Policy Gradient Methods

TODO: currently in this section s is used to denote states

Q-learning and SARSA use information from *every* transition (s, a, r, s') in every trajectory, while black-box policy optimization methods only look at the total reward of the trajectories. Why should we not use the state and the action data already encoded in the trajectories to update the policy directly? Doing so leads to *policy gradient methods*.

As we have seen in the previous lecture, if the environment model and the reward function are *known*, we can compute the policy gradient conveniently using the back-propagation algorithm. However, in reinforcement learning, we often care about the case when we don't have access to the environment model and/or the reward function. *Policy gradient methods* seek to estimate the policy gradients from trajectories without access to the environment model and the reward function.

Before we dive in to the details, we should consider whether a gradient exists for a certain policy class. This can be interpreted as a continuity condition of the mapping from the parameters in the policy class to the trajectories. This is nontrivial to show for discrete action spaces and deterministic policies, since an infinitesimally small change in parameters can drastically change the policy and hence the trajectories. Therefore, in this lecture, we consider a class of stochastic policies parameterized by θ, $\pi_\theta : s \mapsto \pi_\theta(a|s)$. Under mild assumptions about the environment, we can safely assume that the policy gradient always exists for this policy class since stochastic policies "smooth out" the problem.

Let ξ denote a *trajectory* of states and actions, $\xi = (s_0, a_0, \ldots, s_T, a_T)$. We define the *total reward of the trajectory* ξ as,

$$R(\xi) = \sum_{t=0}^{T-1} r(s_t, a_t).$$

Our goal is to find the parameters that produce the policy that maximizes the expected total reward of the trajectories,

$$J(\theta) = E_{p(\xi|\theta)}[R(\xi)] = E_{p(\xi|\theta)}\left[\sum_{t=0}^{T-1} r(s_t, a_t)\right],$$

where $p(\xi|\theta)$ is the probability of the trajectory ξ given the policy parameterized by θ, which, we will see later, is also dependent on the transition model of the environment.

To find the optimal policy, we compute the policy gradient by

taking the derivative with respect to θ.

$$\begin{aligned}\nabla_\theta J &= \nabla_\theta E_{p(\xi|\theta)}\left[R(\xi)\right] \\ &= \nabla_\theta \sum_{\xi\in\Xi} p(\xi|\theta)\, R(\xi),\end{aligned}$$

where Ξ denotes the set of all possible trajectories. In the case when the state and/or action space is continuous, the sum should be replaced by an integral. The derivation will remain the same for integrals, although some steps would require additional justification[11].

[11] For example, dominated convergence theorem need to be invoked in order to swap the integral with the gradient operator in the next step.

Since $R(\xi)$ is the total reward of a *given* trajectory ξ, it has no dependence on θ. Therefore,

$$\nabla_\theta J = \sum_{\xi\in\Xi} \left(\nabla_\theta p(\xi|\theta)\right) R(\xi). \tag{o.o.88}$$

However, we cannot compute the gradient with eq. (o.o.88) because it requires us to evaluate the gradient for *all* possible trajectories. Instead, we want to obtain at least an estimate of the policy gradient using *samples* of trajectories. Therefore, we want to express the gradient as an *expectation* over probability $p(\xi|\theta)$ – the moment we do that, we can use the law of large numbers to draw samples from the distribution and estimate the expectation. Therefore, we use a simple trick,

$$\begin{aligned}\nabla_\theta J &= \sum_{\xi\in\Xi} \frac{p(\xi|\theta)}{p(\xi|\theta)} \left(\nabla_\theta p(\xi|\theta)\right) R(\xi) \\ &= E_{p(\xi|\theta)}\left[\frac{\nabla_\theta p(\xi|\theta)}{p(\xi|\theta)} R(\xi)\right].\end{aligned}$$

By the chain rule, we have, $\nabla_\theta \log\left(p(\xi|\theta)\right) = \frac{\nabla_\theta p(\xi|\theta)}{p(\xi|\theta)}$. So, we have an elegant expression of the policy gradient as an expectation,

$$\nabla_\theta J = E_{p(\xi|\theta)}\left[\nabla_\theta \log\left(p(\xi|\theta)\right) R(\xi)\right]. \tag{o.o.89}$$

This is sometimes called the *likelihood ratio policy gradient*. The likelihood ratio policy gradient can be interpreted as increasing the (log) probability of the trajectories with high reward and decreasing the (log) probability of the trajectories with low reward. To see this, consider a single trajectory ξ. Imagine that $R(\xi)$ is a large positive number, then if we do gradient ascent with respect to the total reward J, we are in some sense doing *gradient ascent* with respect to $\log\left(p(\xi|\theta)\right)$ according to eq. (o.o.89). Conversely, if $R(\xi)$ is a large negative number, we are performing *gradient descent* with respect to its log probability in some sense.

