle continuous action spaces
- policy gradient theorem provides a strong theoretical foundation
- REINFORCE directly follows PG theorem
- REINFORCE w/ baseline reduces variance w/o new bias
- AC, critic introduces bias, but bootstrapping (TD) is superior than MC (substantially reduce variance)

1.9 Learning Objectives (UA RL MOOC)

Lesson 1: Learning Parameterized Policies

1. Understand how to define policies as parameterized functions

Parameterized policy: $\pi(a|s, \boldsymbol{\theta})$. Parameterized value function: $\hat{q}(s, a, \boldsymbol{w})$

Constraints on the policy parameterization

- $\pi(a|s, \boldsymbol{\theta}) \geq 0 \quad \forall a \in \mathcal{A}, s \in \S$
- $\sum_{a \in \mathcal{A}} \pi(a|s, \boldsymbol{\theta}) = 1 \quad \forall s \in \S$

2. Define one class of parameterized policies based on the softmax function

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b \in \mathcal{A}} e^{h(s,a,\boldsymbol{\theta})}}$$

h function is action preference (it can be parameterized in any way), the exponential function guarantees the probability is positive for each action. The denominator normalizes the output of each action s.t. the sum over actions is one. Note action preferences != action values; only the difference between preferences is important.

3. Understand the advantages of using parameterized policies over action-

value based methods

- Parameterized policies can autonomously decrease exploration over time. Specifically, the policy can start off stochastic to guarantee exploration; as learning progresses, the policy can naturally converge towards a deterministic greedy policy
- They can avoid failures due to deterministic policies with limited function approximation
- Sometimes the policy is less complicated than the value function

Lesson 2: Policy Gradient for Continuing Tasks

4. Describe the objective for policy gradient algorithms

Our objective:

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

- $\sum_{s',r} p(s', r|s, a) r$ is $\mathbb{E}[R_t|S_t = s, A_t = a]$ expected reward if we start in state s and take action a
- $\sum_a \pi(a|s, \boldsymbol{\theta}) \sum_{s',r} p(s', r|s, a) r$ is $\mathbb{E}[R_t|S_t = s]$ expected reward of state s
- $r(\pi)$ is $\mathbb{E}_\pi[R_t]$ overall average reward by considering the fraction of time we spend in state s under policy π.

Optimizing the average reward objective (policy gradient method):

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

The main challenge is modifying policy changes the distribution $\mu(s)$; in contrast, recall in $\overline{VE}$ objective, the distribution is fixed. We do gradient ascent for PG, while gradient descent in optimizing the mean squared value error.

5. Describe the results of the policy gradient theorem

Note that chain rule of gradient requires to estimate $\nabla\mu(s)$, which is difficult to estimate since it depends on a long-term interaction between the policy and the environment. The PG theorem proves we don't need that, and following is

our result:

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$$

$\sum_a \nabla \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$. This is a sum over the gradients of each action probability, weighted by the value of the associated action. $r(\pi)$ takes the above expression and sum that over each state. This gives the direction to move the policy parameters to most rapidly increase the overall average reward.

6. Understand the importance of the policy gradient theorem

Lesson 3: Actor-Critic for Continuing Tasks

7. Derive a sample-based estimate for the gradient of the average reward objective

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta_t})}{\pi(A_t|S_t, \boldsymbol{\theta_t})} q_\pi(S_t, A_t)$$

8. Describe the actor-critic algorithm for control with function approximation, for continuing tasks

Full algorithm see chap 13.6

- actor: parameterized policy
- critic: value functions, evaluating the actions selected by the actor

In the update rule, we don't have access to q_π, so we have to approximate it. For example, one-step bootstrap return TD method:

$$q_\pi(s, a) = R_{t+1} - \overline{R} + \hat{v}(S_{t+1}, \boldsymbol{w})$$

. To further improve, we subtract a baseline to reduces the update variance (results in faster learning):

$$q_\pi(s, a) = R_{t+1} - \overline{R} + \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})$$

. Note this is equals the TD error δ.

Lesson 4: Policy Parameterizations

9. Derive the actor-critic update for a softmax policy with linear action preferences

$$\boldsymbol{w} = \boldsymbol{w} + \alpha^{\boldsymbol{w}}\delta \underbrace{\nabla \hat{v}(S, \boldsymbol{w})}_{\boldsymbol{x}(s)} \quad \text{just like semi-gradient TD}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta \quad \underbrace{\nabla ln\pi(A|S, \boldsymbol{\theta})}_{\boldsymbol{x}_h(s,a) - \sum_b \pi(b|s,\boldsymbol{\theta}) x_h(s,b)}$$

10. Implement this algorithm

11. Design concrete function approximators for an average reward actor-critic algorithm

12. Analyze the performance of an average reward agent

13. Derive the actor-critic update for a gaussian policy

Probability density means that for a given range, the probability of x lying in that range will be the area under the probability density curve.

Gaussian Distribution

$$p.d.f = p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

μ = mean of distribution, σ = standard deviation = $\sqrt{variance}$. Gaussian Policy

$$p.d.f = \pi(a|s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} exp(-\frac{(a-\mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2})$$

$\mu(s, \boldsymbol{\theta})$ can be any parameterized function. σ controls the degree of exploration; usually initialized to be large s.t a wide range of actions are tried. As learning process, we expect the variance to shrink and the policy to concentrate around the best action in each state. The agent can reduce the amount of exploration through learning.

For θ,

$$\nabla ln\pi(a|s, \boldsymbol{\theta}_\mu) = \frac{1}{\sigma(s, \boldsymbol{\theta})^2}(a - \mu(s, \boldsymbol{\theta}))\boldsymbol{x}(s)$$

For σ,

$$\nabla ln\pi(a|s, \boldsymbol{\theta}_\sigma) = (\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{\sigma(s, \boldsymbol{\theta})^2} - 1)\boldsymbol{x}(s)$$

Advantages of Continuous Actions

- it might not be straightforward to choose a proper discrete set of actions
- continuous actions allow use to generalize over actions

14. Apply average reward actor-critic with a gaussian policy to a particular task with continuous actions

For discrete actions, we use softmax policy parameterization; for continuous actions, we use Gaussian policy.