1 Setup

The MDP is $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where $\mathcal{S}, \mathcal{A}$ are the (finite for now) state and action spaces, $P(s'|s,a)$ is the probability of transitioning from s to s' given action a, $R(s,a,s')$ is the reward function, $\gamma \in [0,1]$ is the discount factor (sometimes taken strictly less than 1 to make some proofs work under an infinite time horizon) [1]. A policy is a function $\pi : \mathcal{S} \to P(\mathcal{A})$ that defines a distribution over actions for each state. We want to find the policy π^* that maximizes the return:

$$\pi^* = \underset{\pi:\mathcal{S}\to\mathcal{A}}{\arg\max}\, \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1})\right]$$

where H is the (potentially infinite) time horizon and expectation over a policy means that we always attain the next state from the policy π and environment P.

2 Q function

We can also define the Q function for a policy π:

$$Q^\pi(s,a) = \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_0 = s, a_0 = a\right]$$

The first main result is the Bellman equation, which says that

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim P|s,a}\mathbb{E}_{a'\sim\pi|s'}\left[R(s,a,s') + \gamma Q^\pi(s',a')\right]$$

This follows from just expanding the definition:

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_0 = s, a_0 = a\right] \\
&= \mathbb{E}_{s'\sim P}\left[R(s_0, a_0, s') + \mathbb{E}_{P,\pi}\left[\sum_{t=1}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_1 = s'\right] \middle| s_0 = s, a_0 = a\right] \\
&= \mathbb{E}_{s'\sim P|s,a}\left[R(s,a,s') + \gamma \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_0 = s'\right]\right] \\
&= \mathbb{E}_{s'\sim P|s,a}\mathbb{E}_{a'\sim\pi|s'}\left[R(s,a,s') + \gamma Q^\pi(s',a')\right]
\end{aligned}$$

And the optimal value function is

$$Q^*(s,a) = Q^{\pi^*}(s,a) = \max_\pi \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \middle| s_0 = s, a_0 = a\right]$$

And this satisfies the optimal Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'\sim P|s,a}\mathbb{E}_{a'\sim\pi^*|s'}\left[R(s,a,s') + \gamma Q^*(s',a')\right] = \mathbb{E}_{s'\sim P|s,a}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

3 Dynamic Programming

We define the Bellman operator:

$$TQ(s,a) = \mathbb{E}_{s' \sim P|s,a} \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') \right]$$

We now define a dynamic programming algorithm where at each iteration we update for all pairs s, a:

$$Q_{t+1}(s,a) = TQ_t(s,a)$$

Above, we proved that Q^* is a fixed point of this operator, now we will prove that the Bellman operator is a contraction, thus converging to the unique fixed point (by Banach).

Take any two Q functions Q, Q' then at every s, a:

$$\begin{aligned}
|TQ(s,a) - TQ'(s,a)| &= \left| \mathbb{E}_{s' \sim P|s,a} \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - R(s,a,s') + \gamma \max_{a'} Q'(s',a') \right] \right| \\
&= \left| \mathbb{E}_{s' \sim P|s,a} \left[\gamma \max_{a'} Q(s',a') - \gamma \max_{a'} Q'(s',a') \right] \right| \\
&\leq \gamma \mathbb{E}_{s' \sim P|s,a} \left| \max_{a'} Q(s',a') - \max_{a'} Q'(s',a') \right| \\
&\leq \gamma \mathbb{E}_{s' \sim P|s,a} \max_{a'} |Q(s',a') - Q'(s',a')| \\
&\leq \gamma \max_{s'} \max_{a'} |Q(s',a') - Q'(s',a')| = \gamma \|Q - Q'\|_\infty
\end{aligned}$$

4 Learning from samples

Now we realize that it's unrealistic to assume that we know the functions P and R. So, we have to sample trajectories to be able to optimize. In the known-dynamics case our algorithm was:

$$Q_{t+1}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_t(s',a') \right]$$

We can estimate this expectation by repeatedly sampling actions and seeing which states the environment returns, yielding for some learning rate schedule β_k:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \beta_k \left(R(s_t,a_t,s_{t+1}) + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t) \right)$$

The way that actions a_k are sampled becomes a very important issue. For convergence, we cannot simply follow the greedy policy with respect to the current Q-function, but must perform some exploration. Q-learning is referred to as an "off policy" method since you can sample the actions from a different policy than the one which you are trying to estimate.

