Classic and Risk-Sensitive Reinforcement Learning

Tanner Fiez
Department of Electrical Engineering
University of Washington
fiezt@uw.edu

1 Introduction

Reinforcement learning (RL) is a subfield of machine learning that focuses on learning from interactions. The core idea of RL is that an agent learns through the consequences of actions instead of being explicitly taught how to act. Actions are chosen based on past experience and new choices, which can be thought of as learning through trial and error. Inherently this forces a tradeoff between exploitation and exploration that must be solved. This style of learning has often been traced to how humans learn. With each choice we make, feedback from the environment—positive or negative—provides information on how to make future choices. In computer science this abstraction is often ignored and the focus is on computational approaches to learning from interactions.

The distinguishing factors of an RL problem are a closed–loop (actions influence later feedback), indirect feedback (the agent is not explicitly told what to do), and feedback playing out over an extended period of time (feedack from an action may come well after an action is taken). The most fundamental method to model such characteristics is through a Markov Decision Process (MDP). The MDP formulation is designed to include sensing, actions, and a goal in a simple form. An algorithm that solves an MDP is considered to be an RL algorithm. We provide a description of an MDP in Section 2.

We will begin by examining methods for solving MDPs with a focus on the distinct approaches of model based algorithms and model free algorithms in Sections 3 and 4 respectively. In this work we will only consider finite MDPs, i.e. problems with discrete state and action spaces. Note that many problems with continuous state and action spaces can be discretized so that the methods we will discuss are still applicable. The first major contribution of this work is the implementation of many classic RL algorithms in a flexible, object-oriented framework. In support of the algorithms, we develop a grid-world environment that allows for unique problem specifications, rapid testing and comparison of algorithms, and visualization of results. Moreover, we have made our library compatible with OpenAI Gym[1] [1]. OpenAI Gym is a recently developed python toolkit containing a wide variety of RL environments for evaluation purposes. This was a significant step in developing benchmarks for RL because the problems do not arise as naturally as supervised learning problems. Despite this advancement, there is still no universally used python package containing RL algorithms, providing motivation for this portion of the work as we seek to begin to create our own scikit-learn like package to make RL more accessible and ubiquitous.

Following our work with classic RL, we delve deeper into an interesting and promising new line of work called risk-sensitive RL. RL algorithms have historically modeled agents as expected utility maximizers. This modeling paradigm thus considers agents as rational decision makers. A rational decision maker can be described as risk-neutral. However, extensive work in behavior psychology, cognitive science, and economics has shown that humans are inherently irrational decision makers acting according to both a reference point and an internal set of risk preferences. This phenomenon has revealed that humans distort event probabilities and value losses and gains asymmetrically. Specifically, low probability events are overestimated and high probability events are underestimated, and losses are weighed more heavily than gains (see Appendix Section A for a concrete example). In

[1] https://gym.openai.com/envs

[2] an RL framework to model risk-sensitive decision making was developed leveraging behavioral models of human decision making. The results show that many of the convergence properties and optimality conditions from classic RL still apply. We discuss the methods and implications of the paper, our implementation, and the tests we run in Section 5.

Our final contribution, detailed in Section 6, is to apply the RL methods we discuss to the New York Taxi dataset[2] [3]. In this problem we formulate an MDP for taxi drivers and pre-process the data—which includes trip times, distances, fares, and pick-up and drop-off locations—accordingly to create the environment. The goal in this problem is to find the optimal policy for a driver, i.e. to find where a driver should go to look for a new ride following dropping off passengers to maximize their earning rate. Because we have access to the data, we can find the empirical policy of a driver and compare this to the optimal policy. Furthermore, we can apply a parameter sweep for the value functions in risk-sensitive RL to find the set of parameters which produces a policy that most closely aligns with the empirical policy of a driver, allowing us to characterize the risk-preferences of a driver.

2 Markov Decision Process

The key requirement of the state and environment in an MDP is that they obey the Markov property. The Markov property refers to the memoryless property of a stochastic process. In plain language, the Markov property says that given the present, the future is independent of the past. In the most general case the dynamics of a process are defined by the joint probability distribution

$$\mathcal{P}(S_{t+1} = s', R_{t+1} = r' | S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t). \tag{1}$$

In the case where the Markov property is satisfied, the dynamics can equivalently be represented by the following:

$$p(s', r' | s, a) \triangleq \mathcal{P}(S_{t+1} = s', R_{t+1} = r' | S_t = s, A_t = a). \tag{2}$$

The Markov property is fundamental in reinforcement learning because the dynamics of one transition allow for prediction of the next state and the expected reward given only the current state and action. In RL problems, even when the state does not obey the Markov property, it is often still thought of as at least approximating it [4].

