Statistical Techniques in Robotics 16-831 - The Robotics Institute

Project 4:
Reinforcement Learning and Inverse Reinforcement Learning

Deadline: November 26, 2017 11:59 PM (submit via Canvas)

1 Introduction

In this project, we are going to look Reinforcement Learning and Inverse Reinforcement Learning. 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 and Policy Iteration, as well as the Linear Program formulation for IRL in a finite and discrete world.

Detailed instructions on what to submit are given in Section 4. As usual, remember to start **early** as this class has no late days! Also, try not to leave things for the Thanksgiving break.

2 Reinforcement Learning (50 points)

In this problem, we are going to (1) study Policy Gradient and (2) Temporal Difference 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 $\tau = \{s_1, a_1, s_2, a_2, ..., s_{H-1}, a_{H-1}, s_H, a_H\}$, 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

2.1.1 Trajectory Likelihood

For a policy π, given any trajectory $\tau = \{s_1, a_2, ..., s_H, a_H\}$ and a policy π , we can compute the likelihood $P(\tau; \pi)$ of generating τ under π.

Question (5 points) Write down $P(\tau; \pi)$ using ρ, π, and the transition dynamics P.
Hint: What is the likelihood of getting the first state s_1? Now conditioned on s_1, what is the likelihood of getting action a_1? Now conditioned on s_1 and a_1, what is the likelihood of getting to state s_2?

2.1.2 Vanilla Policy Gradient

With the trajectory likelihood expression, we can now write down the objective function as follows:

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

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

To make the gradient computation easier, we will work on a parameterized policy $\pi(a|s; \theta)$ (otherwise we have to deal with functionals and functional gradients). You can think about $\pi(a|s; \theta)$ as a logistic regressor or a neural network with parameter θ. Let us denote $P(\tau; \pi)$ as $P(\tau; \theta)$, and re-write the objective function as $J(\theta) = \int_{\tau} P(\tau; \theta) R(\tau) d\tau$.

Question (10 points) Prove that the policy gradient $\nabla_\theta J(\theta)$ is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t|s_t; \theta)) R(\tau) \right] \tag{2}$$

Hint: Verify that $\nabla_\theta P(\tau; \theta) = P(\tau; \theta) \nabla_\theta \ln(P(\tau; \theta))$ using chain rule. Now use the likelihood expression for $P(\tau; \theta)$ derived from the previous problem.

2.1.3 Variance Reduction

Now, we have the form for the policy gradient. In practice, however, we cannot compute the expectation in Eq. 2. 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 independently trajectory $\tau_i = \{s_1^i, a_1^i, ..., s_H^i, a_H^i\}, i = 1, ..., K$. We would do this by generating τ_1 first, and then reseting 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:

$$\nabla_\theta J(\theta) = \frac{1}{K} \sum_{i=1}^{K} \left[\sum_{t=1}^{H} \nabla_\theta \ln(\pi(a_t^i|s_t^i;\theta)) R(\tau_i) \right] \tag{3}$$

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:

$$Var(R(\tau)) = \sum_{t=1}^{H} Var(r(s_t, a_t)) + \sum_{i \neq j} Cov(r(s_i, a_i), r(s_j, a_j)) \tag{4}$$

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.

Let us switch back again to the real policy gradient shown in Eq. 2 and study two common tricks to reduce the variance.

2.1.4 Variance Reduction Trick 1

Note that there is **no question** for this section. The first trick is to re-write Eq. 2 as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(\sum_{i=t}^{H} r(s_i, a_t) \right) \right) \right] \tag{5}$$

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. 5 is equal to Eq. 2. 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. 5 and Eq. 2 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 $t-1$.

