Reinforcement Learning

Introduction

Reinforcement learning takes the perspective that a learning agent exists within an environment for which appropriate actions must be learned in order to maximize the agent's expected cumulative reward. In many cases the agent is placed in the environment and is allowed to explore different **action policies** in hope of learning an optimal policy; i.e. on that maximizes expected cumulative reward.

Reinforcement learning has beein studied for over half a century, with applications to control and guidance systems, economic/trade policy formation, robotics, and game playing. Recently the company Deep Mind combined reinforcement learning with deep neural networks to develop the computer program AlphaGo, the first computer program to defeat a professional Go player on a full-sized board, and without any handicap. What makes this accomplishment unique from past computer game-playing excellence is that AlphaGo makes moves based on pattern reocognition as opposed to searching through the vast space of possible move sequences in a manner similar to how a computer might play a game of chess. The latter strategy is infeasible for the Go game, since a typical Go game position offers several more moves to be played.

Definitions

An **agent** is any entity that can perceive its environment through one or more **sensors** (e.g. cameras, infrared sensors, ports for reading data streams, etc.), and act within the environment through **actuators** (e.g. motors, voltage contollers, ports for sending data, etc.). The current action performed by the agent depends on both the estimated current state of the environment, and on the current state of the agent. For example, if the agent is a road vehicle v, then the environmental state will include the road conditions and the locations and speeds of other vehicles on the road, while the state of v will include the lane it resides in, its speed, acceleration, and the position of its steering wheel. Thus, the state of the environment, along with the state of the agent, together form a state vector that we refer to as the **system state**. Set S denotes the set all possible system states. Also, let A denote the set of actions that are available to the agent while in any given state. Then we may view an agent as moving from one system state to the next, where the next system state depends on both the current

state $s \in S$, and the agent's action $a \in A$ while in s. For a real-time **continuous system** the time that elapses between transitioning from the current state to the next state depends on the physical capabilities of the senors, actuators, and computer hardware needed to perceive, decide, and act. On the other hand, for a **discrete system**, a state change occurs as the result of some event, such as a single move in a two-player game. A **terminal state** is any state for which the agent is either unable or unwilling to act further. For example, if a vehicle runs off the road and into a ditch, then it has reached a terminal state. On a more positive note, the vehicle reaching its desired destination and parking itself in a garage also results in a terminal state, called a **goal state**, since the state results in the goal being reached.

To simplify matters, we also assume that the agent is fully aware of the current system state s. However, the agent may not be able to completely predict the next state s' that results from an action a. For this reason we assume the existence of a probability distribution $P(s'|s, a)$ that gives the probability of transitioning to state s' on condition that the current state is s, and action a is taken.

We assume the agent exists within the environment for the purpose of accomplishing some goal. For example, the goal of a vehicle is to safely transport passengers from one location to another within an amount of time consistent with posted speed laws. The goal of a game-playing agent is to win the game it is playing. The goal of a conversational agent might be to provide useful information to the speaker. With this in mind, we assume the existence of a real-valued **utility function** $U : S \to R$ that assigns each state $s \in S$ to a degree of desirability $U(s)$ that reflects the degree to which being in state s supports the agent's ultimate goal. Usually $U(s)$ is not completely known (or entirely unknown) to the agent, and so the agent must possess an algorithm for computing/learning $U(s)$. However, once the agent has accquired a sutiable model for estimating the utility of any state, then it may direct its actions towards reaching high utility states. This can be done by following the principle of **maximum expected utility (MEU)**, where the agent, while in state s, chooses an action a for which the quantity

$$\sum_{s'} P(s'|s, a)U(s')$$

is maximized.

In a perfect world the agent has complete knowledge of $U(s)$, and thus would choose its actions based on the MEU principle. However, because $U(s)$ is usually not fully known, there is a degree of uncertainty with respect to what action should be performed. This uncertainty is captured by a **policy function** $\pi(s)$ that assigns each state to a probability distribution of actions. Here $\pi(s)(a)$ is interpreted as the degree to which performing action a while in state s is conducive to helping the agent reach a goal state. We let π^* denote the optimal policy for which $\pi^*(s)$ is a distribution that is entirely concentrated at the action that maximizes the expected utility.