Given the dynamics specified by the Markov property in (2) all quantities of interest with respect to the environment can be computed. Specifically, we can determine the state-transition probabilities and the expected rewards of state-action-state triples. The state-transition probabilities are obtained by marginalizing out the rewards.

$$p(s' | s, a) \triangleq \mathcal{P}(S_{t+1} = s' | S_t = s, A_t = a) = \sum_{r' \in \mathcal{R}} p(s', r' | s, a). \tag{3}$$

The expected rewards are obtained by using the definition of expectation.

$$r(s, a, s') \triangleq \mathbb{E}[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r' \in \mathcal{R}} r' p(s', r' | s, a)}{p(s' | s, a)}. \tag{4}$$

We can now define an MDP using the above quantities. An MDP is a tuple given as follows:

$$\text{MDP} = (\mathcal{X}, \mathcal{U}, \mathcal{P}(\cdot|\cdot, \cdot), \mathcal{R}(\cdot, \cdot, \cdot), \gamma). \tag{5}$$

The quantities encompassed by the MDP are defined as:

- $\mathcal{X}$ is a finite set of states.
- $\mathcal{U}$ is a finite set of actions.
- $\mathcal{P}(s'|s, a)$ is a transition kernel giving the probability that taking action a in state s will lead to state s'.
- $\mathcal{R}(s, a, s')$ is a reward kernel giving the reward received from taking action a in state s and ending up in state s'.
- $\gamma \in [0, 1]$ is a discounting factor on the rewards representing the importance of immediate and future rewards.

With the problem framework defined we now focus our attention on defining what it means to solve or approximately solve an MDP and methods to do so computationally.

[2]The New York Taxi dataset is available at `https://publish.illinois.edu/dbwork/open-data/`

3 Model Based Reinforcement Learning

Recall that the goal in RL is to find the optimal policy. This means we want to learn the optimal action to take in each state in the state space. Methods that use a model of the environment given by an MDP to compute an optimal policy are referred to as model based RL algorithms. These methods utilize dynamic programming principles and are sometimes referred to as planning methods because they are offline in the sense that they do not require explicit interaction with the environment.

RL algorithms almost always estimate value functions. Value functions are functions of states or state-action pairs which estimate how much value a state or state-action pair has. The value here means the expected future rewards from a state or state-action pair. Because the expected future rewards depend on future actions, value functions are defined with respect to a policy. We define a policy as a probability mass function from a state to an action. Formally we will denote a policy as $\pi(a|s)$ and when we drop a, s this denotes following π at each state encountered. The value function of state s under a policy π is then given by

$$v_\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s], \tag{6}$$

and similarly the value function of a state-action pair s, a under a policy π is then given by

$$q_\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]. \tag{7}$$

Each of these definitions can be unrolled through recursive relationships to give the following equivalent formulations

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r'} p(s', r'|s, a)[r' + \gamma v_\pi(s')], \tag{8}$$

$$q_\pi(s, a) = \sum_{s',r'} p(s', r'|s, a)[r' + \gamma v_\pi(s')], \tag{9}$$

which are known as the Bellman equations. This formulation is convenient as it is explicitly clear that the sum over s', r' is an expectation over a state-action pair s, a as we noted was the meaning of a value function.

The preceding expressions define value functions for a policy, while the goal of RL is to find the optimal policy. This lends naturally to a set of optimization problems that must be solved.

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{and} \quad q_*(s, a) = \max_\pi q_\pi(s, a). \tag{10}$$

The solutions to these optimization problems give what are known as the Bellman optimality conditions. The derivations follow from the Bellman equations and are provided in Appendix Section B.

$$v_*(s) = \max_a \sum_{s',r'} p(s', r'|s, a)[r' + \gamma v_*(s')], \tag{11}$$

$$q_*(s, a) = \sum_{s',r'} p(s', r'|s, a)[r' + \gamma \max_{a'} q_*(s', a')]. \tag{12}$$