Note, however, that we still can not compute the policy gradient using the above equation because it requires us to evaluate

$\nabla_\theta \log p(\xi|\theta)$ in the expectation, yet we do *not* know the transition model $p(s_{t+1}|a_t, s_t)$.

However, we will see that it is not a problem for policy gradient methods. If we assume the Markov property, we have,

$$p(\xi|\theta) = p(s_0)\left(\prod_{t=0}^{T-2} p(s_{t+1}|a_t, s_t)\right)\left(\prod_{t=0}^{T-1} \pi_\theta(a_t|s_t)\right).$$

Then, we have,

$$\nabla_\theta \log p(\xi|\theta) = \nabla_\theta \log p(s_0) + \left(\sum_{t=0}^{T-2} \nabla_\theta \log p(s_{t+1}|a_t, s_t)\right) + \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right).$$

However, $\log p(s_0)$ and $\log p\left(s_{t+1}|s_t, a_t\right)$ do not depend on θ, so the gradients with respect to these terms are zero. Hence,

$$\nabla_\theta J = E_{p(\xi|\theta)}\left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t|s_t\right)\right) R(\xi)\right].$$

Notice that we don't know and can't control the system dynamics, but by formulating the problem this way, we don't need to – we have control over the policy class we choose, and thus can easily compute gradients. For example, we can use the *back-propagation* algorithm that we saw last week to compute the gradient $\nabla_\theta \log \pi_\theta\left(a_t|s_t\right)$.

As mentioned earlier, we can now use the law of large numbers to estimate this expectation,

$$\widetilde{\nabla}_\theta J = \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta\left(a_t^{(i)}|s_t\right)\right) R(\xi^{(i)})\right]. \tag{0.0.90}$$

By the law of large number, we know that the estimated gradient in eq. (0.0.90) is an *unbiased* estimate of the true policy gradient. Therefore, we can run *stochastic gradient ascent* with this estimated gradient. This forms the basis of the REINFORCE (Algorithm 18)

algorithm (version 1, we will show some improvements soon).

Start with an arbitrary initial policy π_θ
while *not converged* **do**
  Run simulator with π_θ to collect $\{\xi^{(i)}\}_{i=1}^N$
  Compute estimated gradient

$$\widetilde{\nabla}_\theta J = \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta\left(a_t^{(i)}|s_t\right)\right) R(\xi^{(i)})\right]$$

  Update parameters $\theta \leftarrow \theta + \alpha\,\widetilde{\nabla}_\theta J$
end
return π_θ

Algorithm 17: The REINFORCE algorithm.

In step 1, we run the simulator using the current policy to collect training sequences. In step 2, we approximate the expectation by the sample mean. Step 3 is the update rule of the algorithm with α being the step size. The algorithm is then repeated until convergence or until you are bored.

An example: Tetris

We will use Tetris as an example to show how you might choose your policy function $\pi_\theta(a|s)$ and how you would compute $\nabla_\theta \log \pi_\theta(a|s)$. Suppose we have some features representing the state-action pair of the Tetris game. For instance f_1 =the number of "holes" after the placement, f_2 =the height of the highest column after the placement, etc. Due to the log in eq. (0.0.90), a convenient stochastic policy is,

$$\pi_\theta(a|s) = \frac{\exp\left(\theta^\top f(s,a)\right)}{\sum_{a'}\exp\left(\theta^\top f(s,a')\right)}.$$

This is sometimes called the Boltzmann distribution or Gibbs distribution.

