Statistical Techniques in Robotics 16-831 Sping 2021 - The Robotics Institute

Homework 4: Reinforcement Learning

Deadline: April 21th, 2021 11:59 PM

Instructor: Kris Kitani TA: Xingyu Lin
Total points: 100

April 8, 2021

1 Introduction

In this homework, we are going to look at Reinforcement Learning (RL). We will start off with some theoretical questions on Policy Gradients and Temporal Difference Learning. Then, we will move on to the implementation of the basic approaches for RL like Value Iteration, Temporal Difference Learning and Policy Iteration.

1.1 Instructions

Submit this homework on Gradescope. Make sure your Gradescope student account is made using your **Andrew email ID** and your official name in the CMU system - this is essential to correctly record grades. Remember, you can only use a maximum of 2 late days (out of 5 for the semester) for any assignment. Beyond that, you will be penalized by a $1/3^{\text{rd}}$ deduction in points per day. Detailed instructions on what to submit are given in Section 4.

You are required to use **Python** for this homework. It is mandatory to **type-set** your answers; images of hand-written documents will not be accepted. Detailed instructions on what to submit are given in Section 4.

2 Reinforcement Learning (50 points)

In this problem, we are going to (1) study Policy Gradient and (2) Temporal Difference (TD) Learning. Policy Gradient and Temporal Difference are the two main building blocks of modern deep RL approaches. In fact, most of the deep RL methods are simply a combination of policy gradient and temporal difference learning with small modifications such as adding a replay buffer (which happens to be an old idea developed here in CMU in 1992[2]).

We will work on a Markov Decision Process (MDP) defined as $(\mathcal{S}, \mathcal{A}, P, r, H, \rho)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition dynamics such that $P(s'|s,a)$ is the probability of reaching state s' from s by taking action a; $r(s,a)$ is the reward of taking action a at state s and H is the planning horizon; ρ is the initial distribution where we sample the initial state s_1. We do not assume $\mathcal{S}$ is discrete, but for simplicity we assume that $\mathcal{A}$ is discrete, i.e., we only have $|\mathcal{A}|$ number of actions. Note that the policy gradient and temporal difference learning algorithms can work for both discrete and continuous setting. But we will focus on discrete action setting here.

We will define policy $\pi : \mathcal{S} \to \Delta(|\mathcal{A}|)$, namely $\pi(a|s)$ is the probability of π taking action a at state s. Given the MDP and a policy π, let us define a trajectory as $\tau = \{s_1, a_1, s_2, a_2, ..., s_H, a_H, s_{H+1}\}$, where $s_1 \sim \rho$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$. Define the total reward of such trajectory as $R(\tau) = \sum_{t=1}^{H} r(s_t, a_t)$.

2.1 Policy Gradient

Given the policy π parameterized by θ, which can be the weights of a neural network, our goal is to maximize the expected reward under this policy. Formally, we aim to find π that maximizes

$$J(\theta) = \int_{\tau} P(\tau; \pi_\theta) R(\tau) d\tau \tag{1}$$

where you can think of τ as a very long vector stacked with many shorter vectors $s_1, a_1, ...s_H, a_H, s_{H+1}$. Our goal is to compute the gradient of $J(\pi)$ with respect to policy π.

During the lecture, you have learned that the gradient of this objective function, i.e. the policy gradient $\nabla_\theta J(\theta)$ can be written as:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\nabla_\theta \ln(\pi(a_t|s_t;\theta)) q_\pi(s_t, a_t) \right], \tag{2}$$

where $q_\pi(s_t, a_t)$ is the expected return at state s_t with action a_t. The proof of the policy gradient theorem in the the lecture involves the use of causality. In this homework, we will prove the policy gradient theorem in a different way by utilizing the stationary state distribution.

2.1.1 Q function and value function

Denote the Q function and value function at state s under policy π as $q_\pi(s,a)$ and $v_\pi(s)$. Note that we are discussing the episodic case and there is no discounting.

Question (2 points)

1. Expand $v_\pi(s)$ in one step and represent it using $q_\pi(s,a)$.
2. Expand $q_\pi(s,a)$ in one step and represent it using $v_\pi(s')$ and $P(s'|s,a)$, where s' denotes the state in the next time step..