Part of the rationale for defining $\pi(s)$ as a probability distribution over actions, is due to the possibility (or likelihood) that an agent may have never before been in state s, but may have experience with other states that are "similar" to s, where similarity can be measured by a learning model, such as an SVM or neural network. This is precisely how a human agent operates in her environment. A human agent will never experience the same internal/environmental state more than once in her lifetime, yet, for all practical purposes, her senors may indicate that the current state is almost identical to

one experienced the previous day. Moreover, if her actions yesterday led to a positive outcome, then chances are it will once again represent an appropriate way to behave. We see then that policies can be learned through experience, and that π may be viewed as a classifier, where the set of classes is the set A of actions. Moreover, many multiclass classifiers return a distribution of actions, rather than a single action. For example, if the class size is n, then it is common for a neural-network classifier to have n output neurons that are collectively normalized to a probability distribution over the n different choices.

Q-Learning

Q-learning is an approach to reinforcement learning for which the agent learns a function $q : S \times A \to R$, where $q(s, a)$ represents the expected utility of performing action a in response to being in state s. The agent may then adopt the policy π for which

$$\pi(s) = \underset{a}{\text{argmax}} q(s, a),$$

i.e., choose the action that yields the highest expected utility. One advantage of Q-learning is that the agent only requires knowledge of how to act in each state, and be able to compute the next state that results from an action.

The most common approach to learning q is to explore the environment by starting in some initial state, and performing a sequence of actions until a terminal state has been reached. Each such sequence will be referred to as a **learning experience**. Of course, the agent requires an action policy before it can acquire a learning experience. To obtain an initial policy, rather than thinking of $q(s, a)$ as a single (utility) value, we may think of $q(s, a)$ as representing a probability distribution over possible utility values. These distributions (one for each state-action pair) are then updated over time as the agent has more learning experiences. For example, suppose we know that the utility of any state s lies between -1 and $+1$. Then one could initialize $q(s, a)$ as the uniform distribution over $[-1, 1]$, meaning that any utility value between -1 and 1 is equally likely. Now, letting $Q(s, a)$ denote a random sample using distribution $q(s, a)$, the agent may then explore its enviroment using policy

$$\pi(s) = \underset{a}{\text{argmax}} Q(s, a).$$

Notice that, initially, $Q(s, a)$ will have high entropy, and so the actions of the agent will seem more random. However, once q is learned, then $q(s, a)$ will be concentrated about the expected utility of performing action a while in state s, and the actions of the agent should become more predictable and purposeful towards maximizing utility.

Now, each learning experience may be represented as a sequence $\gamma = (s_0, a_0, v_0), \ldots, (s_n, a_n, v_n)$ of state-action-assessment-value triples, where s_0 is an initial state, s_n a terminal state, $a_n = v_n = \emptyset$ are `null` values (since s_n is terminal), and assessment value v_i, $i = 0, \ldots, n-1$ may represent either a **reward** $(v_i > 0)$, **punishment** $(v_i < 0)$, or **indifference** $(v_i = 0)$ for the agent performing action a_i, while in state s_i. The assessment of an action can be made immediately after the action is performed, or it can be made retrospectively based on the ultimate outcome of the learning experience (i.e. was the goal reached?).

For many tasks, such as game playing, assessment values are usually assigned in retrospect. This leads to what is referred to as the **credit assignment problem** in which each action must be assessed based on the final outcome of the learning experience. The most common solution is to assign each action an assement value equal to the utility (e.g. $+1$ or -1) of reaching or not reaching the goal. Or if the agent is expected to reach the goal via shorter action sequenes, then the utility value may be divided by the length of the sequence.

Of course, for Q-learning to be effective, there must be a balance of reward and punishment actions over time. For this reason it is often necessary to modify the environment so that reward actions are easier to achieve with a primitive action policy. In the case of a vehicle, this could mean reducing the length of its trip, and allowing for less traffic and fewer obstacles. In the case of a two-player game, the agent could play against itself (or earlier versions of itself) to ensure an opponent with similar skill level.

Representing the Policy

For problems involving a large number of states the most economical approach to representing an action-value policy q is to use a learning model, such as an svm, random forest, or neural network. For example, the AlphaGo program makes use of a deep convolutional neural network for learning the policy function needed for making moves in a game of Go. The only requirement for the learning model of choice is that it should be amenable to being updated after each learning experience, or after a batch of learning experiences. In other words, it should allow for **online learning**, which is essential for Q-learning, and reinformcement learning in general.

The remainder of this section focuses on the case when the number of states and actions are relatively small (e..g. can fit into computer memory). In such cases online learning may be facilitated with the use of statistics.

For simplicity, assume state s and action a are fixed, and let $X = Q(s, a)$ be a random variable whose domain is the set of possible reinforcement values that can be assigned to the state-action pair (s, a). The idea is that each learning experience γ for which $(s, a, v)\gamma$, for some reinforcement value v, allows us to sample X. And the more samples we have of X, the more accurate estimate we can make of $E[X]$, the expected utility that results in performing a while in state s. Indeed, let $x_1, \ldots, x_n$ denote the samples of X collected thus far. These samples may then be used to compute **posterior statistics** for X; i.e. statistics that are obtained *a posteriori*, meaning after having observed the data. These statistics are defined as follows.