4.1 *Convergence (stochastic approximation)*

Full proof is in [2] and relies on some classic results from stochastic approximation theory [3]. We can look at the conditions of the proof and try to give some insight.

Conditions.

1. $\mathcal{S}$ and $\mathcal{A}$ are finite
2. $\sum_{i=0}^{\infty} \beta_i = \infty$ but $\sum_{i=0}^{\infty} (\beta_i)^2 < \infty$
3. All states and actions are visited infinitely often. Or the Markov chain induced by P and our sampling technique is irreducible and aperiodic.
4. The variance of $R(s, a, \cdot)$ is finite

Rough Sketch. We can subtract $Q^*(s_k, a_k)$ from both sides of the learning rule:

$$\begin{aligned} Q_{k+1}(s_k, a_k) - Q^*(s_k, a_k) &\leftarrow (1 - \beta_k)[Q_k(s_k, a_k) - Q^*(s_k, a_k)] \\ &\quad + \beta_k \left(R(s_k, a_k, s_{k+1}) + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q^*(s_k, a_k) \right) \end{aligned}$$

Then we define $\Delta_k(s_k, a_k) = Q_k(s_k, a_k) - Q^*(s_k, a_k)$ and $F_k(s_k, a_k) = R(s_k, a_k, s_{k+1}) + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q^*(s_k, a_k)$ so that the rule becomes

$$\Delta_{k+1}(s_k, a_k) \leftarrow (1 - \beta_k)\Delta_k(s_k, a_k) + \beta_k F_k(s_k, a_k)$$

Now we need to show that this is equivalent to a process of the form

$$w_{n+1}(x) \leftarrow (1 - \beta_n) w_n(x) + \beta_n r_n(x)$$

where $\mathbb{E}[r_n(x)|past] = 0$ and $\mathbb{E}[(r_n(x))^2|past] \leq C$ for some constant C with probability 1. Then we can appeal to a result from stochastic approximation that says this process will converge to 0, which would give us $\Delta_k \to 0 \implies Q_k \to Q^*$.

5 Function approximation

Now we realize that if the state and action spaces are too large, we cannot keep a table of the Q (or value) function for all states. Instead, we can approximate these functions. The first intuition is to try to minimize the squared error:

$$(Q_\theta(s, a) - Q^*(s, a))^2$$

we don't know Q^*, but we do have the strategy from above of using the Bellman update to move towards $\mathbb{E}_{s' \sim P}\left[R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')\right]$, to give us the objective:

$$(Q_\theta(s, a) - \mathbb{E}_{s'}[R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')])^2$$

To optimize by SGD, we want to take the gradient:

$$\begin{aligned} &\nabla_\theta (Q_\theta(s, a) - \mathbb{E}_{s'}[R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')])^2 \\ &= 2 \left(Q_\theta(s, a) - \mathbb{E}_{s'}[R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')] \right) \left(\nabla_\theta Q_\theta(s, a) - \nabla_\theta \mathbb{E}_{s'}[\gamma \max_{a'} Q_\theta(s', a')] \right) \end{aligned}$$

But we can't actually estimate this from a single sample of s' because it requires taking the product of two expectations over s'. This is the so called **double sample problem**. In practice (and in the lecture) this is swept under the rug by defining $target(s') = R(s, a, s') + \gamma \max_{a'} Q_\theta(s', a')$ and pretending that $target(s')$ does not depend on θ. This gives us the gradient:

$$\nabla_\theta (Q_\theta(s,a) - target(s'))^2 = 2(Q_\theta(s,a) - target(s'))\nabla_\theta Q_\theta(s,a)$$

which yields the SGD update:

$$\theta_{k+1} \leftarrow \theta_k + \beta \left(R(s,a,s') + \gamma \max_{a'} Q_\theta(s',a') - Q_{\theta_k}(s,a) \right) \nabla_{\theta_k} Q_{\theta_k}(s,a)$$

This has some issues theoretically and you can actually construct examples where this method will diverge. However, in practice this works well (with some tricks, see the DQN section).

5.1 *Linear functions*

A classic result is the convergence of TD(0) under linear function approximation [4].