The gradient of the probability distribution can be computed by any method, e.g. using back-propagation. However, it is fairly simple to solve analytically:

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(a|s) &= \nabla_\theta\left[\theta^\top f(s,a) - \log\sum_{a'}\exp\left(\theta^\top f(s,a')\right)\right] \\
&= f(s,a) - \frac{\sum_{a'} f(s,a')\exp\left(\theta^\top f(s,a')\right)}{\sum_{a'}\exp\left(\theta^\top f(s,a')\right)} \\
&= f(s,a) - \sum_{a'} f(s,a')\,\pi_\theta\left(a'|s\right) \\
&= f(s,a) - E_{\pi_\theta(a'|s)}\left[f(s,a')\right]
\end{aligned} \tag{0.0.91}$$

This is essentially computing the difference between the feature at state s and action a versus the expectation over all actions for that state that we could have chosen, in a way the "average" feature. Assume that we observe that feature i for action a is *larger* than the average over all actions. According to eq. (0.0.91), if performing action a at state s produces a trajectory that has high reward, we will *increase* the value of θ_i to *upweight* this particular feature. Because it seems that this feature is "helpful" for getting high rewards. On the other hand, if this state-action pair produces low reward trajectories, we may conclude that feature i is "harmful". So we make the corresponding parameter θ_i to be small or negative to reflect this observation.

Reducing Variance

Although the estimated gradient in eq. (0.0.90) can in theory provide an unbiased estimate, it suffers from *high variance*. In order to see this, recall that the likelihood ratio policy gradient increases the probability of the trajectories with high reward and decreases the probability of the trajectories with low reward. However, imagine when *every* trajectory has a very high reward – although some are higher than others. Then, since we only has finite number of samples at each iteration, the estimated gradient will push the probability of all these trajectories higher (if possible) since the total reward is high (and hence make the probability of other trajectories lower). However, the algorithm has no idea about the reward of trajectories *compared to other trajectories*. Therefore, we can imagine that the estimated gradients are pointing in different directions at each iteration. In fact, without making the modifications introduced in this part, the REINFORCE algorithm performs poorly compared to "black-box" approaches.

One simple modification to reduce the variance is to take advantage of causality – the actions selected now cannot affect past rewards.

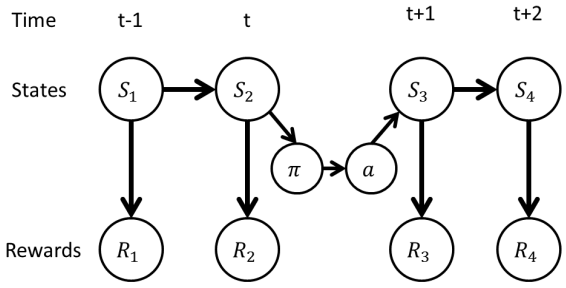

Figure 0.0.29: A trajectory of states, actions and rewards. We consider changing the action at time t in order to get a better expected future reward.

If we consider a trajectory of states and rewards, we want to change the action at time t to maximize expected reward. Intuitively, we know that changing the action at time t cannot affect the rewards obtained in the past, since we have already received them. Thus, we can represent our expected reward as only the future reward.

$$\begin{aligned}\nabla_\theta J &= E_{p(\xi|\theta)}\left[\sum_{t=0}^{T-1}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=0}^{t-1} r(s_{t'},a_{t'}) + \sum_{t'=t}^{T-1} r(s_{t'},a_{t'})\right)\right)\right] \\ &= E_{p(\xi|\theta)}\left[\sum_{t=0}^{T-1}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\sum_{t'=t}^{T-1} r(s_{t'},a_{t'})\right)\right],\end{aligned} \tag{0.0.92}$$

where $\sum_{t'=t}^{T-1} r(s_{t'},a_{t'})$ is sometimes called *future reward* or *reward-to-go*. We can use this idea to remove the dependence of past rewards from the calculation of our gradient.

One can reduce the variance even further by introducing *baselines* for the expected total rewards. Recall that one of the reasons for the high variance is that the algorithm does not know how well the trajectories perform *compared to other trajectories*. Therefore, by introducing a *baseline* for the total reward (or reward to go), we can update the policy based on how well the policy performs compared to a baseline. The variance can hopefully be reduced if the baseline approximates the average performance of the trajectories. But how do we know that whether the estimated gradient still makes sense?

Let's first take a look at the expectation $E_{p(\xi|\theta)}[\nabla_\theta \log p(\xi|\theta) b]$. We have,

$$\begin{aligned}E_{p(\xi|\theta)}[\nabla_\theta \log p(\xi|\theta)\, b] &= \sum_{\xi\in\Xi} \nabla_\theta\, p(\xi|\theta)\, b \\ &= \nabla_\theta \left(\sum_{\xi\in\Xi} p(\xi|\theta)\right) b \\ &= (\nabla_\theta 1)\; b = 0.\end{aligned} \tag{0.0.93}$$

Therefore, the estimated policy is still unbiased if we introduce a baseline for the total reward (or reward to go). Note here that the above equation holds as long as b does not depend on θ, hence b can potentially be a function of the state, i.e. $b = b(s_t)$.[12] In fact, a common choice of baseline is the value function or some estimate of the value function.

