2

BACKGROUND

2.1 MARKOV DECISION PROCESSES

A Markov Decision Process (MDP) is a mathematical object that describes an agent interacting with a stochastic environment. It is defined by the following components:

- $\mathcal{S}$: **state space**, a set of states of the environment.
- $\mathcal{A}$: **action space**, a set of actions, which the agent selects from at each timestep.
- $P(r, s' \mid s, a)$: a transition probability distribution. For each state s and action a, P specifies the probability that the environment will emit reward r and transition to state s'.

In certain problem settings, we will also be concerned with an **initial state distribution** $\mu(s)$, which is the probability distribution that the initial state s_0 is sampled from.

Various different definitions of MDP are used throughout the literature. Sometimes, the reward is defined as a deterministic function $R(s)$, $R(s, a)$, or $R(s, a, s')$. These formulations are equivalent in expressive power. That is, given a deterministic-reward formulation, we can simulate a stochastic reward by lumping the reward into the state.

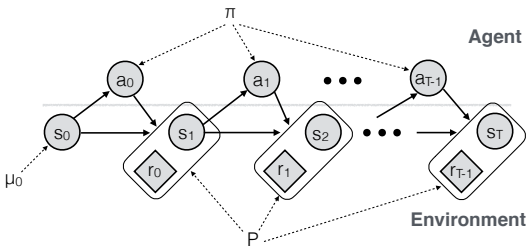
The end goal is to find a **policy** π, which maps states to actions. We will mostly consider stochastic policies, which are conditional distributions $\pi(a \mid s)$, though elsewhere in the literature, one frequently sees deterministic policies $a = \pi(s)$.

2.2 THE EPISODIC REINFORCEMENT LEARNING PROBLEM

This thesis will be focused on the episodic setting of reinforcement learning, where the agent's experience is broken up into a series of **episodes**—sequences with a finite number of states, actions and rewards. Episodic reinforcement learning in the fully-observed setting is defined by the following process. Each episode begins by sampling an initial state of the environment, s_0, from distribution $\mu(s_0)$. Each timestep $t = 0, 1, 2, \ldots$, the

agent chooses an action a_t, sampled from distribution $\pi(a_t \mid s_t)$. π is called the *policy*—it's the probability distribution that the agent uses to sample its actions. Then the enviroment generates the next state and reward, according to some distribution $P(s_{t+1}, r_t \mid s_t, a_t)$. The episode ends when a *terminal state* s_T is reached. This process can be described by the following equations or diagram below.

$$
\begin{aligned}
s_0 &\sim \mu(s_0) \\
a_0 &\sim \pi(a_0 \mid s_0) \\
s_1, r_0 &\sim P(s_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid s_1) \\
s_2, r_1 &\sim P(s_2, r_1 \mid s_1, a_1) \\
&\dots \\
a_{T-1} &\sim \pi(a_{T-1} \mid s_{T-1}) \\
s_T, r_{T-1} &\sim P(s_T \mid s_{T-1}, a_{T-1})
\end{aligned}
$$



The goal is to find a policy π that optimizes the expected total reward per episode.

$$
\begin{aligned}
&\underset{\pi}{\text{maximize}}\, E_\tau[R \mid \pi] \\
&\text{where } R = r_0 + r_1 + \cdots + r_{\text{length}(\tau)-1}
\end{aligned}
$$

The expectation is taken over trajectories τ, defined as the sequence of states, actions, and rewards, $(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$, ending in a terminal state. These trajectories are sampled using policy π to generate actions.