This is a famous result and is detailed in [5]. It is also worth noting that the solution is unique, this argument hinges on formulating the Bellman equation as a fixed point problem and showing that it is a contraction. Additionally, if the dynamics are known the optimality conditions reduce to the problem of solving a system of equations in the dimension of the state space, meaning that any nonlinear system equation solving method can be applied. The optimal policy then naturally follows from this analysis. In the case of the state-value function the optimal policy comes from finding the actions in each state which obtains the maximum of the Bellman optimality condition. Formally this is the following equation

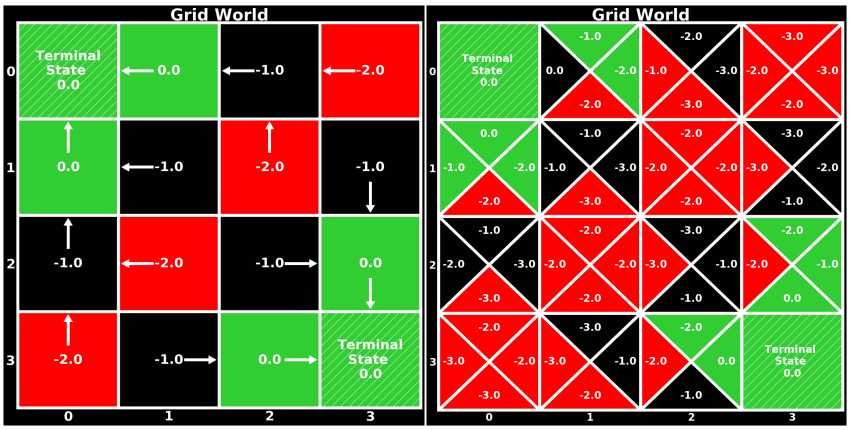
$$\pi^*(s) = \arg\max_a \sum_{s',r'} p(s', r'|s, a)[r' + \gamma v_*(s')]. \tag{13}$$

Similarly, in the case of the state-action value function the optimal policy simply comes from taking the action which maximizes the state-action value function at each state. The optimal policy is thus a greedy policy over the value functions.

The reason such attention was paid to defining the state-value function and the state-action value function and deriving the optimality conditions is that these conditions form the basis for nearly all RL algorithms whether through solving explicitly or through approximation. Model based dynamic programming methods in fact use exactly the Bellman equations and the Bellman optimality conditions. We have implemented the three primary methods: these are policy iteration, value iteration, and q-value iteration. In this section we give an overview and refer the reader to Appendix Section C for the explicit algorithms.

Policy iteration sweeps over the state space and alternates between two steps until convergence. These are policy evaluation, given by (10), and policy improvement (13). Value iteration sweeps over the state space and applies the optimality condition in (11) until convergence. q-value iteration sweeps over the state space and applies the optimality condition in (12) until convergence. Each of these algorithms are optimal and will arrive at the same solution.

We now provide results testing these algorithms and demonstrate that grid-world environment we created in Fig. 1. To prove that the algorithms are implemented correctly we use a very simple MDP. We designate the state space to be indexes of the grid and actions to be the compass directions $\{N, E, S, W\}$. The transition function is deterministic (actions take the agent to the desired state) with the exception that actions that cause the agent to go off the grid result in the agent staying in the same state with probability 1 and an agent remains in a terminal state when reached, all transitions incur a reward of -1 with the exception transitions to and from a terminal state incur a reward of 0, and the discount factor is 1. Thus the problem is to find the shortest path to a terminal state from an initial state which would be the Manhattan Distance.

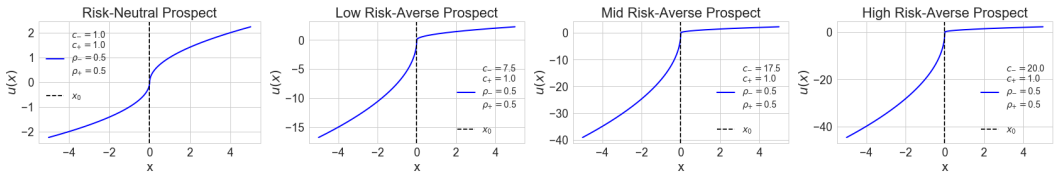


(a) Value Iteration and Policy Iteration. (b) q-value Iteration.