For both questions you don't need to show the proof.

2.1.2 Notation of state transition

As a first step to prove the policy gradient theorem, let's write the policy gradient in terms of a sum of state-wise Q function. Here we utilize an auxiliary function $\Pr(s \to x, k, \pi)$ which is the probability of transitioning from state s to state x in k steps under policy π.
Question (3 points) Write out $\Pr(s \to x, 1, \pi)$ and $\Pr(s \to x, 2, \pi)$ using the transition function $P(s'|s, a)$ as well as the policy $\pi(a|s)$.

2.1.3 Policy Gradient Theorem

Question (5 points) The policy gradient $\nabla J(\theta)$ can also be written as $\nabla v_\pi(s_1)$, where s_0 is the initial state (or a random variable in the case of stochastic initial states). Show that

$$\nabla v_\pi(s_1) = \sum_x \sum_{k=0}^{H} \Pr(s \to x, k, x) \sum_a \nabla \pi(a|x) q_\pi(x, a)$$

Hint: Keep unrolling the state into the future to get the term $\Pr(s \to x, k, \pi)$. Differentiate into each term using the product rule.

Denote $\eta(s) = \sum_{k=0}^{H} \Pr(s_0 \to s, k, \pi)$. Then the normalized version $\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$ is actually the on policy distribution of policy π, which we can directly sample from! Now show that

$$\nabla J(\theta) = \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a) \tag{3}$$

$$\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a). \tag{4}$$

In the episodic case, the constant of proportionality is the average length of an episode and in the continuing case (where we sum the time step to ∞ instead of H), the constant is 1, so that the relationship is actually an equality.

2.1.4 Get the gradient estimator

Now that we know

$$\nabla J(\theta) = \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a|s) \tag{5}$$

$$\propto \mathbb{E}_{s_t \sim \pi} \Big[\sum_a q_\pi(s_t, a) \nabla \pi(a|s_t) \Big], \tag{6}$$

it is easy to get the policy gradient estimator that we learn from class. Note the difference between a and a_t in our notation a is under $\sum$ and is a known value while a_t is a random variable.

Question (5 points) Show that

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[\nabla_\theta \ln(\pi(a_t|s_t;\theta)) q_\pi(s_t, a_t) \right]$$

2.1.5 Variance Reduction

For the questions below, we will use a slightly different but equivalent form of the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t|s_t;\theta)) R(\tau) \right] \tag{7}$$

In practice, we cannot compute the expectation in Eq. 7. Instead, we can form an empirical estimate of the policy gradient by samples. Given a policy $\pi(a|s;\theta)$, let us execute π on the MDP to generate K iid trajectories $\tau_i = \{s_1^i, a_1^i, ..., s_H^i, a_H^i\}, i = 1, ..., K$. We would do this by generating τ_1 first, and then resetting the system to generate τ_2 to make sure τ_1 and τ_2 are independent. Now the unbiased empirical estimate of the policy gradient can be written using the sampled trajectories:

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t^i|s_t^i;\theta)) R(\tau_i) \right] \tag{8}$$

The above empirical estimate is unbiased but it may have large variance, due to $R(\tau_i) = \sum_{t=1}^{H} r(s_t, a_t)$, which sums over all rewards along the entire horizon H. We have $Var(R(\tau_i))$ where $\tau_i \sim P(\tau;\theta)$ as follows:

$$Var(R(\tau)) = \sum_{t=1}^{H} Var(r(s_t, a_t)) + \sum_{i \neq j} Cov(r(s_i, a_i), r(s_j, a_j)) \tag{9}$$

where $Cov(x, y)$ is the covariance between random variables x and y. As we can see, the variance $Var(R(\tau))$ grows in the order of H^2 due to the covariance between any two time steps i and j. Hence the larger H is, the larger its variance becomes.