2.1.5 Variance Reduction Trick 2

We can further reduce the variance by adding a *baseline* to Eq. 5. 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:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(\sum_{i=t}^{H} r(s_i, a_t) - V(s_t) \right) \right) \right] \quad (6)$$

We claim that Eq. 6 is equal to Eq. 2. We can prove it by showing the following is true:

$$\mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(V(s_t) \right) \right) \right] = 0 \quad (7)$$

Question (10 points) Prove Eq. 7. Here is a partial proof, and your job is to fill in the rest.

$$\begin{aligned} &\mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=1}^{H} \left(\nabla_\theta \ln(\pi(a_t|s_t;\theta)) \left(V(s_t) \right) \right) \right] \\ &= \sum_{t=1}^{H} \left[\mathbb{E}_{s_t \sim P_t(s;\theta)} \left[\mathbb{E}_{a_t \sim \pi(a|s_t;\theta)} \nabla_\theta \ln(\pi(a_t|s_t;\theta)) V(s_t) \right] \right] \\ &= ... \end{aligned} \quad (8)$$

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:

$$Var(X - Y) = Var(X) + Var(Y) - 2Cov(X, Y). \quad (9)$$

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 $\sum_{i=t}^{H} r(s_t, a_t)$.

2.1.6 Putting Everything Together

Now we have shown we can plug in any baseline function into Eq. 6, 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. In summary, we can

write down the empirical estimate of the policy gradient using K trajectories generated by executing π on the MDP:

$$\nabla_\theta J(\theta) = \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{t=1}^{H}\nabla_\theta \ln(\pi(a_t^k|s_t^k;\theta))\left(\sum_{i=t}^{H} r(s_i^k,a_i^k) - V(s_i^k)\right)\right] \tag{10}$$

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 Evaluation in the RL literature. There are many methods for policy evaluation, 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 such that we minimizes the prediction error:

$$\mathbb{E}_{s\sim v}\Big[|f(s;\theta) - V^\pi(s)|\Big] \tag{11}$$

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. 11 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:

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

[1]But 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 Evaluation is such an example.

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,

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^{\pi}(s')] \tag{12}$$

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:

$$BE(s;\theta) = \Big| f(s;\theta) - \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta)] \Big| \tag{13}$$

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. 12. Hence, intuitively minimizing $BE(\theta)$ with respect to θ seems like a good idea. However, in practice, due to the modelling 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 (10 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:

$$\big| f(s;\theta^*) - V^{\pi}(s) \big| \leq \frac{1}{1-\gamma} c, \forall s \in \mathcal{S} \tag{14}$$

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

$$\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} V^{\pi}(s')] \tag{15}$$

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

$$\begin{aligned} f(s;\theta^*) =& f(s;\theta^*) - \big(\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta^*)]\big) \\ &+ \big(\mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta^*)]\big) \end{aligned} \tag{16}$$

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:

$$\tilde{V}^\pi(s) = r(s,a) + \gamma f(s';\theta) \tag{17}$$

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]

$$\ell_n(\theta) = \mathbb{E}_{s\sim v}\left[\Big(f(s;\theta) - \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)} f(s';\theta_n)]\Big)^2\right] \tag{18}$$

Note that regression target used in Eq. 18 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:

$$\theta_{n+1} = \theta_n - \mu \nabla_\theta \ell_n(\theta)|_{\theta=\theta_n} \tag{19}$$

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]Note that we replace the absolute error in Eq. 13 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.

2.3 Summary

Policy Gradient 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 gradient 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 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 this year's NIPS!

3 Programming: RL and IRL (50 Points)

For this question, you will look at the discrete and finite Grid World and implement a few basic RL and IRL approaches. The code skeleton can be found on canvas.

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. 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. 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

Bonus: 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.

3.3 Inverse Reinforcement Learning using LP

Now that we have an optimal policy, let us try to work backwards! Given this policy, can we recover a reasonable reward function which represents our Grid World? For this, will look at implementing the Linear Program formulation of IRL that we saw in class.

