13.0: Intro

Up until now, the methods learned are based on action-value or state-value methods. They focus on learning estimates of $q_*(s, a)$ or its approximation $q(s, a, w)$. The policy is derived from these.

Now we explicitly model the policy, and this parametrized policy will select actions without looking at action-values:

$$\pi(a \mid s, \theta) = \Pr(A_t = a \mid S_t = s, \theta_t = \theta),$$

$\theta \in \mathbb{R}^{d'}$ is the policy's parameter vector.

If learned value function is used then the value function's weight vector is denoted:

$w \in \mathbb{R}^d$ (as in prior chapters).

Here we'll consider methods for learning this policy based on the gradient of some scalar performance measure $J(\theta)$ wrt the policy parameters.

we want to maximize performance so we'll use gradient-ascent update rule:

$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$, where,

- θ_t is the vector of policy parameters at time step t.
- α = learning rate
- $\widehat{\nabla J(\theta_t)}$ is a stochastic estimate whose expectation is approximately the gradient of the performance measure J wrt policy parameters θ_t.

The performance measure J is usually the expected return.

If we learn a value-function approximation additionally to the policy approximation, we have action-critic method, where the actor refers to the learned policy and the critic to the learned value-function.

13.1: Policy Approximation and its Advantages

This section explains why learning a policy directly can be better than learning a value function.

Exploration: to ensure exploration we generally require that the policy never become deterministic:

$$\pi(a|s,\theta) \in (0,1) \quad \forall \quad s, a, \theta.$$

Parameterizing the Policy:

To use gradient ascent, our policy $\pi(a|s,\theta)$ must be differentiable wrt its parameters θ. The most common way to do this for discrete action spaces is with the softmax function:

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

$h(s,a,\theta)$: Action preferences. This is the core of the parameterization. For each state-action pair we calculate a "preference" score. Higher # = higher preference for that action in that state.

The action preferences can be computed with a NN or simply be linear $(h(s,a,\theta) = \theta^T x(s,a))$.

Advantages:

1) The policy can approach a deterministic policy.
2) Enables a stochastic policy, which may be the optimal policy.
3) The policy might be simpler to approximate than the value function.

13.2: The Policy Gradient Theorem

This section presents one of the most important theoretical results in RL: it gives us a way to calculate the gradient of our performance objective, $\nabla J(\theta)$, w/o needing to know the dynamics of the environment.

Policy gradient methods have stronger convergence properties than action-value methods: why?

- Action-value instability: action-value functions max over q-values, so a small change can switch the action abruptedly.
- policy gradient smoothness: with a parameterized policy like softmax, a small change will only cause a small-smooth change in action probabilities $\pi(a|s,\theta)$.

Performance Measure J: defined as the value of the start state from S_0 and following our policy π_θ until the episode terminates, given our policy π_θ:

$$\boxed{J(\theta) \doteq V_{\pi_\theta}(S_0)},$$

V_{π_θ} = true value function for π_θ, the policy determined by θ.

This means "How much reward do we expect to get, on average, from starting state S_0 and following π_θ until the episode terminates?"

Policy Gradient Theorem: Episodic case.

sum over all possible action a.

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a \mid s, \theta)$$

- $\nabla J(\theta)$: overall direction in parameter space that will increase our expected total rewards the most.
- $\sum_s$: sum over all states in the environment, weighted by $\mu(s)$, which is the on-policy distribution under π.
- $q_\pi(s,a)$: true action-value function for policy π.
- $\nabla_\theta \pi(a \mid s, \theta)$: gradient of our policy function. It's a vector; tells us how we should change θ to make action a more/less likely in state s.

Here, π is the policy corresponding to parameter vector θ.

Proof in the book.

13.3: REINFORCE: Monte Carlo Policy Gradient

This section introduces REINFORCE, which is the first complete, pratical policy gradient algorithm in the book. It's a MC algorithm, so it learns from completed episodes.

1) Start w/ the theorem:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a \mid s, \theta)$$

2) Turn the sum into expectation: the term $\sum_s \mu(s)[\dots]$ is the definition of an expected value over states visited under policy π, since $\mu(s)$ is the probability of being in state s:

$$\nabla J(\theta) \propto E_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a \mid S_t, \theta) \right]$$

3) Sample the action: we still have a sum over all possible actions a, which is impossible to compute. The trick is to realize this sum can be turned into an expectation by multiply and dividing by $\pi(a \mid S_t, \theta)$:

$$\nabla J(\theta) = E_\pi \left[\sum_a \pi(a \mid S_t, \theta) q_\pi(S_t, a) \frac{\nabla(a \mid S_t, \theta)}{\pi(a \mid S_t, \theta)} \right]$$

$$= E_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)} \right] \quad \text{// replaced } a \text{ with the sample } A_t \sim \pi$$

4) Monte-Carlo Step: expression contains $q_\pi(S_t, A_t)$ which we don't know. This is where MC comes in. By definition the return G_t is an unbiased sample of $q_\pi(S_t, A_t)$. So we replace the unknown true value w/ the sample return G_t, which we can calculate after an episode is finished:

$$= E_\pi\left[G_t \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}\right] \qquad \text{// bc } E[G_t \mid S_t, A_t] = q_\pi(S_t, A_t).$$

This is what we need.