Question (5 points) Consider the simple case where $r(s, a) = c \in \mathbb{R} \quad \forall \quad s \in \mathcal{S}, a \in \mathcal{A}$. Prove that

$$Var\left(\widehat{\nabla_\theta J(\theta)}\right) = O\left(c^2\right) \tag{10}$$

2.1.6 Variance Reduction Trick 1

Let us switch back again to the real policy gradient shown in Eq. 7 and study two common tricks to reduce the variance. Note that there is **no question** for this section.

The first trick is to re-write Eq. 7 as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(\sum_{i=t}^{H} r(s_i, a_i) \right) \right) \right] \tag{11}$$

Namely, we replace $R(\tau)$ by $\sum_{i=t}^{H} r(s_i, a_i)$ for (s_t, a_t) at time step t. We can verify that Eq. 11 is equal to Eq. 7. We have reduced the variance because we truncated the horizon: instead of using $R(\tau)$ for every pair s_t, a_t at any t, we only use the $\sum_{i=t}^{H} r(s_i, a_i)$. This can be termed as reward-to-go, as it sums over the future rewards starting from and including time step t, while ignoring all rewards before time step t. One can prove the equality between Eq. 11 and Eq. 7 by using the intuition that the action a_t at time step t will affect the future rewards starting from and including time step t, but will not affect any past rewards from time step 1 to $t-1$.

2.1.7 Variance Reduction Trick 2

We can further reduce the variance by adding a *baseline* to Eq. 11. Let us define a baseline as a function $V : \mathcal{S} \to \mathbb{R}$, namely V takes state s as input and outputs a scalar. Consider the following policy gradient form:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(\sum_{i=t}^{H} r(s_i, a_i) - V(s_t) \right) \right) \right] \tag{12}$$

We claim that Eq. 12 is equal to Eq. 7. We can prove it by showing the following is true:

$$\mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(V(s_t) \right) \right) \right] = 0 \tag{13}$$

Question (5 points) Prove Eq. 13. Here is a partial proof. Your job is to fill in the rest.

$$\begin{aligned} &\mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(V(s_t) \right) \right) \right] \\ &= \sum_{t=1}^{H} \left[\mathbb{E}_{s_t \sim P_t(s;\theta)} \left[\mathbb{E}_{a_t \sim \pi(a|s_t;\theta)} \nabla_\theta \ln(\pi(a_t|s_t;\theta)) V(s_t) \right] \right] \\ &= \ldots \end{aligned} \tag{14}$$

where $P_t(s;\theta)$ is the state distribution at time step t under policy $\pi(a|s;\theta)$. *Hint:* You need to show that $\mathbb{E}_{a_t \sim \pi(a|s_t;\theta)} \nabla_\theta \ln(\pi(a_t|s_t;\theta)) V(s_t) = 0$.

Hence any function $V(s)$ can be used as a baseline, as long as it does not dependent on action a. We refer you to [1] for how to choose baselines and how a baseline could decrease the variance. The intuitive explanation is that if we have

a random variable X and a random variable Y which is highly correlated with X, then the variance of $X - Y$ could potentially be smaller than the variance of X and variance of Y. This is because:

$$Var(X - Y) = Var(X) + Var(Y) - 2Cov(X, Y). \tag{15}$$

So, if X and Y are highly correlated, i.e., $2Cov(X, Y)$ is bigger than $Var(Y)$, then $Var(X - Y) < Var(X)$. The extreme case is that $X = Y$. In this case $Cov(X, Y) = Var(X) = Var(Y)$, and $Var(X - Y) = 0$, which is less than $Var(X)$ and $Var(Y)$. Back to the policy gradient case, think about X as $\sum_{i=t}^{H} r(s_i, a_i)$ while Y as the baseline $V(s_t)$. Hence intuitively we want to choose some baseline $V(s_t)$ that is correlated with the random variable $\sum_{i=t}^{H} r(s_t, a_t)$.

2.1.8 Putting Everything Together

Now we have shown we can plug in any baseline function into Eq. 12, as long as $V(s)$ is independent of the actions. A good choice is to use $V^{\pi}(s)$, the value-to-go of the current policy π, if we can somehow estimate V^{π}.