Sample Mean used to approximate $E[X]$

$$\hat{\lambda}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Sample Variance used to approximate $\mathrm{Var}(X)$

$$\hat{\sigma}_n^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - n\hat{\lambda}_n^2\right]$$

Standard Error If Z is defined as $Z = \frac{1}{n}(x_1 + \cdots + x_n)$, then the standard error represents an approximation of the standard deviation of Z. In other words, since $E[Z] = E[X]$, if we were to repeat the IMC experiment several times, then it would represent the average observed deviation from $E[X]$.

$$\text{s.e.} = \frac{\hat{\sigma}_n}{\sqrt{n}}$$

To simplify notation, when the value of n is understood or not important, we write $\hat{\lambda}$ instead of $\hat{\lambda}_n$. Similarly, $\hat{\sigma}^2$ denotes $\hat{\sigma}_n^2$. The justification of the posterior statistics representing good approximations of the prior statistics for increasingly large values of n is justified by the following theorem(s) from probability.

Theorem 2. Asymptotic Properties of Posterior Statistics. Assume that $x_1, \ldots, x_n$ are drawn independently from X, and are distributed according to $\omega(x)$. Moreover, suppose that $E[X^4] < \infty$. Define $\lambda = E[X]$ and $\sigma^2 = \mathrm{Var}(X)$. Then

Law of Large Numbers For every $\epsilon > 0$ and $\delta > 0$, n can be chosen sufficiently large so that

$$P(|\hat{\lambda} - \lambda| > \epsilon) < \delta.$$

Convergence of Sample Variance For every $\epsilon > 0$ and $\delta > 0$, n can be chosen sufficiently large so that

$$P(|\hat{\sigma}^2 - \sigma^2| > \epsilon) < \delta.$$

Central Limit Theorem As n increases, the CDF of

$$Z = \frac{\hat{\lambda} - \lambda}{\sqrt{\sigma^2/n}}$$

converges to the CDF $\Phi(z)$ of the standard normal distribution.

Central Limit Theorem with Sample Variance Replacing Prior Variance As n increases, the CDF of

$$Z = \frac{\hat{\lambda} - \lambda}{\sqrt{\hat{\sigma}^2/n}}$$

converges to the CDF $\Phi(z)$ of the standard normal distribution.

The Central Limit Theorem may be interpreted in the following way. Let n be fixed, and suppose that for a long period of T days we plan to each day obtain n independent samples of X, and compute

a sample mean for X. Let $\hat{\lambda}(i)$ denote the mean we obtain on day i, $i = 1, \ldots, T$. Then if we make a histogram for the T numbers (say using $\log T$ bins) $\hat{\lambda}(1), \ldots, \hat{\lambda}(T)$, the histogram would approximate a normal distribution with mean $\lambda = E[X]$, and stadard deviation $\sigma_X/\sqrt{n}$. Moreover, for increasing values of n the approximation would not only appear more accurate, but the mass would become more highly concentrated about the true mean λ.

Based on the above discussion, it seems reasonable to approximate $q(s, a)$ by sampling from a normal distribution having mean $\hat{\lambda}$ and standard deviation $\hat{\sigma}/\sqrt{n}$, where n is sufficiently large (say 10 or more). For smaller values of n, one could continue to use a uniform distribution. In this way the amount of variance in our estimate of $q(s, a)$ is inversely related to the amount of experience the agent has with performing a when in state s.

Although the above statistical approach is only feasible for spaces where $|S| \times |A|$ is relatively small (say in the tens of thousands), a similar method may be used in the case of using a regression learning model $\hat{q}(s, a)$, where the output of this model is interpreted as approximating the mean of a normal distribution having variance $\sigma(t)$, where t is a time parameter that increments with each learning experience, and $\sigma(t) \to 0$ as t increases. In other words, $\hat{q}(s, a)$ can display more random behavior when the agent has little experience, but then converges towards a fixed value once the agent has enough experience and is "set in its ways".

Example 1. Consider the 4×3 grid shown in Figure 1, where each cell represents a different environment state. The agent begins in state $(1, 1)$ and must reach goal state $(4, 3)$. The other terminal state is $(4, 2)$ which represents failure. Notice that $(2, 2)$ is not a legal state, and represents a physical barrier. While in any state, the agent has four possible actions: `up`, `down`, `left`, and `right`. Unfortunately, whenever an agent performs action a, there is a 20% chance that it instead moves in direction that makes a right angle with respect to its intended direction. For example, if the agent decides to move `up`, there is a 10% chance that it instead moves `left`, and another 10% chance that it instead moves `right`. We assume that the agent remains in the same cell if its action leads it to hitting a wall or barrier.