[12] However, some additional effort is needed to show that a time-dependent baseline actually works, including expanding $p(\xi|\theta)$ in the expectation as a product of the transition probability and the policy.

Putting everything together, we can generate another policy gradient expression,

$$\nabla_\theta J = E_{p(\xi|\theta)}\left[\sum_{t=0}^{T-1}\left(\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T-1} r(s_{t'},a_{t'}) - b(s_t)\right)\right)\right], \tag{0.0.94}$$

We estimate the above policy gradient as

$$\widetilde{\nabla}_\theta J = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \left(\nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t}^{T-1} r(s_{t'}^{(i)}, a_{t'}^{(i)}) - b(s_t^{(i)}) \right) \right). \tag{0.0.95}$$

This can give us an unbiased estimate of the policy gradients with lower variance.

The Policy Gradient Theorem

The REINFORCE algorithm calculates the gradient using expected future reward as determined by a trajectory.

$$\nabla_\theta J = E_{p(\xi|\theta)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r(s_{t'}, a_{t'}) \right]$$

We can instead replace the the estimate of future reward $\sum_{t'=t}^{T-1} r(s_{t'}, a_{t'})$ with the action value Q^{π_θ}, which by definition gives us the expected future reward.

$$\nabla_\theta J = E_{p(\xi|\theta)} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \, Q^{\pi_\theta}(s_t, a_t) \right]$$

We can update the gradient rule to take the expectation over the distribution of *states* rather than the expectation over the *trajectories*, this leads to the *Policy Gradient Theorem*.

$$\nabla_\theta J = E_{s\sim d^{\pi_\theta}(s), a\sim\pi_\theta(a|s)} \left[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \right] \tag{0.0.96}$$

Here, $d^{\pi_\theta}(s)$ is the distribution of states under policy π_θ, i.e., the fraction of time spent in state s,

$$d^{\pi_\theta}(s) = \frac{1}{T} \sum_{t=0}^{T-1} p^{\pi_\theta}(s, t),$$

where $p^{\pi_\theta}(s, t)$ is the probability that state s is visited at step t under policy π_θ.

The policy gradient theorem states that the gradient of average reward under a policy π_θ parametrized by θ is given by

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \, Q^{\pi_\theta}(s, a) \right] \tag{0.0.97}$$

The expectations are with respect to the distribution $d^{\pi_\theta}(s)$ of states given a policy π_θ and the actions taken under the policy π_θ given the state s. We can prove that, for the value function $V^{\pi_\theta}(s)$ is only a function of the state s, it can viewed as a *baseline* as we saw above. Thus, Eq. 0.0.97 is equal to:

$$\nabla_\theta J = E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \, (Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)) \right], \tag{0.0.98}$$

where $A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$ is referred to as the *advantage* of action a at state s under policy π_θ. So why is this true?First, consider the inner expectation. Because V^{π_θ} does not depend on a, this is equivalent to,

$$E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) V^{\pi_\theta}(s) \right] = V^{\pi_\theta}(s) \, E_{\pi_\theta(a|s)} \left[\nabla_\theta \log(\pi_\theta(a|s) \right]. \tag{0.0.99}$$

That leaves $\nabla_\theta \log(\pi_\theta(a|s)$ in the expectation. Intuitively that must be equal to zero because the probability distribution π_θ must sum to one, so the sum over all changes must be equal to zero. We show more explicitly below that this is indeed the case. We expand (Eq. 0.0.99) into sums over the states and actions. We can show that,

$$\begin{aligned} E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s)\right] &= \sum_{a\in\mathbb{A}} \pi_\theta(a|s)\, \nabla_\theta \log(\pi_\theta(a|s)) \\ &= \sum_{a\in\mathbb{A}} \pi_\theta(a|s)\, \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= \sum_{a\in\mathbb{A}} \nabla_\theta \pi_\theta(a|s) \\ &= \nabla_\theta \left(\sum_{a\in\mathbb{A}} \pi_\theta(a|s) \right) = \nabla_\theta 1 = 0. \end{aligned} \tag{0.0.100}$$

Through linearity of expectation, we have,

$$\begin{aligned} & E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s) V^{\pi_\theta}(s)\right] \\ =& E_{d^{\pi_\theta}(s)}\left[V^{\pi_\theta}(s)\, E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s)\right]\right] \\ =& E_{d^{\pi_\theta}(s)}\left[V^{\pi_\theta}(s)\cdot 0\right] = 0. \end{aligned} \tag{0.0.101}$$

Finally,

$$\begin{aligned} \nabla_\theta J &= E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right] \\ &= E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s)\, (Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s))\right] \\ &= E_{d^{\pi_\theta}(s)} E_{\pi_\theta(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s) A^{\pi_\theta}(s,a)\right] \end{aligned} \tag{0.0.102}$$

Intuitively, this shows that the algorithm wants the advantage of the action to be high, and wants to choose actions that are correlated with the advantage being high. It adjusts the policy by making small changes towards Q values that are higher than the average.