Figure 1: Model Based Algorithms in Grid World

4 Model Free Reinforcement Learning

Sequential decision making problems can often be formulated as an MDP that can be solved via dynamic programming algorithms such as value iteration, policy iteration, and Q-value iteration. These methods however, require that transition probabilities and event outcomes are known a priori. In practice these quantities are typically unknown and thus a policy must be gradually improved as an agent explores an environment. One of the most popular reinforcement learning algorithms to do this is called Q-learning [4]. The Q-learning algorithm iteratively updates the Q-function, a mapping from state-action pairs to value estimates, by taking steps in the direction of an error-like term called the temporal difference.



(a) (b) (c) (d)

Figure 2: Prospect Value Function.

5 Risk-Sensitive Reinforcement Learning

The novel contribution of [2] was to extend this method by applying a transformation to the temporal difference term with a value function obeying certain properties and coming from the class of functions that have been developed to model human behavior, while maintaining convergence guarantees. Through applying the value function to the temporal difference term, a nonlinear transformation is applied not only to the rewards, but also to the transition probabilities. This is significant given that this is precisely what humans have been observed to do when making decisions.

5.1 Algorithm

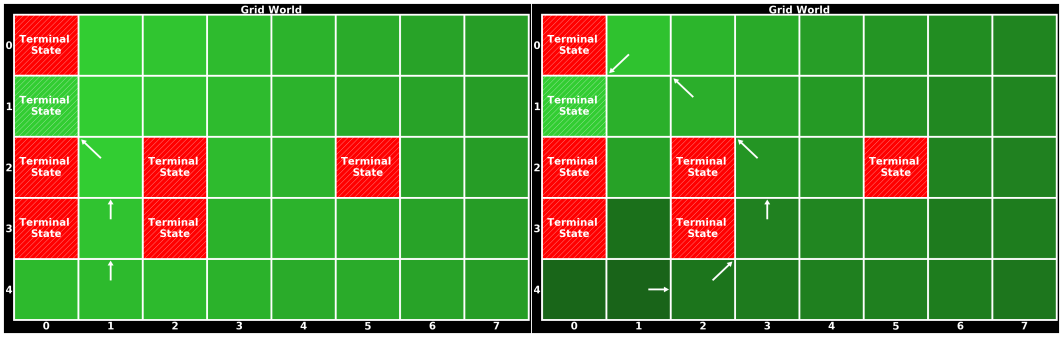
The complete procedure for risk-sensitive Q-learning with a finite horizon and a single episode is in Algorithm 1. For a more detailed description see Section C.1.

Algorithm 1 Risk-Sensitive Q-Learning

procedure RISKSENSITIVEQLEARNING
  Initialize $Q(s,a) = 0$ *and* $N(s,a) = 0\ \forall s, a.$
  for $t = 1$ to T **do**
    $a_t \sim \pi(a|s_t)$
    $N(s_t, a_t) = N(s_t, a_t) + 1$
    $\alpha_t(s_t, a_t) = 1/N(s_t, a_t)$
    $Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left[u(r_t + \gamma \max_a Q_t(s_{t+1}, a_t) - Q_t(s_t, a_t))\right]$