We will now derive a minimium variance estimate of the baseline b in context of the episodic REINFORCE gradient estimator. After executing K trajectories, each coefficient (corresponding to each weight θ_h) of the empirical estimate of the gradient vector is given by:

$$\widehat{\nabla_{\theta_h} J(\theta)} = \frac{1}{K} \sum_{k=1}^{K} \left[\sum_{t=1}^{H} \nabla_{\theta_h} \ln(\pi(a_t^k | s_t^k; \theta)) \left(\sum_{i=0}^{H} r(s_i^k, a_i^k) - b_h \right) \right] \tag{16}$$

Question (5 points) Derive the optimal baseline $b_h \in \mathbb{R}$ that minimizes the variance of $\widehat{\nabla_{\theta_h} J(\theta)}$. How is this related to the expected reward?

Hint: You might find your results from 2.1.7 useful. You may leave your answer in expectations.

If $f(\tau) = \sum_{t=1}^{H} \nabla_{\theta_h} \ln(\pi(a_t | s_t; \theta))$ and $R(\tau) = \sum_{i=0}^{H} r(s_i, a_i)$, look at your result for this question and think about what happens if $f(\tau)$ and $R(\tau)$ are independent.

2.2 Temporal Difference Learning

In this section, we are interested in estimating the value-to-go V^π for a given policy π. This problem is usually called Policy Prediction in the RL literature. There are many methods for policy prediction, and we here will focus on the Temporal Difference Learning algorithm (TD).

Note that to make sure TD works, we need to switch from a finite, undiscounted MDP to a discounted MDP. Again, let us define the discounted MDP as $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho)$, where $\gamma \in (0, 1)$ is the discounted factor. We are going to use function approximator $f(s; \theta)$ to estimate $V^\pi(s)$. You can think of $f(s; \theta)$ as a linear regressor or a neural network parameterized by θ. Our objective is to find a θ that minimizes the prediction error:

$$\mathbb{E}_{s\sim v}\left[||f(s;\theta) - V^\pi(s)||_2^2\right] \tag{17}$$

where v is some distribution over the state space $\mathcal{S}$ which we assume is given. You can think of v as a prior distribution that puts different weights on different states; it could be the uniform distribution if we believe that every state is equally important.

However, we cannot directly optimize Eq. 17 with respect to θ. Why? Because we simply do not know V^π. You might say that we can easily get an unbiased estimate of $V^\pi(s)$ by executing the policy π from state s, and summing over all received rewards. Namely, $\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t)$ is the unbiased estimate of $V^\pi(s)$, where we set $s_1 = s$, and $a_t \sim \pi(a|s_t)$. However, as we explained in the previous section, $\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t)$ may be unbiased but it has large variance.

We need to reduce the variance. Temporal Difference (TD) uses the idea of *Bootstrap* to reduce the variance.[1]

2.2.1 Bellman Equation

Recall the definition of value function:

$$V^\pi(s) = \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) | s_1 = s, a_i \sim \pi(a|s_i) \forall i]$$

which is the expected total reward of executing π from state s.

Question (5 points) Prove the Bellman Equation. Namely, show that for any state s,

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^\pi(s')]. \tag{18}$$

[1] However, note that TD introduces bias; it trades reducing variance for bias. Trading bias for variance and vice versa is usually called the *Bias-Variance Tradeoff* in Machine Learning literature. Intuitively, bias and variance together determine the performance of our learning algorithms. Ideally we want no bias and low variance, but sometimes we just cannot have both. Policy Prediction is such an example.

2.2.2 Small Bellman Error Leads to Small Prediction Error

Define the Bellman Error for a function approximator $f(s;\theta)$ at state s as:

$$BE(s;\theta) = \left| f(s;\theta) - \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta)] \right| \tag{19}$$

Note that if there is a parameterization θ^* such that $f(s;\theta^*) = V^\pi(s)$, then $BE(s;\theta^*) = 0$, which can be easily verified by using Eq. 18. Hence, intuitively minimizing $BE(\theta)$ with respect to θ seems like a good idea. However, in practice, due to the modeling capacity of our function approximator $f(\cdot;\theta)$, there may be no such θ^* which makes $f(s;\theta^*) = V^\pi(s)$ for every state s. What we can realistically hope in practice is that we can find a θ^* such that the Bellman Error is small: $BE(s;\theta^*) \leq c, \forall s \in \mathcal{S}$, where $c \in \mathbb{R}^+$ is a small positive real number.

