1

Policies, Optimization, and Gradients DRAFT

We begin our study of reinforcement learning by jumping into the two core questions. First, given observed information about the world, which action should we take? We answer this question by reasoning about decision *policies*, which map observations to actions. Second, how do we design good policies? To answer this, we must define what it means to make good decisions; we take an empirical rather than philosophical approach. It is assumed that we can observe a *reward* signal and that a policy is better if it achieves higher reward. However, we observe *only* the reward of actions taken, rather than labels of the best action or counterfactual outcomes, requiring careful thought about how to *estimate* the reward a policy achieves. Eventually, we turn policy design into an *optimization* problem, which we primarily tackle using first-order methods, i.e. algorithms based on estimates of the *gradient* of our objective function.

1.1 Setting

The decision making entity is referred to as "the agent", and the setting in which it acts is called "the environment". The interaction between an agent and the environment is as follows.

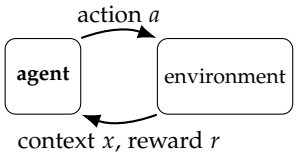

Figure 1.1: The interaction loop.

The agent observes information about the environment, referred to as the context for the decision. The context is denoted as $x \in \mathcal{X}$, where $\mathcal{X}$ is the set of all possible contexts. The agent takes an action $a \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions. After taking an action, the agent observes a reward $r \in \mathbb{R}$ generated by the environment. There is a causal order: first observation, then action, and finally reward. The action can depend on the observation (but not the reward) and the reward can depend on both the context and the observation.

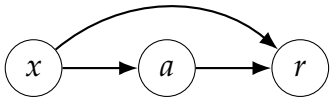

Figure 1.2: Graphical model illustrating dependence of context $x \sim \mathcal{D}_x$, action $a \sim \pi(x)$, and reward $r \sim R(x, a)$.

However, it is limiting to expect that the reward is always a deterministic function of the two. As a result, we take a probabilistic viewpoint and write $r \sim R(x, a)$ where $R : \mathcal{X} \times \mathcal{A} \to \Delta(\mathbb{R})$ is the reward function. We do not assume that we know the function $R(x, a)$

a priori.

Example 1.1 (Recommendation)**.** Consider movie recommendation on a streaming platform. The context x includes user profile information and viewing history. The action a is the new movie to promote on the front page, and $\mathcal{A}$ is the set of all recent movies available on the platform. The reward r is whether the user clicks $r = 1$ or not $r = 0$ on the recommended movie.

Alternative reward definitions are possible: whether the user eventually watches the movie, whether they watch at least 15 minutes of the movie, or the proportion of the movie that they watch. Each definition corresponds to a slightly different platform goal.

Our main focus in reinforcement learning is on the agent; the specification of the environment via the context and reward is outside our scope for now. As such, the main task is to design a (possibly randomized) policy $\pi : \mathcal{X} \to \Delta(\mathcal{A})$ which maps contexts to distributions over actions. That is, the agent samples an action $a \sim \pi(x)$ from the distribution defined by the context and the policy. We will also write $\pi(a|x)$ to mean the probability of taking action a in context x. In the special case that policy is deterministic, $\pi(a|x)$ is equal to one for exactly one action a (and zero for the rest), and we will use the notation $a = \pi(x)$.

We consider a family of parametric policies, similarly to how we consider parametric functions in supervised machine learning. We denote by π_θ the policy instatiated by the parameters $\theta \in \Theta$. Then the policy class is $\Pi = \{\pi_\theta \mid \theta \in \Theta\}$.

Example 1.2 (Recommendation cont.)**.** Continuing the movie recommendation example, we may choose to map user contexts x to movies a with a deep network. The parameters θ are the weights of the network and the policy class Π is defined by the chosen architecture. The user profile information and viewing history in x is represented as vectors or tokens. The network output consists of $|\mathcal{A}|$ nodes equal to the number of possible movies to promote, and given x, the output of each node is between zero and one, representing the probability $\pi(a|x)$.

The goal in reinforcement learning is to maximize the expected reward. Supposing that contexts are drawn from a distribution $\mathcal{D}_x$, and given a parametric policy π_θ,

$$J(\theta) = \mathbb{E}_{x\sim D_x}[\mathbb{E}_{a\sim\pi_\theta(x)}[\mathbb{E}_{r\sim r(x,a)}[r]]]$$

where the above expression uses the dependence structure imposed by the causal order illustrated in Figure 1.2. The objective J writes the expected reward as a function of the policy parameters, suppressing the implicit dependence on contexts, actions, and details of the policy. This objective is our guiding quantity; we want to make it large.

Comparison with supervised learning Recall that in the setting of supervised machine learning, features x are used to predict labels y with data coming from some distribution $x, y \sim \mathcal{D}$. Given a loss function ℓ, the key objective is find a parametric model f_θ with small expected loss

$$\mathcal{L}(\theta) = \mathbb{E}_{x,y\sim\mathcal{D}}[\ell(f_\theta(x), y)]\,.$$

How does this compare to reinforcement learning? We can think of the loss as analogous to the negative reward, the label y as analogous to the action a, and the model f_θ as analogous to the policy π_θ. In fact, these identifications are valid, and any reinforcement learning algorithm could be applied to a supervised machine learning setting where we define $r = -\ell(\pi(x), y)$.