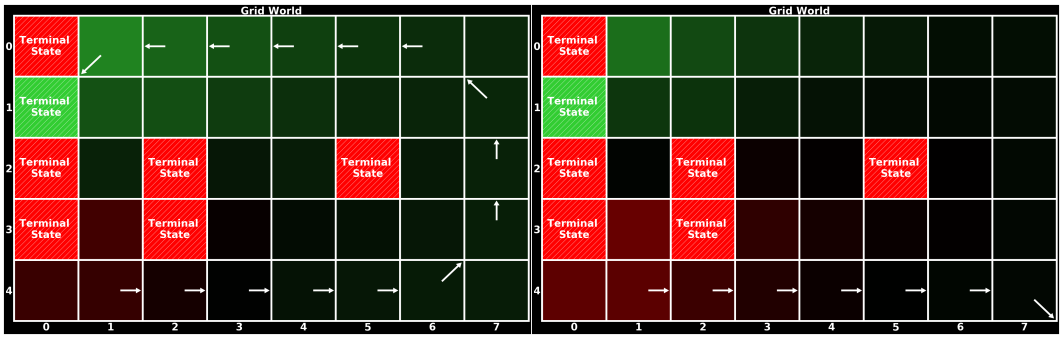
5.2 Simulations

In order to test the implementation of the risk-sensitive Q-learning algorithm we created a synthetic environment called Grid-World. In this environment each state is given by a tile in the grid and the possible actions are the the compass directions (N, E, S, W). The terminal states were set to be the upper left and lower right corners of the grid. Fig. 3 provides an example of the results. We designed the environment, which can be thought of as the underlying MDP, very simply to verify the algorithm was learning correctly. Thus transitions were deterministic, i.e. when an action is chosen the agent goes to the expected next state, and each action incurred reward of -1 except in the terminal states the reward was 0. We then used the prospect value function with all risk-parameters set to 1 which reduces the problem to standard q-learning. Thus the solution that should be learned is to take the Manhattan path to the nearest terminal state, and we indeed see this is the case. More interesting examples can be designed to show how risk-preferences will change the policy that is learned, but we have verified the algorithm is implemented correctly. Detailed results are provided in Section C.3



(a) Q-value function results.

(b) Value function results and policy.



(c) Q-value function results.

(d) Value function results and policy.

Figure 3: Risk Sensitive Q-learning results.

6 New York Taxi Dataset

6.1 Data Description

The New York Taxi Dataset covers taxi operations from 2010–2013, in total there are nearly 700 million recorded trips. The data is stored in separate CSV files for each month. We will analyze a subset of the drivers and over only a few months. Each row in a file contains the information for a trip record. The key information that is contained is the hack license (driver ID), pickup date-time, dropoff date-time, trip time in seconds, trip distance in miles, GPS coordinates at the starting location, GPS coordinates at the ending location, total fare including tip, and the total cost of tolls.

6.2 MDP Formulation for Taxi Drivers

We model a taxi driver as acting according to a finite MDP where an episode corresponds to a single days work. The complete formulation is given in Section 6.4. They salient features of the formulation are:

- Each state is a tuple containing the node the driver is in–we discretized location into a grid using district boundaries for New York City–an indicator of whether the taxi currently is full (just picked up passengers) or empty (just dropped off passengers), and the cumulative reward interval the driver is in–we discretized the reward intervals assuming that how a driver behaves may be a function of the earnings at a point in the workday.
- Actions are moving between nodes in the location grid we created.
- The reward functions use values derived from the data, such as earning rate, the expected time searching for a passenger, and the fare between grid nodes. The transition probabilities use empirical transition probabilities as well as expected earning rates.

6.3 Data Preprocessing

The data required a significant amount of preprocessing to clean the data and obtain the needed quantities for the MDP formulation. This dataset has been used a fair amount for research and there are some well documented errors. In fact, it was estimated by the folks who obtained the data that nearly 10% of trips contained erroneous information. Below are some of the things we did to clean the data:

- The trip time reported is often inaccurate. To compensate for this we calculate the trip time using the difference between the reported pickup and dropoff time.
- We drop the following trip instances: trips that occur in which the GPS coordinates are not in New York, trip times that are extremely short to the point of being infeasible, and trips with extremely low (less than \$.30 a minute) or extremely high (greater than \$20 an hour) earning rates.

These are only a subset of the issues with the data that we cleaned but these are the most significant and well documented from previous work with the data.

There are also a couple of important things we do to create the MDP which we make note of. Drivers work abnormal hours, i.e. the typical driver begins work in the evening and works until the early morning hours. In light of this we shift all transaction times back by 12 hours so that we can observe the full workday within a single date. We estimate the time spent searching for a new trip as the time from dropoff to pickup, but if this time exceeds 20 we assume the driver was taking a break.

6.4 Taxi MDP Model Description

6.4.1 State Space

The state space is

$$\mathcal{X} = \{\mathcal{N} \cup \{x_f\} \times \mathcal{S} \times \mathcal{R}\} \backslash \mathcal{X}_{na}$$

where

- x_f is the terminal state representing the taxi being done for the period,
- $\mathcal{N}$ be the index set for the zones or nodes in the city with N nodes,
- $\mathcal{S} = \{\mathsf{e}, \mathsf{f}\}$ is an indicator of if the taxi is e=empty or f=full, and
- $\mathcal{R}$ is the discretized cumulative fare value space which has the structure

  $$\mathcal{R} = \{\mathcal{R}_1\} \cup \cdots \cup \{\mathcal{R}_m\} \cup \{\mathcal{R}_f\}$$

  where $\mathcal{R}_i = [a_i, b_i]$ with $a_1 < b_1 \leq a_2 < b_2 \cdots \leq a_m < b_m = \bar{r}$ and $\mathcal{R}_f = [\bar{r}, \infty)$ where $\bar{r}$ is some reference point for period earnings (e.g., if the period of investigation is a day, then this is the daily earnings reference point).
- $\mathcal{X}_{na}$ are the states not allowed and is defined by

  $$\mathcal{X}_{na} = \{(x_f, \mathsf{f}, r), r \in \mathcal{R}\} \cup \{(x_f, \mathsf{e}, r), r \notin \mathcal{R}_f\}$$