The policy gradient theorem connects estimating the gradient $\nabla_\theta J$ with estimating Q^{π_θ} or A^{π_θ}. For example,we can estimate Q^{π_θ} with some parameterized function $Q^{\pi_\theta}_\phi$ using *Approximate Dynamic Programming* methods like Fitted Q-Iteration or an advantage estimator $A^{\pi_\theta}_\phi$, to approximate the advantage function $A^{\pi_\theta}(s,a)$. Also, we can use samples trajectories under policy π_θ to estimate the expectation in Eq. (0.0.102), which results in an estimated policy gradient,

$$\widetilde{\nabla}_\theta J = \frac{1}{N} \sum_{i=1}^{N} \left(\nabla_\theta \log \pi_\theta(a_i|s_i)\, A^{\pi_\theta}_\phi(s_i,a_i) \right). \tag{0.0.103}$$

This leads to a class of methods called *Actor–Critic Methods*. Actor–Critic methods learn a *actor* (the policy) and a *critic* simultaneously.

The critic produces the estimate of some value function (e.g., state-value function, action-value function, advantage function, etc.) for bootstrapping (updating the value function estimate for a state from the estimated values of other states). By introducing the critic, the variance of the gradient estimate can be further reduced. Many popular policy gradient algorithms, including TRPO, PPO and DDPG, adopt the actor–critic architecture.

Examples

Let us consider a simple example of the actor-critic algorithm. Say we have two actions that we can take from a given state and one feature for the state s. One of our actions a_0 is bad, while the other one a_1 is good.

We use the Boltzmann distribution that we have seen in the previous example,

$$\pi_\theta(a|s) = \frac{\exp[\theta^\top f(s,a)]}{\sum_{a'} \exp[\theta^\top f(s,a)]}.$$

Suppose that the features of our state and the two actions are $f(s,a_0) = 3$ and $f(s,a_1) = 1$.

Let's say our current value of the parameter θ is $\theta = 1$. Then, the probabilities for taking each action are,

$$\pi_\theta(a_0|s) = \frac{\exp[\theta^\top f(s,a_0)]}{\exp[\theta^\top f(s,a_0)] + \exp[\theta^\top f(s,a_1)]} = \frac{e^3}{e^3+e} = \frac{e^2}{e^2+1} \approx 0.88,$$

$$\pi_\theta(a_1|s) = \frac{\exp[\theta^\top f(s,a_1)]}{\exp[\theta^\top f(s,a_0)] + \exp[\theta^\top f(s,a_1)]} = \frac{e}{e^3+e} = \frac{1}{e^2+1} \approx 0.12,$$

where $e \approx 2.71828$ is the base of the natural logarithm

We then get an estimate of the future reward, possibly through our critic: $Q^\pi(s,a_0) = 1$ and $Q^\pi(s,a_1) = 100$.

We have already seen previously that we can compute the derivative of the log probability as follows:

$$\nabla_\theta \log \pi_\theta(a|s) = f(s,a) - E_{\pi_\theta(a'|s)}[f(s,a')],$$

where,

$$E_{\pi(a'|s)}[f(s,a')] \approx 0.88 \times 3 + 0.12 \times 1 = 2.76.$$

We can just compute the gradient estimate[15],

$$\begin{aligned}
\widetilde{\nabla}_\theta J &= E_{a\sim\pi_\theta(a|s)}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right] \\
&= \pi_\theta(a_0|s)\nabla_\theta \log \pi_\theta(a_0|s) Q^\pi(s,a_0) \\
&\qquad + \pi_\theta(a_1|s)\nabla_\theta \log \pi_\theta(a_1|s) Q^\pi(s,a_1) \\
&\approx 0.88 \times (3 - 2.76) \times 1 + 0.12 \times (1 - 2.76) \times 100 \\
&\approx -20.79
\end{aligned}$$

[15] Note that this is an estimate because we are not taking expectation over state

Thus the policy gradient algorithm tells us to decrease the value of θ since the higher feature value seems to result in lower future reward. This makes the probability of choosing a_1 at s higher than the previous iteration.

