Lecture 20
Reinforcement Learning Part 1

Alice Gao

November 2, 2021

Contents

1 Learning Goals

- Describe the setting and the goals of passive reinforcement learning.
- Perform direct utility estimation and describe its pros and cons.
- Describe the steps of the adaptive dynamic programming algorithm for passive reinforcement learning.
- Trace the execution of the temporal difference learning algorithm for passive reinforcement learning.

2 Introduction to Reinforcement Learning

Let's look at the basic setting of a reinforcement learning problem.

Recall that there are three types of problems in machine learning. For supervised learning, we have labels for every example. For unsupervised learning, we have no label for any example. Reinforcement learning is somewhere between the two. In a reinforcement learning problem, we are given some numeric feedback called rewards or punishments, once in a while, and we want to determine what to do at each time step given these numeric feedback,

Let's consider a fully-observable, single-agent reinforcement learning problem. We will model this problem as a Markov decision process. At the beginning, the agent is given the set of states and the set of possible actions. At each time step, the agent observes the state and the reward since the environment is fully observable. After observing the state and reward, the agent carries out an action. The agent's goal is to maximize its total discounted reward over time.

Reinforcement learning is challenging for several reasons. First, the agent receives a reward infrequently. It is often difficult to determine which action or which sequence of actions was responsible for a reward. For example, playing a chess game requires a lot of actions but we only get one reward at the end (win or loss).

Second, an action may have long-term effects on the agent's utility. Carrying out a seemingly bad action at the beginning may allow the agent to receive large rewards later on. Choosing an action is particularly challenging since the agent does not know the effect of the action beforehand.

Third, at any time step, should the agent explore or exploit? If the agent always exploits, it may not discover better actions. If the agent always explores, it never makes use of the learned knowledge to maximize its utility. The agent needs to carefully balance exploration and exploitation.

3 Passive ADP Algorithm

Before I tackle the full reinforcement learning problem, let's consider a simpler setting — a passive learning problem. In this setting, the agent follows a fixed policy pi. Its goal is to learn the expected value of following this policy. That is, learn the utility value V for every state s.

The passive reinforcement learning problem is similar to the policy evaluation step in the policy iteration algorithm. Given a policy, we want to learn the utility values V. However, this problem is more difficult for two reasons. The agent does not know the transition probabilities. Nor does it know the reward function.

Fortunately, we can tackle this problem using the same approach as policy evaluation — solving for the utility values V using the Bellman equations. To do this, we must learn the transition probabilities and the reward function. We can learn these by using the observed transitions and rewards as we navigate the world.

In other words, we will take actions one at a time based on the policy. Each action will allow us to observe the reward and a transition. We will use these observations to update our estimates of the transition probabilities and the reward function.

This algorithm is called the passive adaptive dynamic programming algorithm or the passive ADP algorithm.

ADP is a model-based algorithm because it requires us to learn a model of the world. The model consists of the transition probabilities and the reward function.

Example: The Passive ADP Algorithm

1. Repeat steps 2 to 5.
2. Follow policy π and generate an experience $\langle s, a, s', r' \rangle$.
3. Update reward function: $R(s') \leftarrow r'$
4. Update the transition probability.

$$\begin{aligned} N(s,a) &= N(s,a) + 1 \\ N(s,a,s') &= N(s,a,s') + 1 \\ P(s'|s,a) &= N(s,a,s')/N(s,a) \end{aligned}$$

5. Derive $V^\pi(s)$ by using the Bellman equations.

$$V(s) = R(s) + \gamma \sum_{s'} P(s'|s,\pi(s))V(s')$$

Let's take a look at the passive ADP algorithm.

There is a loop from step 2 to step 5. The agent will go through this loop whenever it takes an action and generates a new experience.

Step 2: Generating an experience.

Suppose that the agent is in state s right now. The agent takes an action a based on the policy pi. This causes the agent to transition to state s' and receive a reward r'. The agent will use this experience s, a, s', r', to update its estimates of V(s), the utility value of each state.

Step 3: Updating the reward function.

Entering state s' gives us a rewrad of r'. If we haven't observed this reward before, we will record it for our reward function.

Step 4: Updating the transition probability estimates.

The experience contains one state transition: taking action a in state s causes the agent to transition to state s'. We can use this to update our transition probability

To calculate the transition probabilities, we keep track of two counts. N(s,a) records the number of times the agent takes action a in state s. N(s,a,s') records the number of times the agent takes action a in state s and lands in state s'. The probability of s' given s and a is simply N(s,a,s') divided by N(s,a).

We will increment the two counts and calculate the updated transition probability.

Step 5: Update our estimates of the V(s) values.

After updating the reward function and the transition probability, we are ready to estimate the utility values.

Looking at the Bellman equations, R is known, discount factor is given, and P is known. The V values are the variables. We can solve for the V values exactly since all the Bellman equations are linear. You can use your favourite linear programming technique to solve them. Alternatively, we can solve for the V values iteratively using value iteration or policy iteration.