In order to simulate learning experiences for the agent, $q(s, a)$ was computed by i) sampling a uniform $U(-1, 1)$ random variable if the agent had performed a while in state s fewer than $n = 10$ times, and ii) sampling a normal distribution with mean $\hat{\lambda}_n$ and standard deviation $\hat{\sigma}_n/n$ in case $n > 10$, where $\hat{\lambda}_n$ is the sample mean of the rewards obtained for each learning experience that used a while in state s. Figure 2 shows the optimal actions that were learned for each state. These actions were learned after only a few hundred experiences.

Learning $U(s)$

Notice that the process of learning a policy through unsupervised and supervised learning does not require knowledge of utility function $U(s)$. However, there are cases where learning $U(s)$ can seem beneficial. One way of learning $U(s)$ is to do so in conjunction with learning $\hat{\pi}$. This is done by considering each sequence $\gamma = (s_1, a_1), \ldots, (s_n, a_n)$ of state-action pairs (where s is either a training sequence, or is produced during unsupervised learning), and each utility value $u(s_n)$, and randomly

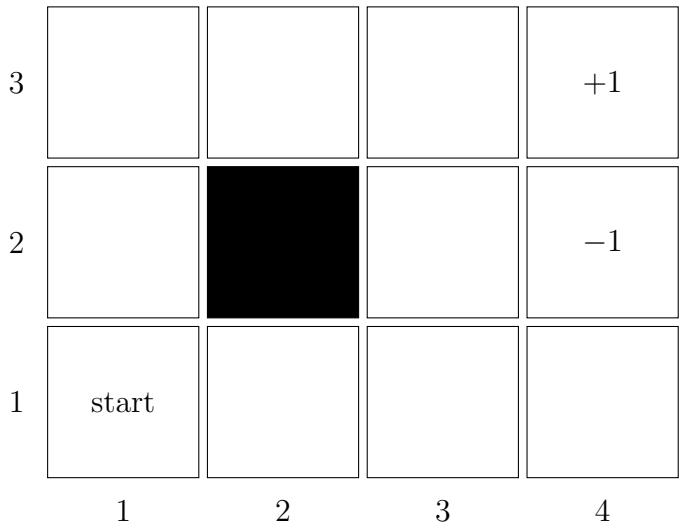

Figure 1: Grid environment for Example 1

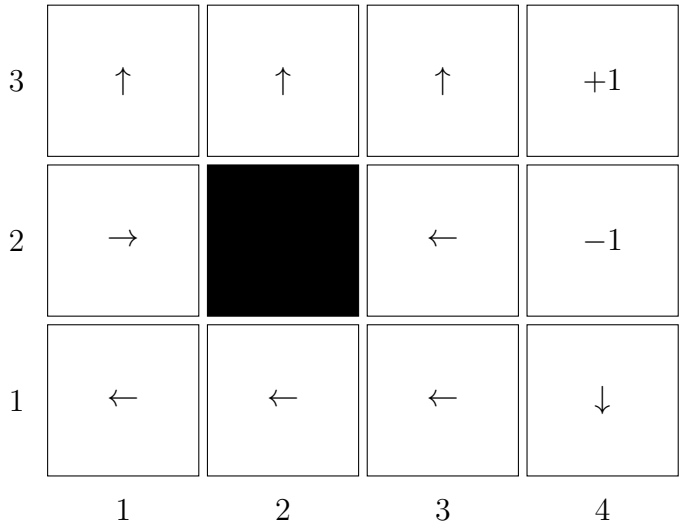

Figure 2: Optimal actions for Example-1 grid

selecting a state $s \in \{s_1, \ldots, s_n\}$ to produce the training pair $(s, u(s_n))$. Note that only one state from γ is selected due to the statistical correlation between states of γ.

Example 2. We can learn the utility function $U(s)$ for the environment from Example 1, by initializing $u(s)$ and $n(s)$ to zero for each state s, and then randomly selecting a state s' from each simulation, and incrementing $n(s')$, and either incrementing or decrementing $u(s')$ by 1 depending on whether or not the simulation resulted in reaching the goal state. Then $U(s)$ can be approximated as $u(s)/n(s)$.