However, there is a crucial distinction: in supervised learning, the correct label is given so the loss can be directly optimized. On the other hand, in reinforcement learning, "correct" actions are only implicitly defined as those achieving the best reward. Furthermore, the reward distribution $R(x, a)$ is generally unknown, and only samples $r \sim R(x, a)$ are observed. This *black-box* access to rewards is much less informative than knowing the full functional form. While it is possible to apply a reinforcement learning algorithm to supervised training, it amounts to throwing away what we know about loss functions and true labels and is generally much more inefficient. On the other hand, if we do have labeled "expert" actions, supervised learning algorithms may be a good way to design policies, which we discuss further in a later chapter.

1.2 *Estimation*

The objective $J(\theta)$ is an expectation over unknown distributions from which we only have samples. This is not an unfamiliar problem: in supervised learning, the true objective is the expected loss over the data distribution $\mathcal{L}(\theta)$, and evaluation is done by average loss over a test data set. As long as the test data set contains samples $(x_i, y_i) \sim \mathcal{D}$ for $i = 1, ..., n$, this is an unbiased estimate, often called a Monte Carlo estimate. In expectation over the data sampling distribution, the average test loss is equal to $\mathcal{L}(\theta)$. As long as n is reasonably large, the estimate should be reasonably accurate because the variance of the estimate decreases as $1/n$.

Monte Carlo estimates are named after the Monte Carlo Casino in Monaco. This is not the only time we will encounter gambling in this text. (Metropolis, *The Beginning of the Monte Carlo Method*, 1987.)

Let's review these ideas in detail in the reinforcement learning setting. Recall that $J(\theta)$ is defined as the expected reward under $x \sim D_x$, $a \sim \pi_\theta(x)$, and $r \sim R(x, a)$. Suppose we have a dataset of this interaction process $(x_1, a_1, r_1), \ldots, (x_n, a_n, r_n)$. The Monte Carlo estimate $\hat{J}_\theta$ of $J(\theta)$ is the average reward, and it is unbiased

$$\mathbb{E}[\hat{J}_\theta] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} r_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[r_i] = J(\theta)\,.$$

The variance of an unbiased estimate captures how much it differs from the truth on average.

$$Var(\hat{J}_\theta) = \mathbb{E}\left[(\hat{J}_\theta - \mathbb{E}[\hat{J}_\theta])^2\right] = \mathbb{E}\left[(\hat{J}_\theta - J(\theta))^2\right]\,.$$

Let σ_θ^2 denote the variance of r_i under π_θ. Notice three sources of randomness contribute to the variance: variation in which contexts we see, variation in which actions our policy takes, and variation in the reward itself. Then for $Var(\hat{J}_\theta)$ we have

$$\mathbb{E}\left[\left(\frac{1}{n}\sum_{i=1}^{n}(r_i - J(\theta))\right)^2\right] = \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}\left[(r_i - J(\theta))^2\right] = \frac{\sigma_\theta^2}{n}\,.$$

The average of n samples reduces the variance by a factor of n, verifying the intuition that more samples are better.

Exercise 1. *Show that the variance of the reward σ_θ^2 can be decomposed into the following three terms. Then, explain the meaning of each of the terms.*

$$\begin{aligned}\sigma_\theta^2 = \mathrm{Var}(r) = \mathbb{E}_{x\sim\mathcal{D}_x, a\sim\pi_\theta(x)}\left[\mathrm{Var}_{r\sim R(x,a)}(r)\right] \\ + \mathbb{E}_{x\sim\mathcal{D}_x}\left[\mathrm{Var}_{a\sim\pi_\theta(x)}\left(\mathbb{E}_{r\sim R(x,a)}[r]\right)\right] \\ + \mathrm{Var}_{x\sim\mathcal{D}_x}\left(\mathbb{E}_{a\sim\pi_\theta(x), r\sim R(x,a)}[r]\right)\end{aligned}$$

Exercise 2. *In each of the following scenarios, how does the variance of the reward compare to the general setting? Your answer show refer to the three terms in Exercise 1.*

1. *In the movie recommendation setting (Example 1.1), there is only a single user whose context x_0 never changes.*
2. *Still in the movie recommendation setting, but now back to multiple users, the policy π_θ is as follows: use a logistic regression model to predict how likely the user is to click on each movie, then recommend the one with the highest probability.*
3. *Consider training a language model to answer mathematical word problems, where contexts x are questions, actions a are integer numerical answers, and reward r is 1 if the answer is correct and 0 otherwise.*

1.2.1 Variance Reduction with Baselines

The variance of the reward σ_θ^2 depends on the variation in contexts and the randomness of the policy, in addition to inherent uncertainty in $R(x, a)$. One method for reducing variance is to account for the variation across contexts. This variation might have a large effect on rewards, but it is not affected by our actions, so we should be able to ignore it when designing the policy. Intuitively, given a context, it is more important to consider the increased reward due to our action, rather than the raw value of reward.

Example 1.3 (Recommendation cont.)**.** Continuing the movie example, some users barely glance at the recommendation banner while others are

more open to suggestions. Variation in the user population on platform contributes to the variance of the observed clicks. By estimating a per-user "suggestibility" baseline $b(x)$, we may be able to mitigate this source of variance.