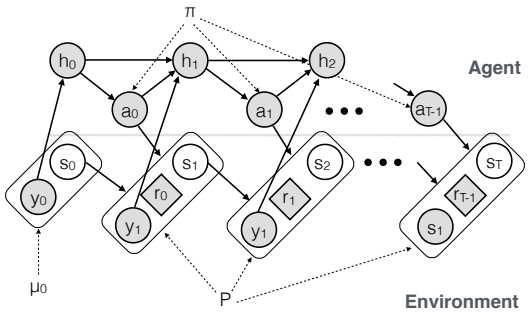
Note: expectation notation. $E_x[f(x) \mid z]$ is defined to mean $E_x[f(x) \mid z] = \int dx\, p(x \mid z) f(x)$. In words, the subscript is the random variable we are averaging over, and the conditioning expression (z) is a variable that affects the distribution over x. In a slight abuse of notation, we'll place functions like the policy π in this conditioning expression, i.e., $E_\tau[f(\tau) \mid \pi] = \int d\tau\, p_\pi(\tau) f(\tau)$, where $p_\pi(\tau)$ is the probability distribution of trajectories obtained by executing policy π.

2.3 PARTIALLY OBSERVED PROBLEMS

In the *partially-observed setting*, the agent only has access to an observation at each timestep, which may give noisy and incomplete information about the state. The agent

should combine information from many previous timesteps, so the action a_t depends on the preceding history $h_t = (y_0, a_0, y_1, a_1, \ldots, y_{t-1}, a_{t-1}, y_t)$. The data-generating process is given by the following equations, and the figure below.

$$
\begin{aligned}
s_0, y_0 &\sim \mu_0 \\
a_0 &\sim \pi(a_0 \mid h_0) \\
s_1, y_1, r_0 &\sim P(s_1, y_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid h_1) \\
s_2, y_2, r_1 &\sim P(s_2, y_2, r_1 \mid s_1, a_1) \\
&\ldots \\
a_{T-1} &\sim \pi(a_{T-1} \mid h_{T-1}) \\
s_T, y_T, r_{T-1} &\sim P(s_T, y_T, r_{T-1} \mid s_{T-1}, a_{T-1})
\end{aligned}
$$



This process is called a partially observed Markov decision process (POMDP). The partially-observed setting is equivalent to the fully-observed setting because we can call the observation history h_t the state of the system. That is, *a POMDP can be written as an MDP* (with infinite state space). When using function approximation, the partially observed setting is not much different conceptually from the fully-observed setting.

2.4 POLICIES

We'll typically use *parameterized stochastic policies*, which we'll write as $\pi_\theta(a \mid s)$. Here, $\theta \in \mathbb{R}^d$ is a parameter vector that specifies the policy. For example, if the policy is a neural network, θ would correspond to the flattened weights and biases of the network. The parameterization of the policy will depend on the action space of the MDP, and whether it is a discrete set or a continuous space. The following are sensible choices (but not the only choices) for how to how to define deterministic and stochastic neural network policies. With a discrete action space, we'll use a neural network that outputs action probabilities, i.e., the final layer is a softmax layer. With a continuous action space, we'll use a neural network that outputs the mean of a Gaussian distribution, with a separate set of parameters specifying a diagonal covariance matrix. Since the optimal policy in an MDP or POMDP is deterministic, we don't lose much by using a simple action distribution (e.g., a diagonal covariance matrix, rather than a full covariance matrix or a more complicated multi-model distribution.)

2.5 DERIVIATIVE FREE OPTIMIZATION OF POLICIES

Recall from the previous chapter that episodic reinforcement learning can be viewed as the following optimization problem:

$$\underset{\pi}{\text{maximize}}\, E[R \mid \pi]$$

where R is the total reward of an episode. If we choose a parameterized model π_θ for the policies, then this becomes an optimization problem with respect to $\theta \in \mathbb{R}^d$.

$$\underset{\theta}{\text{maximize}}\, E[R \mid \pi_\theta]$$

In derivative-free optimization, we treat the whole process for turning a parameter θ into a reward R as a black box, which gives us noisy evaluations $\theta \to \blacksquare \to R$, but we know nothing about what's inside the box.