Hence, the stochastic gradient ascent update is:

$$\boxed{\theta_{t+1} \doteq \theta_t + \alpha G_t \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}}$$

Each increment is proportional to the product of the return and the gradient of the probability of taking the action actually taken, divided by th probability of taking that action.

Algo: Note $\nabla \ln x = \frac{\nabla x}{x}$.

- Good convergence properties (update in the same direction as the performance gradient).
- High variance and 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)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

//REINFORCE is a MC algorithm.

13.4: REINFORCE with Baseline

This section introduces an algorithm that addresses the high variance (bc MC return can be very noisy). It does so by adding a baseline.

The policy gradient theorem can be generalized to include a baseline:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a [q_\pi(s,a) - \underbrace{b(s)}_{\text{baseline}}] \nabla_\theta \pi(a \mid s, \theta)$$

The baseline can be any function as long as it does not vary with a; the equation remains valid bc the subtracted quantity is 0:

$$\sum_a b(s) \nabla \pi(a \mid s, \theta)$$

$$= b(s) \sum_a \nabla \pi(a \mid s, \theta) \quad \text{// since } b(s) \text{ not dependent on } a$$

$$= b(s) \nabla \sum_a \pi(a \mid s, \theta) \quad \text{// can move gradient operator out of sum}$$

$$= b(s) \nabla(1) \quad \text{// Sum of all actions in a state must} = 1.$$

$$= 0.$$

Since we subtract a term that sums to zero, we don't change expectation of the gradient. No added bias.

The new update rule:

$$\theta_{t+1} \doteq \theta_t + \alpha \left[G_t - b(S_t) \right] \frac{\nabla \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)}$$

or

$$\theta_{t+1} \doteq \theta_t + \alpha \left[G_t - b(S_t) \right] \nabla \ln \pi(A_t | S_t, \theta).$$

What should we use for baseline?

- use $b(S_t) = \hat{v}(S_t, w)$, then the term $\left(G_t - \hat{v}(S_t, w)\right)$ now represents how much better or worse the actual return was compared to the average expected return from that state. This is often called the advantage.
- In MDPs the baseline should vary with the state.

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

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

- we now have two step size parameters α^θ and α^w.
- For the values (α^w) in the linear case it's easy.

13.5: Actor-Critic Methods

The key change is moving from MC updates to TD updates.

REINFORCE w/ baseline is not considered an actor-critic method bc its state-value function is only used as a baseline, not as a critic (aka not used for updating the values of a state with estimates of subsequent states, i.e., bootstrapping).

What is true actor-critic method?

These methods introduce bootstrapping. The critic updates its value estimates using TD errors. This has 2 major consequences:

1) Introducing bias: the update is no longer based on the true, full return G_t. Rather it's based on estimates $\hat{v}(S_{t+1}, w)$.

2) Reducing variance and enable online learning. The TD update $(R_{t+1} + \gamma \hat{v}(S_{t+1}, w))$ is much less noisy (lower variance) than the full MC return G_t. Since it allows for update after every single step without waiting for the episode to end, it allows for online learning.

13.5.1 : One-step Actor-Critic methods

Simplest form of actor-critic method. Analogous to TD(0), SARSA(0), and Q-learning.

How to:

1) Replace the full MC return G_t with one-step return $[R_{t+1} + \gamma \hat{v}(S_{t+1}, w)]$.

2) Use the TD error. The update for the actor & critic is now driven by the TD error, δ_t:

$$\delta_t \doteq [R_{t+1} + \gamma \hat{v}(S_{t+1}, w)] - \hat{v}(S_t, w)$$

Actor update rule:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \nabla \ln \pi(A_t \mid S_t, \theta)$$

or

$$\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \frac{\nabla \pi(a \mid S_t, \theta)}{\pi(a \mid S_t, \theta)}$$

Use a learned state-value function as the baseline. The natural state-value function learning method to pair with this is semi-gradient TD(0).

Algo:

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

13.5.2: Generalization:

This provides a sophisticated way to blend 1-step (low variance, high bias) and MC (high variance, low bias) updates:

Replace the 1-step return by $G_{t:t+n}$ or G_t^{λ} respectively.