Motivated by the observation, consider subtracting a context-dependent baseline and replace the reward with the *lift* $r_i - b(x_i)$. This corresponds to a new objective

$$J_b(\theta) = \mathbb{E}_{x\sim D_x}[\mathbb{E}_{a\sim\pi_\theta(x)}[\mathbb{E}_{r\sim r(x,a)}[r - b(x)]]] = J(\theta) - \mathbb{E}_{x\sim D_x}[b(x)]$$

where the second equality holds because $b(x)$ does not depend on a or r. The remaining term $\mathbb{E}_{x\sim D_x}[b(x)]$ does not depend on θ at all; the objective $J(\theta)$ is shifted by a constant value. While $J_b(\theta)$ is the expected lift rather than expected reward, we can design a reward maximizing policy equally well by optimizing $J_b(\theta)$ instead of $J(\theta)$.

So how should we pick the baseline? By the law of total variance, we can write the variance of the baseline-corrected reward

$$\begin{aligned}\mathrm{Var}(r - b(x)) &= \mathbb{E}_x\left[\mathrm{Var}_{r|x}(r - b(x))\right] + \mathrm{Var}_x\left(\mathbb{E}_{r|x}[r - b(x)]\right)\\ &= \mathbb{E}_x\left[\mathrm{Var}_{r|x}(r)\right] + \mathrm{Var}_x\left(\mathbb{E}_{r|x}[r] - b(x)\right)\end{aligned}$$

The second line follows since $b(x)$ is a constant when conditioned on x. If $b(x)$ is close to $\mathbb{E}_{a\sim\pi_\theta(x),r\sim R(x,a)}[r]$, then the second term becomes small, and the context-driven variance can be approximately eliminated. This means that a good baseline approximates the per-context expected reward. What is left in the first term depends only on the variance in reward due to the random policy and reward function.

Example 1.4 (Recommendation cont.)**.** Continuing the movie example, we may train a per-user baseline $b(x)$ through supervised learning

$$\min_b \sum_{x,a,r\in\mathcal{D}} \ell(b(x), r)$$

where $\mathcal{D}$ is a large aggregated historical dataset. The baseline captures how often users have been observed to click on movies in general. The baseline is always unbiased as long as it is independent of actions. However, it does not approximate the "gold standard" $\mathbb{E}_{r|x}[r]$. (Instead, it corresponds to estimating $\mathbb{E}_{a\sim\pi'(x),r\sim R(x,a)}[r]$ for old policies π'.) Nonetheless such baselines are often observed to reduce variance in practice.

1.2.2 *Off-Policy Estimation*

So far, our objective estimates require that data is collected using the policy π_θ. But what if data was generated by some other policy $\pi_{\theta'}$? We call such a setting *off-policy*. The average reward is equal to $J(\theta')$ in expectation instead of $J(\theta)$. It turns out that we can still estimate $J(\theta)$ using a simple reweighting trick that we will return to several times. We first state it generally.

Fact 1.1. *Suppose that ρ, ρ' are distributions, and $u \sim \rho'$. As long as $\rho'(u) > 0$ whenever $\rho(u) > 0$,*

$$\mathbb{E}_{u\sim\rho'}\left[\frac{\rho(u)}{\rho'(u)}u\right] = \int_u \frac{\rho(u)}{\rho'(u)}u\rho'(u)du = \mathbb{E}_{u\sim\rho}[u]$$

The ratio of likelihoods is often referred to as the *importance weight* and will be denoted by w. This inspires the estimate

$$\hat{J}'_\theta = \frac{1}{n}\sum_{i=1}^{n}\frac{\pi_\theta(a_i|x_i)}{\pi_{\theta'}(a_i|x_i)}r_i = \frac{1}{n}\sum_{i=1}^{n} w_i r_i.$$

By the reweighting rule,

$$\mathbb{E}_{a\sim\pi_{\theta'}(x)}\left[\mathbb{E}_{r\sim r(x,a)}\left[\frac{\pi_\theta(a|x)}{\pi_{\theta'}(a|x)}r\right]\right] = \mathbb{E}_{a\sim\pi_\theta(x)}[\mathbb{E}_{r\sim r(x,a)}[r]$$

and therefore $\mathbb{E}[\hat{J}'_\theta] = J(\theta)$. It seems magical to get unbiased estimates even when samples are from a different action distribution. The temper the magic, there are two important observations. One is that for the importance weight to be well defined, we need that $\pi_{\theta'}(a|x) > 0$ whenever $\pi_\theta(a|x) > 0$. Our mathematical justification fails if we try to evaluate a policy π_θ that considers actions that were impossible under the data collection policy $\pi_{\theta'}$. Second, though the estimate is still unbiased, it is not the same quality as the "on-policy" estimate $\hat{J}_\theta$. Consider the variance of the reweighted rewards under the off-policy distribution $\pi_{\theta'}$,

While we are using $\pi_{\theta'}$ to denote the alternative policy, nothing in the argument requires that it actually shares a parametric form with π_θ. In fact, as long as the data collection policy satisfies the coverage condition, it can take any form, including hand-designed and rule-based policies.

$$\begin{aligned}
Var_{\theta'}(w_i r_i) &= \mathbb{E}_{\theta'}[w_i^2 r_i^2] - J(\theta)^2 \\
&= \mathbb{E}_\theta[w_i r_i^2] - J(\theta)^2 \\
&= \mathbb{E}_\theta[w_i r_i^2] - \mathbb{E}_\theta[r_i^2] + \mathbb{E}_\theta[r_i^2] - J(\theta)^2 \\
&= \mathbb{E}_\theta[(w_i - 1)r_i^2] + Var_\theta(r_i)
\end{aligned}$$

The first equality follows from the standard identity that variance is equal to the expectation of the square minus the square of the expectation. The second line follows by the reweighting trick. The third line adds and subtracts the same quantity. The final line groups terms and applies the standard variance identity again.