A thorough discussion of derivative-free optimization algorithms is beyond the scope of this thesis. However, we'll introduce one algorithm, which is applicable in the noisy black-box optimization setting, and is used in comparisons later. Cross entropy method (CEM) is a simple but effective evolutionary algorithm, which works with Gaussian distributions, repeatedly updating the mean and variance of a distribution over candidate parameters. A simple instantiation is as follows.

Algorithm 1 Cross Entropy Method

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration = $1, 2, \ldots$ **do**
  Collect n samples of $\theta_i \sim N(\mu, \mathrm{diag}(\sigma))$
  Perform one episode with each θ_i, obtaining reward R_i
  Select the top p% of samples (e.g. $p = 20$), which we'll call the **elite set**
  Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new μ, σ.
end for
Return the final μ.

Algorithm 1 is prone to reducing the variance too quickly and converging to a bad local optimum. It can be improved by artificially adding extra variance, according to a schedule where this added noise decreases to zero. Details of this technique can be found in [SL06].

2.6 POLICY GRADIENTS

Policy gradient methods are a class of reinforcement learning algorithms that work by repeatedly estimating the gradient of the policy's performance with respect to its parameters. The simplest way to derive them is to use the *score function gradient estimator*, a general method for estimating gradients of expectations. Suppose that x is a random variable with probability density $p(x \mid \theta)$, f is a scalar-valued function (say, the reward), and we are interested in computing $\nabla_\theta E_x[f(x)]$. Then we have the following equality:

$$\nabla_\theta E_x[f(x)] = E_x[\nabla_\theta \log p(x \mid \theta) f(x)].$$

This equation can be derived by writing the expectation as an integral:

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \int dx\ p(x \mid \theta) f(x) = \int dx\ \nabla_\theta p(x \mid \theta) f(x) \\ &= \int dx\ p(x \mid \theta) \nabla_\theta \log p(x \mid \theta) f(x) = E_x[f(x) \nabla_\theta \log p(x \mid \theta)].\end{aligned}$$

To use this estimator, we can sample values $x \sim p(x \mid \theta)$, and compute the LHS of the equation above (averaged over N samples) to get an estimate of the gradient (which becomes increasingly accurate as $N \to \infty$. That is, we take $x_1, x_2, \ldots, x_N \sim p(x \mid \theta)$, and then take our gradient estimate $\hat{g}$ to be

$$\hat{g} = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p(x_i \mid \theta) f(x_i)$$

To use this idea in reinforcement learning, we will need to use a *stochastic policy*. That means that at each state s, our policy gives us a probability distribution over actions, which will be denoted $\pi(a \mid s)$. Since the policy also has a parameter vector θ, we'll write $\pi_\theta(a \mid s)$ or $\pi(a \mid s, \theta)$.

In the following discussion, a *trajectory* τ will refer to a sequence of states and actions $\tau \equiv (s_0, a_0, s_1, a_1, \ldots, s_T)$. Let $p(\tau \mid \theta)$ denote the probability of the entire trajectory τ under policy parameters θ, and let $R(\tau)$ denote the total reward of the trajectory.

The derivation of the score function gradient estimato tells us that

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log p(\tau \mid \theta) R(\tau)]$$

Next, we need to expand the quantity $\log p(\tau \mid \theta)$ to derive a practical formula. Using the chain rule of probabilities, we obtain

$$\begin{aligned}p(\tau \mid \theta) = &\mu(s_0) \pi(a_0 \mid s_0, \theta) P(s_1, r_0 \mid s_0, a_0) \pi(a_1 \mid s_1, \theta) \\ &P(s_2, r_1 \mid s_1, a_1) \ldots \pi(a_{T-1} \mid s_{T-1}, \theta) P(s_T, r_{T-1} \mid s_{T-1}, a_{T-1}),\end{aligned}$$

where μ is the initial state distribution. When we take the logarithm, the product turns into a sum, and when we differentiate with respect to θ, the terms $P(s_t \mid s_{t-1}, a_{t-1})$ terms drop out as does $\mu(s_0)$. We obtain

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) R(\tau)\right]$$

It is somewhat remarkable that we are able to compute the policy gradient without knowing anything about the system dynamics, which are encoded in transition probabilities P. The intuitive interpretation is that we collect a trajectory, and then *increase its log-probability proportionally to its goodness*. That is, if the reward $R(\tau)$ is very high, we ought to move in the the direction in parameter space that increases $\log p(\tau \mid \theta)$.