Highly Correlated Features

Gradient ascent/descent methods depend greatly on the *parameterization* of the policy. To see this, consider the two parameterizations of Tetris.

Parameterization 1: f_1 = # of Holes after the placement, f_2 = Height after the placement. We use θ to denote the parameter for this parameterization.

Parameterization 2: $g_1 = \ldots = g_{100}$ = # of Holes after the placement, g_{101} = Height after the placement. We use ϕ to denote the parameter for this parameterization

Then, for Parameterization 1, we have,

$$\theta^\top f(x,a) = \theta_1 \times \#\text{ of Holes}(x,a) + \theta_2 \times \text{Height}(x,a).$$

While for Parameterization 2, we have,

$$\phi^\top g = \left(\sum_{i=1}^{100} \phi_i\right) \times \#\text{ of Holes}(x,a) + \phi_{101} \times \text{Height}(x,a).$$

When we take the policy gradient, we have,

$$\nabla_{\theta_i} J = E_{p(\xi|\theta)}\left[\sum_{t=0}^{T-1} \left(f_i(s,a) - E_{\pi_\theta(a'|s)}[f_i(s,a')]\right) Q^{\pi_\theta}(s_t,a_t)\right]$$

$$\nabla_{\phi_i} J = E_{p(\xi|\phi)}\left[\sum_{t=0}^{T-1} \left(g_i(s,a) - E_{\pi_\phi(a'|s)}[g_i(s,a')]\right) Q^{\pi_\phi}(s_t,a_t)\right]$$

Hence, we have $\nabla_{\phi_1} J = \ldots = \nabla_{\phi_{100}} J = \nabla_{\theta_1} J$. The policy gradient algorithm takes a 100 larger step for the *actual weight* corresponding to the number of holes using Parametrization 2 than in Parametrization 1!

Gradient ascent (or steepest ascent) poses the problem of finding $\max_{\Delta\theta} J(\theta + \Delta\theta)$ such that $\delta\theta$ is small. Gradient ascent measures "small" as the l-2 norm $\|\Delta\theta\|_2 = \sqrt{\sum_i (\Delta\theta_i)^2} \leq \epsilon$. However this version of measuring "small" depends on the parameterization of our policy. Ideally, we want the descent to measure "small" based on changes in our policy and not depend on the parameterization of the policy. We will address this problem in the next lecture.

Related Reading

[1] McNamara, A., Treuille, A., Popović, Z. and Stam, J., *Fluid control using the adjoint method*, ACM Transactions On Graphics (TOG) 2004.

[2] Krizhevsky, A., Sutskever, I. and Hinton, G.E., *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012.

[3] Le, Quoc V, *Building high-level features using large scale unsupervised learning*, Acoustics, Speech and Signal Processing (ICASSP), 2013.

Natural Policy Gradient

In the general formulation of steepest descent, as given by Eq. (0.0.104), there are many size metrics that can be utilized.

$$\max_{\Delta\theta} J(\theta + \Delta\theta) \qquad s.t. \qquad ||\Delta\theta|| \leq \epsilon \tag{0.0.104}$$

The gradient descent algorithm comes about when we choose the metric $||\cdot||$ to be the l_2 norm over the parameters ($\sqrt{\Delta\theta^\top \Delta\theta}$). In policy gradient methods such as REINFORCE, this definition of the metric can cause the algorithm to fail, if utilizing highly correlated features. This is due to the fact that the l_2 norm defines a "small" change in the gradient direction as depending on the cumulative sum in parameter change, which may have varying degrees of correlation with actual policy change. Instead, we would like to define the size metric such that the notion of "small" encompasses changes in the parameterized *policy*, not simply the changes in the parameters themselves. This leads to two questions.

Q1) What does steepest descent look like given other metrics?

Q2) What metric captures the fact that we would like our metric to be tied to the difference between the $\pi_\theta(a|s)$ and $\pi_{\theta+\Delta\theta}(a|s)$, and not just θ and $\theta + \Delta\theta$?

Q1 – What does steepest descent look like under other metrics?