Whether the first term is positive or negative depends on the correlation of the importance weight with the reward. It is negative, and thus the the variance is reduced, if the weight is negatively correlated with the magnitude of the reward (squared). This occurs if π_θ down-weights actions (compared to $\pi_{\theta'}$) with large observed reward. On the other hand, the first term is positive if the importance weights are positive correlated with reward. Because we are concerned maximizing reward, the second scenario is commonly encountered. In the reinforcement learning setting, we usually expect importance weighting to increase the variance of our estimates.

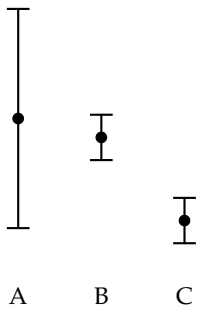

Figure 1.3: Estimated value (points) and uncertainty (intervals), which is proportional to $\sqrt{\text{variance}}$. B is clearly preferred to C, but whether it is preferred to A is ambiguous.

Why does variance matter? Suppose we are choosing between two policies, and policy A appears to be better than policy B, though our estimate also has much higher variance. Which policy should we choose? Policy A is promising but risky, while Policy B appears safer. If we can only make the choice once, we may prefer to avoid risk and make the more conservative choice for Policy B. However, if our setting is *adaptive* and policies can be changed down the line, we may prefer the riskier bet. The former setting is called "offline reinforcement learning," in which policies must be designed with offline data alone, without new interactions. In some sense, this is similar to supervised learning, except that there are no closed-form loss functions. In this text, we focus on "online" settings where interaction is possible. The adaptive setting will be a key topic later on in this text. In this chapter, our focus is instead on designing policies incrementally, and we consider variance insofar as it affects local improvements.

1.3 Policy Gradient

So far, we studied how to estimate the expected reward of a policy given data. But how should we propose new policies? We now study ways of updating and improving policies.

1.3.1 Improvement with Gradients

Everyone knows that we train supervised models with gradients. Why are gradients useful? Because they provide a (local) improvement direction! In other words, taking a (small) step in the direction of the gradient increases the function's value. To see this, we appeal to Taylor's theorem:

$$J(\theta + u) = J(\theta) + \nabla J(\theta)^\top u + u^\top \nabla^2 J(\theta_0) u + \dots$$

As long as we keep the update small, say $\|u\|^2 \leq \varepsilon$, the higher order terms will scale as $\varepsilon^2, \varepsilon^3, \dots$ and become negligible. Then we formalize the problem of choosing improvement step u as a constrained maximization problem:

$$\max_{\|u\| \leq \epsilon} \; J(\theta) + \nabla J(\theta)^\top u$$

The solution is $u_\star = \alpha \nabla J(\theta)$ for $\alpha = \frac{\varepsilon}{\|\nabla J(\theta)\|}$. As long as our approximation is valid (guaranteed when α is small enough to make the higher order terms negligible), this u is an improvement step. The resulting incremental update recovers our familiar gradient ascent algorithm:

To solve the optimization problem, note that $v^\top u = \|v\| \|u\| \cos(\angle(u, v))$.

We "ascend" because we are maximizing reward rather than "descending" to minimize loss.

$$\theta \leftarrow \theta + \alpha \nabla J(\theta)$$

Let's revisit our ongoing comparison with supervised learning. In supervised learning, the loss function is known, so it is possible to compute the gradient $\nabla_\theta \ell(f_\theta(x_i), y_i)$ for each data point directly. Using training data, we approximate the overall gradient of $\nabla \mathcal{L}(\theta)$ by the average gradient over the training set (or a minibatch).

Example 1.5. Suppose $f_\theta(x)$ is a probabilistic model, assigning probabilities $f_\theta(y|x)$ to each possible label y, and we use the cross entropy loss $\ell(f_\theta(x), y) = -\log f_\theta(y|x)$. Using the estimated gradient yields the descent (since in this case we are minimizing the loss) step

$$\theta \leftarrow \theta + \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla \log f_\theta(y_i|x_i) \,.$$

This update encourages the model to make the observed labels y_i more likely given corresponding x_i.