Note: trajectory lengths and time-dependence. Here, we are considering trajectories with fixed length T, whereas the definition of MDPs and POMDPs above assumed variable or infinite length, and *stationary* (time-independent) dynamics. The derivations in policy gradient methods are much easier to analyze with fixed length trajectories—otherwise we end up with infinite sums. The fixed-length case can be made to mostly subsume the variable-length case, by making T very large, and instead of trajectories ending, the system goes into a *sink* state with zero reward. As a result of using finite-length trajectories, certain quantities become time-dependent, because the problem is no longer stationary. However, we can include time in the state so that we don't need to separately account for the dependence on time. Thus, we will omit the time-dependence of various quantities below, such as the state-value function V^π.

We can derive versions of this formula that eliminate terms to reduce variance. This calculation is provided in much more generality in Chapter 5 on *stochastic computation graphs*, but we'll include it here because the concrete setting of this chapter will be easier to understand.

First, we can apply the above argument to compute the gradient for a single reward term:

$$\nabla_\theta E_\tau[r_t] = E_\tau\left[\sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta) r_t\right]$$

Note that the sum goes up to t, because the expectation over r_t can be written in terms of actions $a_{t'}$ with $t' \leqslant t$. Summing over time (taking $\sum_{t=0}^{T-1}$ of the above equation), we

get

$$\begin{aligned}\nabla_\theta E_\tau[R(\tau)] &= E_\tau\left[\sum_{t=0}^{T-1} r_t \sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta)\right] \\ &= E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}\right]. \end{aligned} \tag{1}$$

The second formula (Equation (1)) results from the first formula by reordering the summation. We will mostly work with the second formula, as it is more convenient for numerical implementation.

We can further reduce the variance of the policy gradient estimator by using a *baseline*: that is, we subtract a function $b(s_t)$ from the empirical returns, giving us the following formula for the policy gradient:

$$\nabla_\theta \mathbb{E}_\tau\left[R(\tau)\right] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta)\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t)\right)\right] \tag{2}$$

This equality holds for arbitrary baseline functions b. To derive it, we'll show that the added terms $b(s_t)$ have no effect on the expectation, i.e., that $\mathbb{E}_\tau\left[\nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta) b(s_t)\right] = 0$. To show this, split up the expectation over whole trajectories $\mathbb{E}_\tau[\dots]$ into an expectation over all variables before a_t, and all variables after and including it.

$$\begin{aligned}
&\mathbb{E}_\tau\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta) b(s_t)\right] \\
&= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta) b(s_t)\right]\right] && \text{(break up expectation)} \\
&= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T}, a_{t:(T-1)}}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta)\right]\right] && \text{(pull baseline term out)} \\
&= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_t \mid s_t, \theta)\right]\right] && \text{(remove irrelevant vars.)} \\
&= \mathbb{E}_{s_{0:t}, a_{0:(t-1)}}\left[b(s_t) \cdot 0\right]
\end{aligned}$$

The last equation follows because $\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta)\right] = \nabla_\theta \mathbb{E}_{a_t}[1] = 0$ by the definition of the score function gradient estimator.

A near-optimal choice of baseline is the state-value function,

$$V^\pi(s) = \mathbb{E}\left[r_t + r_{t+1} + \cdots + r_{T-1} \mid s_t = s,\ a_{t:(T-1)} \sim \pi\right]$$

See [GBB04] for a discussion of the choice of baseline that optimally reduces variance of the policy gradient estimator. So in practice, we will generally choose the baseline to approximate the value function, $b(s) \approx V^\pi(s)$.