For small changes in the parameters, we can think of the metric as some quadratic function of the parameters, as evidenced by the Taylor expansion. The steepest descent optimization problem then becomes

$$\max_{\Delta\theta} J(\theta + \Delta\theta) \qquad s.t. \qquad \Delta\theta^\top G(\theta)\Delta\theta \leq \epsilon \tag{0.0.105}$$

where $G(\theta)$ defines the specific metric. In general, G is a distance metric and thus is symmetric positive semidefinite[16]. This matrix defines the notion of distance in the parameter space locally around θ and, in some cases, can be constant; if this is true, the metric is referred to as flat. Intuitively, a flat metric entails that distance is measured the same everywhere in the parameter space. While a flat metric can be helpful, in the general case it will not accurately capture the true notion of distance on the parameter manifold.

[16] Being pedantic, it is actually a pseudometric if it has nontrivial nullspace

However, we don't always want to use flat metrics because it does not always precisely reflect what does "small" means in our particular situations. For example, one change of parameters $\Delta\theta$ at θ_1 can result in a very minor change of our policy, while the same $\Delta\theta$ can result in a large change at θ_2. We want our metric $G(\theta)$ to reflect that.

We can solve this new optimization problem (Eq. (0.0.105)) for the parameters using the technique of Lagrange multipliers. This convert the constrained optimization problem (0.0.105) to unconstrained optimization problem with respect to the *Lagrangian* of the system,

$$\max_{\Delta\theta} \mathcal{L}(\Delta\theta, \lambda) = J(\theta + \Delta\theta) - \lambda \left[\Delta\theta^\top G(\theta)\Delta\theta - \epsilon\right], \tag{0.0.106}$$

where $\lambda \geq 0$ is the *Lagrange multiplier*.

The theory says that there exists a choice of $\lambda \geq 0$ such that the constraint optimization problem (0.0.105) and the unconstrained optimization problem (0.0.106) has the same solution. But we won't worry about that in this lecture because we just want ϵ to be small in general, rather than care about the exact value of ϵ. Therefore, we just take λ to be a fixed scalar and solve for $\Delta\theta$.

Because we are only considering small steps in $\Delta\theta$, we can approximate (0.0.106) by using the first-order Taylor expansion of J:

$$\mathcal{L}(\Delta\theta, \lambda) \approx J(\theta) + \Delta\theta^\top \nabla_\theta J - \lambda \left[\Delta\theta^\top G(\theta)\Delta\theta - \epsilon\right] \doteq \tilde{\mathcal{L}}_\lambda(\Delta\theta). \tag{0.0.107}$$

Here we use the notation $\tilde{\mathcal{L}}_\lambda(\Delta\theta)$ to emphasize that we are taking λ as a constant and hence $\tilde{\mathcal{L}}_\lambda$ is a function of $\Delta\theta$.

Note that the approximated Lagrangian $\tilde{\mathcal{L}}$ is quadratic in $\Delta\theta$. To find the solution, we can simply take the partial derivative of the approximated Lagrangian $\tilde{\mathcal{L}}$ with respect to the change in parameters and set it to zero:

$$\frac{\partial \tilde{\mathcal{L}}_\lambda}{\partial \Delta\theta} = \nabla_\theta J - 2\,\lambda\, G(\theta)\,\Delta\theta = 0. \tag{0.0.108}$$

If $G(\theta)$ is nonsingular, the solution for the change in parameters is thus:

$$\Delta\theta = \frac{1}{2\,\lambda}\, G^{-1}(\theta)\, \nabla_\theta J. \tag{0.0.109}$$

Intuitively, we are taking the gradient and multiplying it by the inverse of the metric that defines what it means to be large, and then taking a step in that direction. However, it may still be the case that $G(\theta)$ is singular, or very close to singular, due to two features being very highly correlated. For example, if we are using two features that are exactly the same, the metric should look something like

$$G(\theta) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}.$$

In this case, if we make change $\Delta\theta = [\Delta\theta_1\ \Delta\theta_2]^\top$ to the parameters,

the size of this change measured by metric $G(\theta)$ is thus,

$$\begin{aligned}\Delta\theta^\top G(\theta)\Delta\theta &= \begin{bmatrix}\Delta\theta_1 & \Delta\theta_2\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}\Delta\theta_1\\ \Delta\theta_2\end{bmatrix}\\ &= \Delta\theta_1^2 + 2\Delta\theta_1\Delta\theta_2 + \Delta\theta_2^2\\ &= (\Delta\theta_1 + \Delta\theta_2)^2\end{aligned}$$

This means that changes in any of the features, or any combination of the features should be the same if they add up to be the same because they effectively act on the same feature. Because this matrix is singular, there exists a space in which we can move, and it will not change the policy at all (the nullspace of $G(\theta)$). For eample, we can add δ to the first parameter and subtract the second by δ, and the plicy is still the same. In this case, the most natural thing to do is use the pseudo-inverse, denoted as $G^\dagger(\theta)$, in place of the inverse, which means that we not trying to do anything in the nullspace, only the space in which we can actually affect things.