For Backward view of the λ-return:

use separate ET for the actor and the critic

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

13.6: Policy Gradient for Continuing problems

This section explains how to adapt policy gradient methods to continuous problems.

In tasks w/ no episodes, the old way of defining performance (total discounted reward from a start state) does not make sense.

Instead of maximizing the total reward, we aim to maximize the average reward per time step, denoted as $\bar{r}(\pi)$.

we want the policy that gives us the most reward on average for every step we take:

$$J(\theta) \doteq \bar{r}(\pi) \doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} E\left[R_t \mid A_{0:t-1} \sim \pi\right]$$

$$= \lim_{t\to\infty} E\left[R_t \mid A_{0:t-1} \sim \pi\right]$$

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

- μ is the steady-state distribution under π, which is assumed to exist and to be independent of S_0 (ergodicity assumption): $\mu(s) \doteq \lim_{t\to\infty} Pr\left(S_t = s \mid A_{0:t} \sim \pi\right)$
  - Remember this is the special distribution under which, if you select actions according to π, you remain in the same distribution:

$$\sum_s \mu(s) \sum_a \pi(a \mid s, \theta)\, p(s' \mid s, a) = \mu(s') \quad \forall\ s' \in S$$

- $v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s]$ and $q_\pi(s,a) \doteq \mathbb{E}[G_t \mid S_t = s, A_t = a]$ are defined w/ the differential return, which measures how good the future rewards are, compared to long-term average:

$$G_t \doteq [R_{t+1} - \bar{r}(\pi)] + [R_{t+2} - \bar{r}(\pi)] + \dots$$

Algo:

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
  $\delta \leftarrow R - \bar{R} + \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}}\delta$
  $\mathbf{z}^{\mathbf{w}} \leftarrow \lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla\hat{v}(S,\mathbf{w})$
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + \nabla \ln \pi(A|S,\boldsymbol{\theta})$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
  $S \leftarrow S'$

proof of policy gradient theorem for continuing case is in the book.

13.7: Policy Parameterization for Continuous Actions

This section tackles a crucial topic: How to apply policy gradient methods to problems w/ continuous action spaces: for example, robotic arms, steering wheel, etc.

Solution: learn a probability distribution.

Instead of learning the probability of each specific action, we learn the parameters of a probability distribution over the entire range of actions. The most common choice is a Gaussian.

$$p(x) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

- The mean (μ): in our policy this represents the agent's "best guess" for the optimal option.
- STD (σ): in our policy this controls the amount of exploration. A large σ means the policy is uncertain and will try actions far from the mean. Small σ means the policy is confident.

Policy Parameterization:

Key insight is to make both mean and std functions of the current state, S.

To produce a policy parameterization, the policy can be defined the normal pdf over a real scalar function, with means and std given by parametric function approximators that depend on the state:

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

$$\mu: S \times \mathbb{R}^d \longrightarrow \mathbb{R}$$

$$\sigma: S \times \mathbb{R}^{d'} \longrightarrow \mathbb{R}^+$$

we use 2 separate function approximators to learn the policy's parameter vector θ:

one for the mean, which takes the state s and outputs the mean of the action distribution:

$$\mu(s,\theta) \doteq \theta_\mu^T x_\mu(s)$$

one for the std, which takes the state and outputs the standard deviation:

$$\sigma(s,\theta) \doteq \exp\left[\theta_\sigma^T x_\sigma(s)\right] \quad \text{// exp ensures this is positive.}$$

Putting It All Together:

1) Select action: While in state s, agent uses its two networks to compute $\mu(s)$ and $\sigma(s)$. It then samples a single real-valued action a from the resulting normal distribution, $N(\mu, \sigma^2)$.

2) Update policy: use the exact same Actor-Critic or REINFORCE algorithms as before. The only thing that changes in the mathematical formula for our policy $\pi(a|s, \theta)$ and its gradient.

13.8: Summary

- Previously, we learned action-values and the policy was simple (e.g. pick action with the highest value).
- Policy gradient methods are different, we directly learn a parameterized policy, $\pi(a \mid s, \theta)$, which is its own complex function. Actions are taken by consulting this policy directly.
- Learning process is gradient ascent. At each step, we update the policy's parameters, θ, in the direction that most improve our long-term performance.

Advantages:

- can learn specific probabilities for taking actions
- can learn appropriate levels of exploration and approach deterministic policies asymptomatically.
- can handle continuous action spaces
- policy gradient theorem has a theoretical advantage.

REINFORCE

- The REINFORCE method follows directly from PGT.
- Adding a state-value function as a baseline reduces variance w/o introducing bias.
- Using state-value function for bootstrapping introduces bias but is often desirable (TD over MC reasons, decreased variance).
- The state-value function [critic] assigns credit to the policy's action selections [actor].