Question (5 points) Assume we find θ^* such that $BE(s;\theta^*) \leq c \in \mathbb{R}^+$, for all $s \in \mathcal{S}$. Prove that the prediction error is bounded in terms of c as follows:

$$\left| f(s;\theta^*) - V^\pi(s) \right| \leq \frac{1}{1-\gamma} c, \forall s \in \mathcal{S} \tag{20}$$

Hint: Use the Bellman Equality to re-write $V^\pi(s)$ as

$$\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^\pi(s')] \tag{21}$$

and then re-write $f(s;\theta^*)$ as:

$$\begin{aligned} f(s;\theta^*) =& f(s;\theta^*) - \left(\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta^*)]\right) \\ &+ \left(\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta^*)]\right) \end{aligned} \tag{22}$$

You may also need to use the inequality $|a+b| \leq |a| + |b|$, and $|\mathbb{E}_x(g(x))| \leq \mathbb{E}_x|g(x)|$, for any real number a, b and any function $g(x)$. You may need to recursively use the above hints.

2.2.3 TD as a method to minimize Bellman Error

We have seen that small Bellman Error leads to small prediction error: if we can find a good θ^* such that c is tiny, then we can guarantee that $f(s;\theta^*)$ is close to $V^\pi(s)$. Hence, our goal is to minimize Bellman Error. TD can be understood as doing just this.

Bellman Equality is nice in the sense that it tells us that we can estimate $V^\pi(s)$ by *bootstrapping*. Namely, we can use the current function approximator $f(s;\theta)$ to generate estimate $\tilde{V}^\pi$ for $V^\pi(s)$ as:

$$\tilde{V}^\pi(s) = r(s,a) + \gamma f(s';\theta) \tag{23}$$

where $a \sim \pi(\cdot|s)$ and $s' \sim P(\cdot|s,a)$. We significantly reduce the variance as $\tilde{V}^\pi(s)$ only uses one-step reward $r(s,a)$, and then bottoms up by $f(s';\theta)$. Why

can we hope $\tilde{V}^\pi(s)$ is a reasonable estimate of $V^\pi(s)$? First, $r(s,a)$ is a real reward we get from the system by executing π for *one step* from s. It is unbiased and it has low variance as it is a one-step reward. On the other hand, though $f(\cdot;\theta)$ is just an approximator, whatever prediction error the approximator $f(s';\theta)$ has on estimating $V^\pi(s')$ at the next state s', it is discounted by $\gamma \in (0,1)$. So, if $f(\cdot;\theta)$ is already a good approximator of $V^\pi(\cdot)$, then by *bootstrapping*, we can get an even better approximator $\tilde{V}^\pi(\cdot)$.

Now we can explain how TD works. We initialize our function approximator $f(\cdot;\theta_0)$ with some parameter θ_0. At every iteration n, we use the current function approximator $f(\cdot;\theta_n)$ to perform bootstrap and form a loss function:[2]

$$\ell_n(\theta) = \mathbb{E}_{s\sim v}\left[\Big(f(s;\theta) - \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta_n)]\Big)^2\right] \tag{24}$$

Note that regression target used in Eq. 24 leverages the current function approximator $f(s';\theta_n)$. Given $\ell_n(\theta)$, we are going to use Online Gradient Descent to compute θ_{n+1}.

Question (5 points) Compute the gradient $\nabla_\theta \ell_n(\theta)$ of $\ell_n(\theta)$ with respect to θ. Note that θ only appears in the first $f(s;\theta)$ in ℓ_n, while the second f in ℓ_n uses θ_n, which is the parameter computed from the last iteration.