Let $V^\pi_\theta(s) = s^T\theta$ where θ, s are both vectors in $\mathbb{R}^d$. We can think of value functions as vectors in $\mathbb{R}^{|\mathcal{S}|}$ and our operator P_π as a matrix in $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ specifying transition probabilities between the various states. Now we can write our update rule as:

$$\theta_{t+1} = \theta_t + \alpha s_t \left(R(s_t, a_t, s_{t+1}) + \gamma s_{t+1}^T \theta_t - s_t^T \theta_t \right).$$

By the stochastic approximation theory [3], we move to the continuous time ODE:

$$\begin{aligned} \dot{\theta} &= \mathbb{E}_{s,\pi,P}[\delta] = \mathbb{E}_{s\sim\mu, a\sim\pi, s'\sim P}[sR(s,a,s')] + \mathbb{E}_{s,a,s'}[s(\gamma s'^T - s^T)]\theta_t \\ &= b - A\theta_t. \end{aligned}$$

To get this linear dynamical system to converge, we need A to have positive eigenvalues. Define $\Phi \in \mathbb{R}^{|\mathcal{S}|\times d}$ to be the matrix of state representations and D_μ to be the diagonal matrix of $\mu(s)$ for μ the stationary distribution under our Markov process with policy π and write:

$$A = -\mathbb{E}_{s,a,s'}[s(\gamma s'^T - s^T)] = \Phi^T D_\mu (I - \gamma P_\pi)\Phi.$$

We assume that Φ is full rank. Then, proving convergence amounts to proving that $D_\mu(I - \gamma P_\pi)$ is positive definite. Since this matrix is not symmetric, we must show that it plus its transpose is positive definite. This can be done by showing that the row and column sums are positive for the matrix:

$$D_\mu(I - \gamma P_\pi)$$

Column sums can be calculated by multiplying by $\mathbf{1}^T$ and rows by $\mathbf{1}$.

$$\begin{aligned} \mathbf{1}^T D_\mu(I - \gamma P_\pi) &= \mathbf{1}^T D_\mu(I - \gamma P_\pi) \\ &= (\mu^T - \gamma \mu^T P_\pi) \\ &= (\mu^T - \gamma\mu^T) > \mathbf{0}^T \\ D_\mu(I - \gamma P_\pi)\mathbf{1} &= \mu - \gamma D_\mu P_\pi \mathbf{1} \\ &= \mu - \gamma D_\mu \mathbf{1} \\ &= (\mu - \gamma\mu) > \mathbf{0} \end{aligned}$$

We used the facts that μ is the stationary distribution, that $\gamma < 1$, and that P_π is a stochastic matrix so $P_\pi \mu \leq \mu$. So, we have our convergence. A similar proof holds for Q-learning with an additional assumption about the behavior policy being near to the optimal policy [5].

5.2 Experience replay

A lot of the recent success in RL uses neural networks in variations of Q-learning. To make this work in practice, a few more tricks (as well as a lot of compute) are required [6].

Rather than make the updates directly from sampled trajectories, each transition in each trajectory is placed into a buffer. Then the buffer is sampled from to break the temporal dependence between transitions (potentially with more important transitions sampled with higher probability). Without this, the network tends to adjust too much to the current trajectory. Using this technique requires using an off-policy method like Q-learning.

5.3 Double Q-learning

One observation is that by using $\max_a Q_k(s_{k+1}, a')$ as the bootstrap estimate of the value of s_{k+1}, we are being overly optimistic. Even if all of our Q values are unbiased estimators of Q^*, there is variance in these estimates since they are calculated from samples. So, taking the max of these estimates, we are almost assured to get a value that is greater than the true max. One solution to this is to train two Q networks Q_1, Q_2 and at each update flip a coin and update one network using the other to estimate the value of s':

$$Q_1(s_k, a_k) = Q_1(s_k, a_k) + \beta_k \left(R(s_k, a_k, s_{k+1}) + \gamma Q_2 \left(s_{k+1}, \arg\max_{a'} Q_1(s_{k+1}, a') \right) - Q_1(s_k, a_k) \right)$$

It is computationally easier to just keep a frozen network at some time as Q_2, if not quite as good. In practice people often use:

$$\theta_{k+1} = \theta_k + \alpha \left(R(s, a, s') + \gamma Q_{\theta^-} \left(s', \arg\max_{a'} Q_{\theta_k}(s', a') \right) - Q_{\theta_k}(s, a) \right) \nabla_{\theta_k} Q_{\theta_k}(s, a).$$

6 Policy Gradient

So far, we have been finding optimal policies by estimating value or Q-functions and greedily (or nearly greedily) choosing a policy with respect to this value function. An alternative is to directly find some function $\pi : S \to A$. We can instead define a parameterized stochastic policy by letting $\pi_\theta : S \to \mathcal{P}(A)$ where $\mathcal{P}(A)$ is the set of all probability distributions over A. Let $R(\tau)$ be the cumulative reward over a trajectory. Then we can solve the problem

$$\arg\max_{\pi} \mathbb{E}_{\tau \sim \pi, P} \left[R(\tau) \right] \qquad \sim \qquad \arg\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta, P} \left[R(\tau) \right]$$

We just optimize this by gradient descent (gradient derived below by the score function estimator). Note that this move to stochastic policies is hiding the fact that the algorithm is in fact a derivative-free method [7]. This is because we only access $R(\tau)$ by function evaluations, the gradients are with respect to a distribution we introduced to guide our exploration.