A state $(i, s, r) \in \mathcal{N} \times \mathcal{S} \times \mathcal{R}$ indicates the taxi is in node i (or terminal state x_f if $i = x_f$), has a empty/full state of s and has current cumulative fare value r. The terminal state is reached when the fare value portion of the state is greater than or equal to $\bar{r}$.

The dimension of the state space is thus,

$$(\dim(\mathcal{N}) + \dim(\{x_f\})) \times \dim(\mathcal{S}) \times \dim(\mathcal{R}) - |\mathcal{X}_{na}|.$$

6.4.2 Action Space

Let $\mathcal{U} = \mathcal{U}_a \cup \{\emptyset\}$ where $\mathcal{U}_a = \{u_{i \mapsto j}, (i, j) \in \mathcal{N} \times \mathcal{N}\}$ be the action space where $u_{i \mapsto j}$ indicates the choice of going from node i to node j and where $\emptyset$ is the null action. The admissible actions are state dependent. In particular, if the state is $x = (i, \mathsf{e}, r)$ for any $i \in \mathcal{N}$ and any cumulative fare value $r \in \mathcal{R}$, then $\mathcal{U}(i, \mathsf{e}, r) = \{u_{i \mapsto j}, (i, j) \in \mathcal{N} \times \mathcal{N}\}$ and, on the other hand, if $x = (i, \mathsf{f}, r)$ for any $i \in \mathcal{N}$ and any cumulative fare value $r \in \mathcal{R}$, then $\mathcal{U}(i, \mathsf{f}, r) = \{\emptyset\}$ indicating that the taxi is currently full and is taking a ride from node i to node k with probability $P_{\mathsf{dest}}(i, k)$ (i.e. the probability that a fare picked up in node i will want to go to node k).

6.4.3 Transition Kernel

Let $\mathcal{P} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to [0, 1]$ be the transition kernel such that $\mathcal{P}(x_t, u_t, x_{t+1})$ is the probability that state x_t will transition to state x_{t+1} given action u_t. Let's consider the different cases.

- First let's look at the e to e transitions for all other state action pairs:

$$\mathcal{P}((i, \mathsf{e}, r), u, (j, \mathsf{e}, r')) = \begin{cases} 1, & \text{if } r, r' \in \mathcal{R}_f \ \& \ \{i \in \mathcal{N}, j = x_f\} \vee \{i = x_f, j = x_f\} \\ 0, & \text{otherwise} \end{cases}$$

- Now let's look at the f to f transitions for all other state action pairs:

$$\mathcal{P}((i, \mathsf{f}, r), u, (j, \mathsf{f}, r')) = 0$$

- Next we will look at the f to e transitions for all other state action pairs:

$$\mathcal{P}((i, \mathsf{f}, r), u, (j, \mathsf{e}, r')) = \begin{cases} P_{\mathsf{dest}}(i, j) P(E[F(i, j)] + r \geq a_l) & \text{if } r' \in \mathcal{R}_l, r' \geq r \ \& \ i, j \in \mathcal{N} \\ 0, & \text{otherwise} \end{cases}$$

  where a_l is the lower bound on the interval $\mathcal{R}_l = [a_l, b_l)$

- Finally we look at the e to f transitions (these are all ones where the choices of action dictates the transition probability)

$$\mathcal{P}((i, \mathsf{e}, r), u, (j, \mathsf{f}, r')) = \begin{cases} 1, & \text{if } u = u_{i \mapsto j} \ \& \ r = r', r \notin \mathcal{R}_f \\ 0, & \text{otherwise} \end{cases}$$

6.4.4 Reward Function

The reward function is a map $R : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \times \to \mathbb{R}$ with $R(x_t, u_t, x_{t+1})$ a random variable such that

- The reward for f to e is

$$R((i, \mathsf{f}, r), u, (j, \mathsf{e}, r')) = \begin{cases} F(i, j), & \text{if } i, j \in \mathcal{N}, u = \emptyset, r' \geq r \\ 0, & \text{otherwise} \end{cases}$$

  where $F(i, j)$ is a random variable representing the fare from i to j.