We can intuitively justify the choice $b(s) \approx V^{\pi}(s)$ as follows. Suppose we collect a trajectory and compute a noisy gradient estimate

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}$$

which we will use to update our policy $\theta \rightarrow \theta + \epsilon \hat{g}$. This update increases the log-probability of a_t proportionally to the sum of rewards $r_t + r_{t+1} + \cdots + r_{T-1}$, following that action. In otherwords, if the sum of rewards is high, then the action was probably good, so we increase its probability. To get a better estimate of whether the action was good, we should check to see if the returns were *better than expected*. Before taking the action, the expected returns were $V^{\pi}(s_t)$. Thus, the difference $\sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is an approximate estimate of the goodness of action a_t—Chapter 4 discusses in a more precise way how it is an estimate of the *advantage function*. Including the baseline in our policy gradient estimator, we get

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi(a_t \mid s_t, \theta) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right),$$

which increases the probability of the actions that we infer to be good—meaning that the estimated advantage $\hat{A}_t = \sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ is positive.

If the trajectories are very long (i.e., T is high), then the preceding formula will have excessive variance. Thus, practitioners generally use a discount factor, which reduces variance at the cost of some bias. The following expression gives a biased estimator of the policy gradient.

$$\hat{g} = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right)$$

To reduce variance in this biased estimator, we should choose $b(s_t)$ to optimally estimate the *discounted sum of rewards*,

$$b(s) \approx V^{\pi,\gamma}(s) = \mathbb{E}\left[\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \;\middle|\; s_t = s; a_{t:(T-1)\sim\pi} \right]$$

Intuitively, the discount makes us pretend that the action a_t has no effect on the reward $r_{t'}$ for t' sufficiently far in the future, i.e., we are downweighting delayed effects by

a factor of $\gamma^{t'-t}$. By adding up a series with coefficients $1, \gamma, \gamma^2, \ldots$, we are effectively including $1/(1-\gamma)$ timesteps in the sum.

The policy gradient formulas given above can be used in a practical algorithm for optimizing policies.

Algorithm 2 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

In the algorithm above, the policy update can be performed with stochastic gradient ascent, $\theta \to \theta + \epsilon \hat{g}$, or one can use a more sophisticated method such as Adam [KB14].

To numerically compute the policy gradient estimate using automatic differentiation software, we swap the sum with the expectation in the policy gradient estimator:

$$\begin{aligned}\hat{g} &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right) \\ &= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \hat{A}_t\end{aligned}$$

Hence, one can construct the scalar quantity $\sum_t \log \pi_\theta(a_t|s_t)\hat{A}_t$ and differentiate it to obtain the policy gradient.

The vanilla policy gradient method described above has been well-known for a long time; some early papers include [Wil92; Sut+99; JJS94]. It was considered to be a poor choice on most problems because of its high sample complexity. A couple of other practical difficulties are that (1) it is hard to choose a stepsize that works for the entire course of the optimization, especially because the statistics of the states and rewards changes;

(2) often the policy prematurely converges to a nearly-deterministic policy with a suboptimal behavior. Simple methods to prevent this issue, such as adding an entropy bonus, usually fail.

The next two chapters in this thesis improve on the vanilla policy gradient method in two orthogonal ways, enabling us to obtain strong empirical results. Chapter 3 shows that instead of stepping in the gradient direction, we should move in the natural gradient direction, and that there is an effective way to choose stepsizes for reliable monotonic improvement. Chapter 4 provides much more detailed analysis of discounts, and Chapter 5 also revisits some of the variance reduction ideas we have just described, but in a more general setting. Concurrently with this thesis work, Mnih et al. [Mni+16] have shown that it is in fact possible to obtain state-of-the-art performance on various large-scale control tasks with the vanilla policy gradient method, however, the number of samples used for learning is extremely large.