1.3.2 *Gradient Estimation*

The fact of black box rewards leads to a key difference compared with supervised learning. We must find a way to estimate gradients without knowing the form of the reward function. Let let ρ_θ define the joint distribution over (x, a) defined by $\mathcal{D}_x$ and $\pi_\theta(x)$. We begin the derivation with the reweighting trick

$$\begin{aligned}
\nabla J(\theta) &= \nabla \mathbb{E}_{x,a\sim\rho_\theta}[\mathbb{E}_{r\sim R(x,a)}[r]] \\
&= \nabla \mathbb{E}_{x,a\sim\rho_{\theta'}}\left[\frac{\rho_\theta(x,a)}{\rho_{\theta'}(x,a)} \mathbb{E}_{r\sim R(x,a)}[r]\right] \\
&= \mathbb{E}_{x,a\sim\rho_{\theta'}}\left[\frac{\nabla \rho_\theta(x,a)}{\rho_{\theta'}(x,a)} \mathbb{E}_{r\sim R(x,a)}[r]\right] \\
&= \mathbb{E}_{x,a\sim\rho_{\theta'}}\left[\frac{\rho_\theta(x,a)}{\rho_{\theta'}(x,a)} \nabla \log \rho_\theta(x,a) \mathbb{E}_{r\sim R(x,a)}[r]\right]
\end{aligned}$$

The first line is the definition of J. The second line is the reweighting trick which is valid for any distribution ρ such that $\rho(x, a) > 0$ whenever $\rho_\theta(x, a) > 0$. The third line uses the linearity of expectation to move the gradient into the expectation, since the expectation no longer depends on θ. The final line uses the log-derivative trick $\nabla f(\theta) = f(\theta) \nabla \log f(\theta)$, which will allow us to remove the dependence on the unknown context distribution $\mathcal{D}_x$. Since $\rho_\theta(x, a) = D_x(x)\pi_\theta(a|x)$,

$$\frac{\rho_\theta(x,a)}{\rho_{\theta'}(x,a)} = \frac{\pi_\theta(a|x)}{\pi_{\theta'}(a|x)}, \quad \nabla \log \rho_\theta(x,a) = \nabla \log \pi_\theta(a|x).$$

The gradient is therefore

$$\nabla J(\theta) = \mathbb{E}_{x,a\sim\rho_{\theta'}}\left[\frac{\pi_\theta(a|x)}{\pi_{\theta'}(a|x)} \nabla \log \pi_\theta(a|x) \, \mathbb{E}_{r\sim R(x,a)}[r]\right].$$

Written in this form as an expectation, it is straightforward to construct a Monte Carlo estimate of the gradient using samples. In the on-policy case where x_i, a_i, r_i are collected with policy π_θ this is known as the REINFORCE policy gradient,

$$\theta \leftarrow \theta + \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla \log \pi_\theta(a_i|x_i)\, r_i\,.$$

The name is a acronym for "*RE*ward *I*ncrement = *N*onnegative *F*actor times *O*ffset *R*einforcement times *C*haracteristic *E*ligibility," (Williams, *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*, 1992.)

It is illuminating to compare this update to that of a supervised model under cross entropy loss from Example 1.5. The updates have a similar form, except that for REINFORCE the data points are weighted by the reward. This makes the policy more likely to take actions which achieved high reward.

In the general case that data collected under some other $\pi_{\theta'}$ (as long as $\pi_{\theta'} > 0$ whenever $\pi_\theta > 0$), the off-policy gradient estimate is

$$\widehat{\nabla J}'_\theta = \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_\theta(a_i|x_i)}{\pi_{\theta'}(a_i|x_i)} \nabla \log \pi_\theta(a_i|x_i)\, r_i.$$

We can interpret this update as follows: increase the likelihood of actions which achieved high reward (large r_i) and were comparatively *unlikely* under the data collection policy $\pi_{\theta'}$ (large $w_i = \frac{\pi_\theta(a_i|x_i)}{\pi_{\theta'}(a_i|x_i)}$). Both aspects contribute to high variance, especially under multiple gradient updates. The first means that importance weights are usually positively correlated with observed rewards, and the second encourages π_θ into actions that are poorly covered by $\pi_{\theta'}$. As a result, it does not usually work well to take multiple off-policy gradient update steps on the same batch of data. In the next section, we will see how to fix this idea by more carefully constraining the distance between policies.

Before moving on, it is worth remembering the baseline trick from Section 1.2.1. In all of the estimates, we may replace r_i with $r_i - b(x_i)$ to reduce variance without introducing bias. This is true because the baseline-adjusted objective $J_b(\theta)$ has the same gradient as the objective $J(\theta)$.

We do not actually need the next section to propose the most common heuristic, which is the following. Simply discard datapoints from the gradient estimate as soon as (i) $w_i > 1+\epsilon$ for positive instances $r_i - b(x_i) > 0$ or (ii) $w_i < 1-\epsilon$ for negative instances $r_i + b(x_i) < 0$. This keeps the variance down and prevents the policy from continuing to update in low coverage directions. This algorithm is known as the "clip" variant of proximal policy optimization (PPO), which we discuss in more detail in the following section. Combining this heuristic with the variance reduction trick in Exercise 3 gives rise to the "group relative policy optimization" algorithm which is widely used "post-training" large language models.