Looking ahead

For some environments it may be possible to spend some time looking ahead to the future before acting on the present. If s_0 is the current state, then one may simulate sequences of action-state pairs $\gamma = (a_1, s_1), \ldots, (a_d, s_d)$ of length d, where the value of d is problem dependent. Given state s_i in the sequence, $i = 0, 1, \ldots, d-1$, then next action $a_i + 1$ may be determined using either learned policy $\hat{\pi}$, or in combination with some other distribution for increased variation. Furthermore, the next state s_{i+1}, may be obtained using the assumed next-state distribution $P(s_{i+1}|s_i, a_{i+1})$. Once the d th state s_d has been reached, utility function U may be applied to s_d, and its value may be recorded for action a_1. Then, letting n_a be the number of simulations for which the first action is a, then associated with each action a is the average utility value

$$\mu_a = \frac{1}{n_a} \sum_{a \in \gamma} U(s_{d,\gamma}),$$

where $s_{d,\gamma}$ denotes the d th state of simulation sequence γ. Then we may choose the next action as

$$a = \underset{a \in A}{\operatorname{argmax}}(\mu_a).$$

Of course there are many different strategies that may be used for looking ahread, depending on the nature of the agent and environment.

Exercises

1. A real-valued function $f(x)$ over real variable x is **convex** iff for any two real numbers x and y, and any real number $\alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y).$$

   Moreover, we say that f is **concave** iff $-f(x)$ is convex. Prove that $f(x) = x^2$ is convex, while $f(x) = \log x$ is concave.

2. Suppose you have utility function $U(x)$, where $x \in \mathcal{R}$ denotes your net worth in US dollars. As a contestant on a game show you have just won \$1,000. The show host now asks if you wish to trade your \$1,000 for a chance to win \$2,500 by playing another game for which you will win with probability 0.5. Assuming you base all your actions on the principle of maximum expected utility do you go for the \$2,500 prize if your utility function is convex? concave? linear? Explain.

3. Suppose you have a monetary utility function of $U(x) = \ln(1 + x)$. Currently your net worth is one million dollars, in the form of a beautiful home. Over the next year the probability of losing your home to a fire is approximately 0.00001. Should this happen, your worth will drop to thirty thousand dollars (the value of the land on which your home sits). If no fire occurs, your home's value is expected to increase by 10% over the coming year. Also, you have an opportunity to buy fire insurance for J dollars. By paying this insurance, should you lose your home to fire, you will be paid \$970,000, the value of your home, minus the land worth. Assuming you base all your actions on the principle of maximum expected utility, determine the value of J for which buying fire insurance has the same expected utility as not buying the insurance.

4. One way of viewing the problem of choosing amongst states for the sake of maximizing utility, is that each state represents a lottery, and we desire the state that yields the highest expected prize. A lottery may be defined using a recursive definition. For the base case, if L is a real number, then L is a lottery with utility $U(L)$. A lottery consisting of a single number is called a **fixed lottery**. For the recursive case, if $p_1, \ldots, p_n$ are real numbers satisfying $p_1 + \cdots + p_n = 1$, and $L_1, \ldots, L_n$, are lotteries, then $L = [p_1, L_1; \ldots; p_n, L_n]$ is a lottery, and the utility of L is defined as

$$U(L) = \sum_{i=1}^{n} p_i U(L_i).$$

   Assuming $U(L) = L$ for all fixed lotteries L, calculate the utility of $L = [0.9, -5; 0.1, A]$, where A is the lottery $A = [0.5, -5; 0.5, 2500]$.

5. This problem is meant to help elicit your personal monetary utility function $U(x)$, where x is in US dollars. The only assumption we'll make about $U(x)$ is that $U(0) = 0$. Which lottery would you prefer: $A = [0.2, 0; 0.8, 4000]$ or $B = [0.0, 0; 1.0, 3000]$? Note: there is no one correct answer. Based on your preference, derive a relationship between $U(4000)$ and $U(3000)$.

6. Repeat the previous problem, but now assume lotteries $A = [0.8, 0; 0.2, 4000]$ and $B = [0.75, 0; 0.25, 3000]$. How does the inequality from this problem compare with the one you derived for the previous one?

7. An urn contains 1/3 red balls and 2/3 balls consisting of a mixture of black and yellow balls. However, the ratio of black to yellow is unknown. A ball is to be randomly selected from the urn. Which choice would you prefer: being paid 100 if the selected ball is red, or being paid 100 if it is black? Explain the reasons for your preference. Note: there is no one correct answer.

8. Repeat the previous problem bu now suppose the choices are being paid 100 if the selected ball is red or yellow, or being paid 100 if it is black or yellow. Assuming f is the fraction of balls that are black, can you find a value for f that makes each of your chocies the better choice? Explain.