- The reward for e to f is

$$R((i, \mathsf{f}, r), u, (j, \mathsf{e}, r')) = \begin{cases} -t_{\mathsf{seek}}(i, j)/E(i, j)^{-1}, & \text{if } i, j \in \mathcal{N}, u = u_{i \mapsto j}, r = r' \notin \mathcal{R}_f \\ 0, & \text{otherwise} \end{cases}$$

  where $t_{\mathsf{seek}}(i, j)$ is a random variable for the time to travel and find a fare when you go from node i to j under the control action $u_{i \mapsto j}$ and where $E(i, j)$ is a random variable for the earning rate for trips from i to j. In practice, we infer the mean values of these quantities from the data.

6.4.5 Initial State

The intial state for the taxi will be $x_0 = (i, \mathsf{e}, r_0)$ where i is the node or zone which the taxi most frequently starts in and $r_0 \in \mathcal{R}_0 = [0, b_0)$. It always starts in the empty state.

7 Future Work

As the final deadline approaches we will work on towards the following experiments and goals. First and foremost, utilizing the underlying MDP framework that we have designed for taxi drivers we will investigate the divergence between the empirical policy of drivers found in the data and the optimal policy found with a standard q-learning formulation. This will be allow us to determine whether drivers have learned to maximize their expected earnings. We will then explore the risk-sensitive q-learning formulation, and with the value functions we examine, attempt to find parameters which produce a policy that gives low KL divergence with the empirical policy. If we can do this successfully, it will allow us to characterize how drivers weigh decisions and their risk preferences. Time permitting, we will then explore the risk-sensitive inverse reinforcement learning problem in Grid-World and then in the taxi dataset.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[2] Yun Shen, Michael J. Tobia, Tobias Sommer, and Klaus Obermayer. Risk-sensitive reinforcement learning. *CoRR*, abs/1311.2097, 2013.

[3] B Donovan and DB Work. New york city taxi trip data (2010–2013), 2014.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[5] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[6] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[7] Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and uncertainty*, 5(4):297–323, 1992.

APPENDIX

A Risk-Sensitive Decision Making Example

A prevalent example to demonstrate how risk factors into human-decision making including warping of the probability of events as well as losses being weighed more significantly than gains is as follows. When asked to choose between being given \$90 or having a 10% chance of winning \$100 and a 90% chance of winning \$0 most people will choose to take the guaranteed \$90. When this question is framed as a loss however, i.e. to choose between losing \$90 or having a 10% chance of losing \$0 and a 90% chance of losing \$100, most will choose to risk an increased loss for a chance at no loss. A rational, risk-neutral decision maker would be indifferent to the options in both framings of the question since the expected value of each option is the same.

B Bellman Optimality Conditions Derivation

The bellman optimality conditions represent that the value of a state under an optimal policy must be equal to the expected return for the best action from the state. The derivations follow from the Bellman equations.

$$\begin{aligned} v_*(s) &= \max_{a \in A} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi^*}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r'} p(s', r' | s, a)[r' + \gamma v_*(s')], \end{aligned} \tag{14}$$

The bellman optimality equation for q_* is then

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r'} p(s', r' | s, a)[r' + \gamma \max_{a'} q_*(s', a')]. \end{aligned} \tag{15}$$

C Model Based RL Algorithms

Algorithm 2 Value Iteration

Input: p - the probability distribution for the MDP, r - the reward distribution for the MDP, γ - the discount factor.

Output: Learned value function $v \approx v^*$ and learned policy $\pi \approx \pi^*$.

procedure VALUEITERATION(π, γ)
  Initialize: $v(s) = 0,\ \forall\ s \in S^+$
  while True **do**
    $\delta \leftarrow 0$
    for $s \in S$ **do**
      $v_{temp} \leftarrow v(s)$
      $v(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a)[r(s, a, s') + \gamma v(s')]$
      $\delta \leftarrow \max(\delta, |v_{temp} - v(s)|)$
    if $\delta < \epsilon$ **then**
      break
  $\pi \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a)[r(s, a, s') + \gamma v(s')],\ \forall\ s$

C.1 Risk-Sensitive Q-Learning Description

Q-learning is an off-policy method, meaning the policy being learned is not the policy that is being sampled from when choosing actions to interact with the environment. The policy that is being followed must be proper, meaning that all states are visited infinitely often. Two such proper policies are the ϵ-greedy policy and the Boltzmann policy. Each of these methods trades off exploration and exploitation. Given the greedy action $a^* = \arg\max_a Q(s,a)$ the ϵ-greedy policy is given by