Now we want to form an unbiased empirical estimate of $\nabla_\theta \ell_n(\theta)$ only using $(s, a, r(s,a), s')$, where $s \sim v$, $a \sim \pi(\cdot|s,a)$, $s' \sim P(\cdot|s,a)$.[3]

Question (5 points) Write down the empirical estimate of $\nabla_\theta \ell_n(\theta)$ using $(s, a, r(s,a), s')$, and show why this is unbiased.

With these formulations, TD algorithm can be understood as online gradient descent:

$$\theta_{n+1} = \theta_n - \mu \nabla_\theta \ell_n(\theta)|_{\theta=\theta_n} \tag{25}$$

You may wonder what the distribution v is and how we can sample from v if the system cannot be set to any state s. In practice, we could simulate v using a replay buffer and uniformly sample from the replay buffer. We refer you to the original experience replay paper from 1992 [2].

2.3 TA comments

Policy Gradients and TD learning are two building blocks of most of modern deep RL approaches. For instance, DDPG can be understood as a combination of policy gradients and TD, where the distribution v used in ℓ_n is replaced by a replay buffer that aggregates all the samples (s, a, r, s') we have encountered

[2] Note that we replace the absolute error in Eq. 19 by squared error in ℓ_n. This is because squared error is smooth while the absolute error is not; in fact, it is non differentiable at some point. A smooth loss is easier to optimize.

[3] In practice, we will use a mini-batch of samples $\{(s, a, r(s,a), s')\}$ to form an unbiased estimate of $\nabla_\theta \ell_n(\theta)$ to reduce the variance.

so far. DDPG uses TD to keep tracking V^π, and then use V^π as the baseline in the policy gradient format. We can easily rename the whole section to "Deep RL" by simply assuming we are using deep neural networks for our function approximators $\pi(a|s;\theta)$ and $f(s;\theta)$!

Last but not least, it turns out that in a lot of cases in robotics, we do not need to go deep. Check out this interesting paper [3] from NeurIPS '17.

3 Programming: Reinforcement Learning(50 Points)

In the previous section, we have went over the basic theory of policy gradient method and temporal different leanring. In this section, we will look at a discrete and finite Grid World task and implement a few basic RL methods to solve . Chapters 4, 6 and 7 from [4] will be helpful.

3.1 Grid World

Here, we will briefly describe the world and the code provided. You will be working with an 8x8 Grid World, fashioned from the 8x8 Frozen Lake in the Open AI Gym environments. It has been stripped down to the basics so you do not need to bother installing Gym and its dependencies. The world looks as follows:

S F F F F F F F
F F F F F F F F
F F F H F F F F
F F F F F H F F
F F F H F F F F
F H H F F F H F
F H F F H F H F
F F F H F F F G

Here, S represents the starting node where the agent begins, F represents a free node on which the agent can safely traverse over, H represents a hole which incurs a -1 reward when the agent reaches it and G represents the goal which gives a reward of 1 on reaching it. G is a terminal state, meaning that once you reach G, the episode is over.

There are four actions possible from every non-terminal state: left, down, right and up indexed as 0, 1, 2 and 3 respectively. Taking an action has a 0.5 probability of success of going in the intended direction. But there is a probability of 0.25 to go to the left of your intended action and 0.25 probability of going to the right. For example, if the intended action is right, the agent would go right with probability 0.5, up with probability 0.25 and down with probability 0.25.

The important data members of the `GridWorld` class are as follows:

```
nS      % Number of states
nA      % Number of actions
P       % Transitions given state s, action a:
        % Dict of dicts of lists, where P[s][a] =
        % [(probability, nextstate, reward, done), ...]
```

For your convenience, the `GridWorld` class has a `step` function which samples the next state and reward given an action, as well a `generateTransitionMatrices` function which generates an $nS \times nA \times nS$ matrix T where $T[s][a]$ is a probability distribution over states when transitioning from state s using action a. The `reset` function sets the initial state of the environment. Take a look at these

functions in the file `environment.py`. You can also take a look at `gridworld.py` to understand how the world is built.