Exercise 3. *Consider training a language model to answer mathematical word problems, where contexts x are questions, actions a are integer numerical answers, and reward r is 1 if the answer is correct and 0 otherwise. In this setting, it is possible to sample multiple outcomes for each context, an observation which we can use to reduce variance. For each context x_i, sample a group of G actions $a_{i,1}, \ldots, a_{i,G} \sim \pi_\theta(\cdot|x_i)$. and define the baseline $b(x_i) = \frac{1}{G}\sum_{j=1}^{G} r_{i,j}$.*

1. *What is $b(x_i)$ an unbiased estimate of? Hint: Recall the variance decomposition in Section 1.2.1.*

2. *How does the variance of $b(x_i)$ depend on G?*

3. *In practice, it is common to further standardize the rewards using* $\sigma_i^2 = \frac{1}{G}\sum_{j=1}^{G}(r_{i,j} - b(x_i))^2$.

$$\widehat{\nabla J}_{GRPO} = \frac{1}{nG}\sum_{i=1}^{n}\sum_{j=1}^{G}\nabla \log \pi_\theta(a_{i,j}|x_i)\frac{r_{i,j} - b(x_i)}{\sigma_i + \varepsilon}$$

Is this still an unbiased estimate of the gradient $\nabla J(\theta)$*?*

1.4 Trust Regions

Taking multiple policy gradient steps from one batch of data quickly runs into issues, due to high variance in off-policy gradient estimates. In this section, we attempt to remedy this issue by more carefully considering the distance between policies. In our motivation for the gradient descent algorithm, we used the Euclidean distance to define "small" updates. This is an arbitrary choice which imposes a geometry on parameter space. However, two parameter vectors θ, θ' that are close in ℓ_2 may induce very different action distributions, and vice versa.

The *KL divergence* of two distributions ρ and ρ' over a variable u is

$$KL(\rho' \parallel \rho) = \mathbb{E}_{u\sim\rho'}\left[\log \frac{\rho'(u)}{\rho(u)}\right].$$

Named for the statisticians Kullback & Leibler who defined it in their 1951 paper, which connected (then) new ideas from information theory to classical perspectives on estimation.

Intuitively, the KL divergence quantifies the "surprise" at seeing data from ρ' when it was expected to be from ρ. The KL divergence is not a true distance because it is asymmetric and does not satisfy the triangle inequality. However, it is always non-negative and equals zero only when the two distributions coincide.

Then, we measure how different policies are by the KL divergence averaged over contexts:

$$d_{KL}(\theta', \theta) = \mathbb{E}_{x\sim D_x}[KL(\pi_{\theta'}(\cdot|x) \parallel \pi_\theta(\cdot|x))] = \mathbb{E}_{x,a\sim\rho_{\theta'}}\left[\log \frac{\pi_{\theta'}(a|x)}{\pi_\theta(a|x)}\right].$$

This is zero when $\theta' = \theta$ and grows as the policies diverge, regardless of how the policies happen to be parameterized.

Exercise 4. *Consider a softmax policy over two actions* $\{0, 1\}$ *with* $\theta \in \mathbb{R}$*:* $\pi(0) = \frac{1}{1+\exp\theta}$ *and* $\pi(1) = \frac{\exp\theta}{1+\exp\theta}$.

1. *What does the policy do when* $\theta = 0$, $\theta \to -\infty$, *and* $\theta \to +\infty$*?*
2. *Consider a fixed step* $u > 0$*. For case A, the initial* $\theta = 0$ *while for case B, the initial* $\theta = \log 99$*. Clearly, the Euclidean size of the step* u *is the same whether we are in case A or care B. Compare the resulting policies for* $u = \log 1.1$ *in case A versus case B.*
3. *Compare the KL-divergence in case A versus case B.*

1.4.1 Natural Policy Gradient

We now derive an alternative update step using d_{KL}. Like before, we will consider a first order approximation to $J(\theta)$. Instead of a Euclidean norm constraint, we will use a second order (quadratic) approximation to d_{KL}. Consider the Taylor expansion of $d_{KL}(\theta, \theta + u)$ around $u = 0$:

$$d_{KL}(\theta,\theta) + \nabla_u d_{KL}(\theta,\theta+u)|_{u=0}^{\top} u + u^{\top} \nabla_u^2 d_{KL}(\theta,\theta+u) u + ...$$

The first term vanishes. Then noting that $u = 0$ is a minimum (since d_{KL} is always non-negative), it must be a critical point so $\nabla_u d_{KL}(\theta, \theta + u)|_{u=0} = 0$. We are left with the quadratic term, defined by the hessian

$$F(\theta) = \mathbb{E}_{x,a\sim\rho_\theta}\left[\nabla \log \pi_\theta(a|x) \nabla \log \pi_\theta(a|x)^{\top}\right],$$

which is called the *Fisher information matrix* $F(\theta)$.

Named for Ronald Fisher who defined it in *On the Mathematical Foundations of Theoretical Statistics*, 1921.

We have that for small u, $d_{KL}(\theta, \theta + u) \approx u^{\top} F(\theta)\, u$. We now derive an approximate KL-constrained improvement step:

$$\max_u \ \nabla J(\theta)^{\top} u \quad \text{s.t.} \quad u^{\top} F(\theta)\, u \le \epsilon^2.$$

The solution is $u_\star = \alpha F(\theta)^{-1} \nabla J(\theta)$ for $\alpha = \epsilon (\nabla J(\theta)^{\top} F(\theta)^{-1} \nabla J(\theta))^{-1/2}$. Thus we derive the *natural policy gradient* update as

By a change of variables $v = F_0^{1/2} \Delta$, $c = F_0^{-1/2} g_0$, this reduces to $\max_{\|v\|^2 \le \delta^2} c^{\top} v$, which has the same form as the previously derived gradient update step.

$$\theta \leftarrow \theta + \alpha F(\theta)^{-1} \nabla J(\theta)\,.$$

The Fisher matrix preconditions the gradient so that equal steps in parameter space correspond to equal steps in policy space. As a result, the update is less sensitive to the parameterization of π_θ.