Q2 – What metric do we want to use for policy gradients?

Despite now knowing how to change and solve the optimization problem for different metrics, we are still left with the question of what the metric should be. It turns out that there is a canonical answer for probability distributions, given by Chentsov's theorem. This theorem effectively says that there is a unique metric such that distance is invariant to a class of changes to the problem, such as label switching, for parametric family of distributions; this metric is known as the *Fisher Information Metric* (Eq. 0.0.110).

$$G(\theta) = E_{p_\theta}\left[\nabla_\theta \log(p_\theta)\, \nabla_\theta \log(p_\theta)^\top\right] \tag{0.0.110}$$

Another way to come to this same result is to consider the Kullback–Leibler divergence, or K-L divergence, of two probability distributions. Given two probability distributions p and q,

$$\mathcal{KL}(p||q) = \sum_{x\in\mathbb{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right). \tag{0.0.111}$$

It turns out that the change in parameters measured by the Fisher Information Metric is exactly the second order approximation of the K-L divergence of the probability distributions before and after the change,

$$\begin{aligned}\mathcal{KL}(p_{\theta+\Delta\theta}\|p_\theta) &\approx \Delta\theta^\top G(\theta)\Delta\theta,\\ \mathcal{KL}(p_\theta\|p_{\theta+\Delta\theta}) &\approx \Delta\theta^\top G(\theta)\Delta\theta.\end{aligned} \tag{0.0.112}$$

In general, the second-order approximation of any obvious metric on probability distributions will result in the Fisher Information Metric.

For the specific problem of policy optimization, we take the Fisher Information Metric on trajectories as our metric (Eq. 0.0.113). This is because we want to essentially measure the distance between trajectories (distributions of states) given changes in parameters.

$$G(\theta) = E_{d^{\pi_\theta}(s),\pi_\theta(a|s)}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right], \qquad (0.0.113)$$

Recall that $d^{\pi_\theta}(s)$ is the distribution of states, or the fraction of time spent in states, under policy π_θ.

In practice, $G(\theta)$ can be estimated as a running average of the states experienced (Eq, 0.0.114), and its inclusion makes an enormous difference in the success of algorithms such as REINFORCE.

$$\tilde{G}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\nabla_\theta \log \pi_\theta(a_i|s_i) \nabla_\theta \log \pi_\theta(a_i|s_i)^\top\right] \qquad (0.0.114)$$

Intuitively, from a Machine Learning perspective, this algorithm is attempting to move in the direction that improves the performance the most, subject to changing the distribution of input examples as little as possible. This is also very similar to whitening of data, a natural normalization technique in Machine Learning.

In the general case, where we are just doing steepest descent with a distance metric, the algorithm is referred to as the covariant gradient method. In the special case shown above when you are measuring is distance between probability distributions, the algorithm is known as the natural gradient method.

Then, we can combine this estimated policy gradient with the natural gradient method, which gives us the update rule,

$$\Delta\theta = \frac{1}{2\lambda}\,\tilde{G}^{-1}(\theta)\,\widetilde{\nabla}_\theta J. \qquad (0.0.115)$$

This is known as the *Natural Policy Gradient* method. Note that Eq. (0.0.115) requires inverting the estimated Fisher information matrix, which can be computationally expensive when the number of parameters is large. One solution is to solve for Eq. (0.0.115) through iterative methods, e.g., Conjugate Gradient method, and terminate early. This in practice gives us reasonably good estimates of the natural policy gradient.

Conservative Policy Iteration

REINFORCE is essentially like a soft policy iteration, trying to change the probability of actions so that they are correlated with things that have high Q values. However, REINFORCE does not suffer from the disadvantages of policy iteration, because it makes small changes.

We can modify approximate policy iteration to avoid the problems caused by making big changes at each time step. We can make the policy iteration stochastic, by choosing to follow the old policy with probability α, and taking action $\text{argmax}_a \tilde{Q}(s, a)$ with probability $1 - \alpha$. This algorithm, known as *conservative policy iteration*, essentially makes a small change to the probability distribution over trajectories, but by choosing actions to go the steepest direction uphill.