3.2 Reinforcement Learning

Here, you will be implementing Value Iteration (and possibly Policy Iteration) in order to learn a policy for Grid World. You will also implement one-step Temporal Difference Learning (and possibly n-step Temporal Difference Learning) to calculate the value function for a given policy. The file to look at is `rl.py`

3.2.1 Value Iteration

Code (10 points) Implement `value_iteration` and `policy_from_value_function`. What is the final value function? What is the final policy? Provide a visual representation of each, like a grid similar to the one in the description of Grid World. How many iterations does it take to converge?

3.2.2 Policy Iteration

Code (15 points) Implement `policy_iteration`.
What is the final value function? What is the final policy? Provide a visual representation of each, like a grid similar to the one in the description of Grid World. How many iterations does it take to converge?

The final policy and value function should be very similar to that from Value Iteration. Ideally, they would be the exact same but given a finite number of iterations and a finite tolerance, they might not exactly match up.

3.2.3 TD(0) Prediction

For the purpose of this assignment we will only focus on calculating the value function $V^\pi(s)$ corresponding to a given policy $\pi(\cdot|s)$. This can be formulated as an online gradient descent problem (cf. Eq. 25) where the following update is made

$$\hat{V}^\pi(s) \leftarrow \hat{V}^\pi(s) + \alpha \left[r(s, \arg\max_a \pi(a|s)) + \gamma \hat{V}^\pi(s') - \hat{V}^\pi(s) \right]. \tag{26}$$

Here $\alpha \in (0, 1]$ is the learning rate or step size.

Code (10 points) Implement `td_zero`. Use the policy calculated in Section 3.2.1. Use a learning rate of $\alpha = 0.05$.
What is the final value function? Provide a visual representation like before. Is it different than the value function in Section 3.2.1? Why/why not?

3.2.4 n-step TD Prediction

TD(0) is also known as *one-step TD* since it only considers the one-step return $r(s, a) + \gamma \hat{V}^\pi(s')$ for the update equation. n-step TD instead updates the value

of a state by utilizing a trajectory of states, actions, and rewards based on the following equation,

$$\hat{V}^{\pi}(s_t) = \hat{V}^{\pi}(s_t) + \alpha \left[G_t^{(n)} - \hat{V}^{\pi}(s_t) \right] \tag{27}$$

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n \hat{V}^{\pi}(s_{t+n}) \tag{28}$$

Code (15 points) Implement `n_step_td`. Use the policy calculated in Section 3.2.1.
What is the final value function? Provide a visual representation like before. Compare it against the value functions in Sections 3.2.1 and 3.2.3. Do you get the same behaviour as TD(0) with $n = 1$? Why/why not? What n did you use for the results here?

4 What to Submit

Your submission should consist of:

1. a zip file named `<AndrewId>.zip` consisting of a folder `code` containing all the code and data (if any) files you were asked to write and generate. It must include the file `rl.py` that you are required to edit as part of this assignment, along with the `environment.py` and `gridworld.py` files you are not supposed to edit. Submit this to `Homework 4 - Programming` on Gradescope.

2. a `pdf` named `<AndrewId>.pdf` with the answers to the theory questions and the results, explanations and images asked for in the coding questions. **It is compulsory to type-set your answers; images of handwritten documents will not be accepted**. Submit this to `Homework 4 - Theory` on Gradescope. For easier grading, please use the Gradescope feature for tagging the pages and position for each question. 5 style points will be deducted from this homework if this is not followed.

It is mandatory to include in the write-up the names of all people you have discussed this assignment with.

References

[1] Greensmith, Evan, Bartlett, Peter L., and Baxter, Jonathan. "Variance reduction techniques for gradient estimates in reinforcement learning." Journal of Machine Learning Research 5.Nov (2004): 1471-1530.

[2] Lin, Long-H. "Self-improving reactive agents based on reinforcement learning, planning and teaching." Machine learning 8.3/4 (1992): 69-97.

[3] Rajeswaran, Aravind, et al. "Towards Generalization and Simplicity in Continuous Control." arXiv preprint arXiv:1703.02660 (2017).

[4] Sutton, Richard S., and Barto, Andrew G.. "Reinforcement Learning: An Introduction." MIT Press, Cambridge, MA (2017). *Available at:* http://incompleteideas.net/book/the-book-2nd.html.