In practice, the update must be computed from data. Given samples $x_i, a_i, r_i \sim \rho_\theta$, the gradient $\nabla J(\theta)$ is estimated with REINFORCE and the Fischer information matrix is estimated by

This algorithm was proposed by Kakade in *A Natural Policy Gradient*, 2002. A related idea is to solve the constrained optimization problem without the second order approximation (and hence without a closed-form update), explored by Schulman et al. in *Trust Region Policy Optimization*, 2015.

$$\hat{F}_\theta = \frac{1}{n} \sum_{i=1}^{n} \nabla \log \pi_\theta(a_i|x_i) \nabla \log \pi_\theta(a_i|x_i)^{\top}\,.$$

Exercise 5. *We will verify the first-order computation for the Taylor expansion of d_{KL}. First, show that*

$$\nabla_u d_{KL}(\theta,\theta+u) = -\mathbb{E}_{x,a\sim\rho_\theta}[\nabla_u \log \pi_{\theta+u}(a|x)]\,.$$

Evaluating at $u = 0$, this simplifies to $-\mathbb{E}_{x,a\sim\rho_\theta}[\nabla_\theta \log \pi_\theta(a|x)]$. Prove the following standard identity about the expected score:

$$\mathbb{E}_{x,a\sim\rho_\theta}[\nabla_\theta \log \pi_\theta(a|x)] = 0$$

Hint: you may want to use the log-derivative trick in reverse along with the fact that $\int_a \pi(a|x) da = 1$.

Exercise 6. *Recall the softmax policy setting from Exercise 4. Derive and then compare the expressions for the policy gradient update and the natural policy gradient update.*

1.4.2 Proximal Policy Optimization

While the natural policy gradient gives a principled update direction, it requires inverting estimates of the Fisher matrix $F(\theta)$, which is expensive for large deep network policies that have millions or even billions of parameters. An alternative approach replaces the KL constraint with a soft penalty. Suppose we have collected data with θ'. We frame the update step with the following optimization problem:

$$\max_{\theta} \; J(\theta) - \lambda \, d_{KL}(\theta', \theta) = \max_{\theta} J_\lambda(\theta; \theta')$$

Rather than consider first or second order approximations, we will take several gradient steps on this objective. Since the data was collected by the policy $\pi_{\theta'}$, we use the off-policy REINFORCE expression for $\nabla J(\theta)$. The gradient of the KL penalty is (see Exercise 5)

$$\nabla_\theta d_{KL}(\theta', \theta) = \mathbb{E}_{x,a \sim \rho_{\theta'}}[\nabla \log \pi_\theta(a|x)].$$

Putting the two terms together and regrouping, the expression for the gradient of the penalized objective is

$$\nabla J_\lambda(\theta; \theta') = \mathbb{E}_{x,a \sim \rho_{\theta'}} \left[\nabla \log \pi_\theta(a|x) \; \left(\frac{\pi_\theta(a|x)}{\pi_{\theta'}(a|x)} \mathbb{E}_{r \sim R(x,a)}[r] + \lambda \right) \right]$$

The corresponding estimate from data (collected with policy $\pi_{\theta'}$) usually incorporates an additional reward baseline $b(x)$ to reduce variance. This is the update used in the *proximal policy optimization* algorithm, commonly referred to as PPO.

$$\widehat{\nabla J}'_{\lambda,\theta} = \frac{1}{n} \sum_{i=1}^{n} \nabla \log \pi_\theta(a_i|x_i) \; \left(\frac{\pi_\theta(a_i|x_i)}{\pi_{\theta'}(a_i|x_i)} (r_i - b(x_i)) + \lambda \right)$$

Because this is an off-policy gradient estimate, we can take multiple steps on one batch of data. Notice how this expression differs from the off-policy REINFORCE estimate, which includes only the importance-weighted reward term. The penalty multiplier λ gives a uniform bonus to all (action, context) pairs in the dataset. This mitigates the variance blowup that can happen when θ is not well covered by θ'. Effectively, the bonus λ encourages the updated policy to imitate the data collection policy θ', and in doing so, maintains a small KL divergence.

The most widely used variant of PPO, called PPO-clip, relies on a different approach. Instead, switching logic is used to prevent importance weights from leading to variance explosions. See the margin note at the end of Section 1.3.2 for further discussion (Schulman et al. *Proximal Policy Optimization Algorithms*, 2017).

1.5 Deterministic Policies and Zeroth-Order Optimization

So far in this chapter we have considered stochastic policies. In many settings, such as automatic feedback control, deterministic policies are widely used. These control policies are deterministic functions with only a handful of tunable parameters. For example, a *PID controller* maps an observed error signal to a continuous control action according to three scalar gains. Such controllers are used in settings ranging from heating and cooling in buildings to motor speed control in robots. PID controllers, along with other simple decision rules with a few tunable parameters, have been bread and butter in engineering practice for over a century.