First, here is a quick recap of the optimization problem:

$$\hat{R} = \arg\max_{R} \left(\sum_{s} \min_{a} \left\{ (P_{a^*}(s) - P_a(s))(I - \gamma P_{a^*})^{-1} R \right\} \right) - \lambda ||R||_1$$

$$\text{s.t.} \quad (P_{a^*} - P_a)(I - \gamma P_{a^*})^{-1} R \geq 0 \ \ \forall \ \ a \neq a^*$$
$$|R(s)| \leq R_{max} \ \ \forall \ \ s \in S$$

Here, R is a vector of rewards for each state, P_a is an $nS \times nS$ matrix representing the transition matrix of probabilities given action a and $P_a(s)$ is an $nS \times 1$ row-vector representing the probability of transitioning from state s using action a. γ is the discount factor, λ is the coefficient of the $L1$ regularizer and R_{max} is the maximum allowed reward for any given state.

Our first job is to convert this to the standard form for a linear program. The Python package `cvxopt` is nice for solving linear programs, so let us try to put our problem into a form they can handle. `cvxopt`[4] expects linear programs in the following form:

$$\min_x \ \ c^T x$$
$$\text{s.t.} \quad Gx \leq h$$
$$Ax = b$$

Let us convert the original form into the one above. We will begin with the objective. There are two parts to it.

3.3.1 $L1$ norm

Let us do the easy part first.

Question (7 points) Convert the $L1$ norm in the objective into the standard form. In other words, convert this into a linear program:

$$\min_x \ \ ||x||_1$$
$$\text{s.t.} \quad Gx \leq h$$
$$Ax = b$$

Hint: Write out the formula for the L1 norm. You will need to introduce an extra variable here to get around the absolute values. What constraints would you need to force your introduced variable to be equivalent to the $||x||_1$? Use the definition of absolute value to create cases.

3.3.2 Separating the best action from the others

Now, for the slightly harder part:

$$\max_R \left(\sum_s \min_a \left\{ (P_{a^*}(s) - P_a(s))(I - \gamma P_{a^*})^{-1} R \right\} \right)$$

Question (8 points) Convert this objective into standard form. Remember to change the max to a min.

[4] http://cvxopt.org/

Hint: Notice that the sum over states allows us to naturally introduce a dot product if we had a variable for each of the states. It would just be the dot product of a vector of these variables with the 1-vector. But now we have to deal with the min inside the sum. We can just have to introduce constraints for each state, action pair s, a using our new variables!

3.3.3 Dealing with the constraints

Question (5 points) Convert the constraints of the original formulation into the standard form.
Hint: We can use the same trick for the constraints involving R_{max} as we did for the $L1$ norm.

3.3.4 Putting it all together

Question (5 points) Finally, write out the final formulation in standard form for the linear program. You should have two sets of introduced variables, one for each term in the objective. Make sure you have a minimization problem!

3.4 Implementing IRL

Code (15 points) Using the standard form, implement the linear program formulation for IRL. You should fill in the function `irl_lp` in the file `irl.py`. Some of the scaffolding has been provided for you with `cvxopt`, which you would need to install.
What is the reward function found? Provide a visual representation of this. Is this a reasonable approximation of the original reward function?

Run Value Iteration (or Policy Iteration) again using this new reward function. There should be some commented code mentioning how to do this. What is the new policy? Again, provide a visual representation of this. How similar is this with the original policy?

4 What to Submit

Submit this homework on Canvas. Your submission should consist of a single zip file named `<AndrewId>.zip`. This zip file should contain:

- a folder `code` containing all the code and data (if any) files you were asked to write and generate.
- a `pdf` named `writeup.pdf` with the answers to the theory questions and the results, explanations and images asked for in the coding questions. Read the write up carefully before your submit. You will lose points if you do not include everything. Feel free to add extra images and figures if you think they are useful for better explaining your answers.

References

[1] Greensmith, Evan, Peter L. Bartlett, and Jonathan Baxter. "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).