The question becomes how to take the gradient with respect of the expected reward when following the parameterized policy. The solution is to use the "score function gradient estimator". This presentation follows [8]. To simplify notation, let τ indicate a full trajectory $(s_0, a_1, s_1, \dots, s_T)$ of length T. Let $\tau \sim \theta$ indicate that τ is sampled from the distribution given by π_θ and environment dynamics P. Then the score function gradient estimator can be derived:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\tau\sim\theta}[R(\tau)] &= \nabla_\theta \int R(\tau)P(\tau|\theta)d\tau = \int R(\tau)\nabla_\theta(P(\tau|\theta))d\tau = \int P(\tau|\theta)R(\tau)\frac{\nabla_\theta(P(\tau|\theta))}{P(\tau|\theta)}d\tau \\ &= \mathbb{E}_{\tau\sim\theta}[R(\tau)\nabla_\theta \log P(\tau|\theta)].\end{aligned}$$

Now we can get an estimate of this expectation by sampling trajectories according to π^θ and calculating the rewards and gradients of the log likelihood of those trajectories according to our parameterized policy. Note that $P(\tau|\theta) = \rho(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0, a_1)\dots$ by the chain rule of probability. Since we take the log this becomes a sum, and the gradient will zero out the environmental probabilities, so we get:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\tau\sim\theta}[R(\tau)] &= \mathbb{E}_{\tau\sim\theta}\left[R(\tau)\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)\right] \\ &= \mathbb{E}_{\tau\sim\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)\sum_{t'=t}^{T-1}R(s_{t'}, a_{t'}, s_{t'+1})\right]\end{aligned}$$

where the last step just comes from writing out the whole reward and switching the order of summation.

6.1 Baseline

Now, we note that sampling this expectation in practice has very high variance. One way to reduce the variance somewhat is to subtract a state-dependent baseline (which in our case will be an estimate of the value function under the policy). To see why this still gives us an unbiased estimator of the gradient, note:

$$\begin{aligned}\mathbb{E}_{\tau\sim\theta}[\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)V(s_t)] &= \mathbb{E}_{s_{0:t},a_{1,t}}[\mathbb{E}_{s_{t+1:T},a_{t+1:T}}[\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)V(s_t)]] \\ &= \mathbb{E}_{s_{0:t},a_{1,t}}[V(s_t)\mathbb{E}_{a_{t+1}}[\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)]] \\ &= \mathbb{E}_{s_{0:t},a_{1,t}}[V(s_t)\nabla_\theta \mathbb{E}_{a_{t+1}}[1]] = 0\end{aligned}$$

where the last step comes from inverting the score function gradient estimator derived above. Now we can use:

$$\mathbb{E}_{\tau\sim\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_{t+1}|s_t)\left(\sum_{t'=t}^{T-1}(R(s_{t'}, a_{t'}, s_{t'+1})) - V(s_t)\right)\right]$$

This gives us an algorithm to estimate the gradient and use some gradient optimizer to maximize the reward with respect to our parameterized policy. At each step we sample some trajectories, estimate a value function, then take the above sum over the trajectories.

References

[1] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* 2017.

[2] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[3] Albert Benveniste, Pierre Priouret, and Michel Métivier. *Adaptive Algorithms and Stochastic Approximations.* Springer-Verlag, Berlin, Heidelberg, 1990.

[4] John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1075–1081. MIT Press, 1997.

[5] Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 664–671, New York, NY, USA, 2008. ACM.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. NIPS Deep Learning Workshop.

[7] B. Recht. A Tour of Reinforcement Learning: The View from Continuous Control. *ArXiv e-prints*, June 2018.

[8] John Schulman. Optimizing expectations: From deep reinforcement learning to stochastic com- putation graphs. *PhD thesis, EECS Department, University of California, Berkeley*, 2016.