In fact, randomness is not usually a desiderata for decision policies, and the expected reward is generally maximized by a deterministic policy. In reinforcement learning practice, after training a stochastic policy, it is common to deploy a deterministic version by taking the mode or mean action rather than sampling. Why not just start with a deterministic policy to begin with? The randomness was essential for the algorithms we discussed so far. For a deterministic policy, $\log \pi_\theta(a|x)$ is either $-\infty$ or 0, and therefore gradients $\nabla_\theta \log \pi_\theta(a|x)$ provide very little information.

The following policy achieves maximal reward: $\pi(x) = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{r \sim R(s,a)}[r]$. We discuss the optimal policy in more detail in the following chapters.

To handle deterministic policies, we need a way to estimate how $J(\theta)$ changes with θ that does not require randomness in actions. The idea behind *zeroth-order* (also called derivative-free) optimization is to introduce randomness in parameter space instead. We may perturb the policy parameter and observe how the resulting (deterministic) policy performs. Define the Gaussian-smoothed objective

$$J_\delta(\theta) = \mathbb{E}_{v \sim \mathcal{N}(0,I)}\left[J(\theta + \delta v)\right],$$

which averages the objective over a distribution of parameters around θ, with $\delta > 0$ controlling the smoothing radius. For small δ this is a good proxy for the actual objective. It also has some benefits: $J_\delta(\theta)$ will be differentiable even when the underlying $J(\theta)$ is not, and furthermore, it is easy to estimate the gradient. To do so, we first rewrite J_δ as an integral and use a change of variables $w = \theta + \delta v$.

$$J_\delta(\theta) = \int J(\theta + \delta v)\, p(v)\, dv = \int J(w)\, p\left(\frac{w - \theta}{\delta}\right) \delta^{-d}\, dw$$

The change of variables moves the dependence on θ into the density

function. Taking the gradient,

$$\begin{aligned}\nabla_\theta J_\delta(\theta) &= \int J(w)\, \nabla_\theta \left[p\left(\frac{w-\theta}{\delta}\right)\right] \delta^{-d}\, dw \\ &= \int J(w)\, p\left(\frac{w-\theta}{\delta}\right) \nabla_\theta \log p\left(\frac{w-\theta}{\delta}\right) \delta^{-d}\, dw\end{aligned}$$

In the second line above, we use the log-derivative trick again. Recalling that p is a Gaussian density function, we compute

$$\nabla_\theta \log p\left(\frac{w-\theta}{\delta}\right) = \nabla_\theta \left(-\frac{1}{2}\left\|\frac{w-\theta}{\delta}\right\|^2 + \text{const}\right) = \frac{w-\theta}{\delta^2}\,.$$

Returning to the gradient expression and undoing the change of variables,

$$\begin{aligned}\nabla_\theta J_\delta(\theta) &= \int J(w)\, p\left(\frac{w-\theta}{\delta}\right) \frac{w-\theta}{\delta^2}\, \delta^{-d}\, dw \\ &= \int J(\theta+\delta v)\, p(v)\, \frac{v}{\delta}\, dv = \mathbb{E}_{v\sim\mathcal{N}(0,I)}\left[\frac{v}{\delta}\, J(\theta+\delta v)\right]\end{aligned}$$

This expression makes it clear how to apply the Monte Carlo estimate idea. Given n independently sampled perturbations $v_1, \ldots, v_n \sim \mathcal{N}(0, I)$, we take actions from each deterministic policy $\pi_{\theta+\delta v_i}$ for $j = 1, ..., m$ steps, observing reward r_{ij}. Then the estimate is

$$\widehat{\nabla J_\delta}(\theta) = \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{v_i}{\delta}\, r_{ij}\,.$$

This gives us the basic *random search* algorithm: repeatedly perturb the policy parameters in random directions, weight each direction by the reward it achieved, and average.

A common refinement is to use the two-point estimate, which requires symmetric sampling. For each v_i, use both $\theta + \delta v_i$ and $\theta - \delta v_i$, obtaining rewards r_{ij}^+ and r_{ij}^-, and form

$$\widehat{\nabla J_\delta}(\theta) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{i=1}^{m} \frac{v_i}{2\delta} \left(r_{ij}^+ - r_{ij}^-\right)\,.$$

This estimate reduces the variance significantly. One reason is that the reward difference removes the variation in the magnitude of $J(\theta)$ itself, similar to the baseline idea. The two point estimate is also easy to understand as an approximation of the classical definition of a directional derivative, $\nabla J(\theta)^\top v = \lim_{\delta\to 0} \frac{J(\theta+\delta v) J(\theta-\delta v)}{2\delta}$.

Random search underlies several practical algorithms which have found success in reinforcement learning. It is especially well motivated for structured policies with relatively low numbers of parameters compared to actions, especially when the number of actions is large or infinite (such as in the continuous case).

For examples, see Salimans et al., *Evolution Strategies as a Scalable Alternative to Reinforcement Learning*, and Mania et al., *Simple Random Search Provides a Competitive Approach to Reinforcement Learning*.