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & a = a^* \\ \frac{\epsilon}{|A|} & a \neq a^* \end{cases} \tag{16}$$

where $\epsilon \in [0,1]$ controls the greediness. The Boltzmann policy is given by

$$\pi(a|s) = \frac{e^{\tau Q(s,a)}}{\sum_a e^{\tau Q(s,a)}} \tag{17}$$

where τ is a temperature parameter $\in [0,\infty)$ that controls the greediness. It is common to use a decay rate on the parameters controlling the greediness to make the algorithm more greedy as the environment is further explored. To guarantee convergence of the algorithm the Robbins and Monro conditions [6] on the learning rate must hold:

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty, \ \forall\, s,a. \tag{18}$$

One such method of choosing the iterates that is common is to let the learning rate be $\alpha_t(s,a) = 1/N_t(s,a)$ where $N_t(s,a)$ is the number of times action a has been taken from state s. The Q-learning update applying the value function to the temporal difference is given as follows:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha_t(s_t,a_t)\left[u(r_t + \gamma \max_a Q_t(s_{t+1},a_t) - Q_t(s_t,a_t))\right]. \tag{19}$$

See Section C.2 for value functions we consider as the nonlinear transformation u.

C.2 Value Functions

We explore several functions that capture risk-sensitive decision making including a prospect theory value function [7] and a logarithm-based value function [**?**]. The prospect value function is given by:

$$u(y) = \begin{cases} c_+(y)^{\rho^+} & y > 0 \\ -c_-(-y)^{\rho^+} & y \leq 0 \end{cases} \tag{20}$$

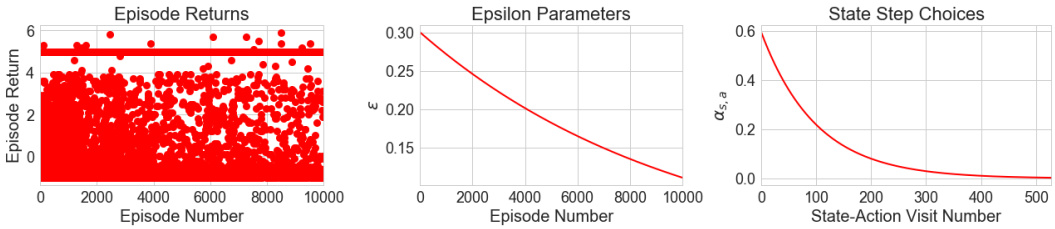
and the logarithm based value function is given by:

$$u(y) = \begin{cases} c_+ \log(1+\rho^+ y) & y > 0 \\ -c_- \log(1-\rho^+ y) & y \leq 0 \end{cases} \tag{21}$$

where we are setting the reference point to be 0. The parameters $(c_+, c_-, \rho+, \rho_-)$ control the degree of risk-sensitivity and loss aversion. Typically human decision makers have $0 < \rho_+, \rho_- < 1$. This leads to leads to risk-averse preferences in gains and risk-seeking preferences in losses. In terms of the shape of the function these preferences correspond to concavity in gains and convexity in losses.

C.3 Further Simulation Results

Further simulation results are given in Fig. 4. In particular we show the reward at each episode which we see see quickly goes towards being optimal, the value of ϵ from using the ϵ-greedy with decay policy, and learning rate using decay for a state.



(a) Episode Returns. (b) Epsilon choices. (c) Learning rate choices.

Figure 4: Detailed Simulation Results.

C.4 Further Figures

D Algorithms

D.1 Model-Based

- Iterative Policy Evaluation
- Policy Iteration
- Value Iteration
- Q-Value Iteration

D.2 Model-Free

- One-Step Temporal Differences
- Sarsa (On Policy Temporal Difference Learning)
- Q-Learning (Off Policy Temporal Difference Learning)

D.2.1 Risk-Sensitive Reinforcement Learning

- Expected Utility Q-Learning (Nonlinear Mapping of Rewards with Value Function)
- Risk-Sensitive Q-Learning (Nonlinear Mapping of Temporal Differences with Value Function)

D.2.2 Value Functions

- Prospect Theory Value Function
- Logarithmic Value Function
- Entropic Value Function