Problem:

s_{11}	$+1$
s_{21}	-1

- $\pi(s_{11}) = down, \pi(s_{21}) = right$
- $\gamma = 0.9$
- $R(s_{11}) = -0.04, R(s_{21}) = -0.04, R(s_{12}) = 1, R(s_{22}) = -1$
- $N(s, a) = 5, \forall s, a.$

- $N(s, a, s') = 3$ for the intended direction.
- $N(s, a, s') = 1$ for a direction to the left or right of the intended direction.

For the counts, assume that we have encountered each state-action pair 5 times. Out of the 5 experiences, we travelled in the intended direction 3 times. We traveled to the left of the intended direction 1 time and travelled to the right of the intended direction 1 time. Our current transition probability estimates are: the probability of traveling in the intended direction is 0.6, and the probability of traveling to the left or the right of the intended direction is 0.2.

The current state is s_{11}. Suppose that the agent tries to move down and ended up reaching state s_{21}. Calculate $V(s_{11})$ and $V(s_{21})$.

Solution:

The execution steps are as follows.

1. No need to update the reward function.
2. Update the counts.

   $N(s_{11}, down) = 6$ and $N(s_{11}, down, s_{21}) = 4$.
3. Solve the Bellman equations.

$$\begin{aligned} V(s_{11}) &= -0.04 + 0.9(0.667V(s_{21}) + 0.167(1) + 0.167V(s_{11})) \\ V(s_{21}) &= -0.04 + 0.9(0.6(-1) + 0.2V(s_{11}) + 0.2V(s_{21})) \end{aligned}$$

   The solutions are:

$$V(s_{11}) = -0.4378, V(s_{21}) = -0.8034$$

Let's go through the loop once. The current state is s11. the policy says that the agent should go down. Suppose that the agent reached s21 and received a reward of -0.04. The experience is ¡s11, down, s21, -0.04¿. Let's do some updates.

We have observed the reward before. No need to update the reward function.

For the counts, N(s11, down) should be 5 + 1 = 6. N(s11, down, s21) should now be 3 + 1 = 4. The new transition probability is P(s21 — s11, down) is 0.667.

With the updated reward function and the transiton probabilities, let's calculate the utility values. There are two non-goal states, so we can write down the Bellman equations and solve them exactly.

4 Active ADP Algorithm

Previously, I discussed the passive ADP agent. The agent follows a fixed policy and learns the expected value of following the policy.

However, in the real world, the agent is not limited to using a fixed policy. They have the complete freedom to follow any policy. The question is, what should the agent do? What action should the agent take at each step?

What is the effect of taking an action? In our grid world, taking an action serves two purposes.

1. An action can provide rewards. This is the short-term benefit to the agent.
2. An action can help us gather more data to learn a better model. This is a potential long-term benefit.

Based on these two purposes, there are two useful things that the agent can do.

This first option is exploit. This option may be what you think of by default. If the agent has learned a model consisting of the transition probabilities and the utility values, the agent should take the optimal action based on this model.

An alternative option is explore. The agent may want to take an action that is different from the optimal one. What is a good idea to take a sub-optimal action? For example, this action may take the agent to a state that they haven't been before. The agent may discover actions that are better than the ones they have found so far.

If an agent choose to exploit all the time, we call this agent a greedy agent.

Experiments have shown that being greedy all the time is a terrible idea. The greedy agent seldom converges to the optimal policy and sometimes converges to horrible policies.

The main reason for this is that at any point, we have limited observations of the environment. Our learned model based on these limited observations is never the same as the true environment. Therefore, behaving optimally based on our learned model is often very different from the optimal behaviour in the true environment.

Because of this, the best strategy for the agent is to perform exploration and exploitation, trying to maintain a balance between the two. This has been studied in depth in a subfield of statistical decision theory that deals with multi-armed bandit problems.

4.1 Trade-Off Exploration and Exploitation

What is the best way to maintain the trade-off between exploration and exploitation? Let's look at a few strategies.

Strategy one: We will take a random action (or explore) a epsilon fraction of the time, and take the best action (or exploit) a (1 - epsilon) fraction of the time. Epsilon is a parameter that we need to set. In practice, we may want to start with a relatively large epsilon and decrease it over time. In this way, we explore more at the beginning and decrease the exploration as time goes on.

Strategy two: One problem with strategy one is that, it treats all the sub-optimal actions in the same way. Some sub-optimal actions may be better than other ones. One way to improve strategy one is to select each action with a probability that is proportional to the expected utility of taking the action.

This leads to the idea of softmax selection. We can do this using the Gibbs/Boltzmann distribution.

The probability of taking an action is proportional to Q(s,a), the expected utility of taking action a in state s. T in the formula is a parameter called the temperature. We can use T to adjust the shape of the distribution.

When T is high, the distribution is close to a uniform distribution, and we would be choosing every action with similar probabilities. When T is low, the distribution does a better job of distinguishing the actions with different expected utilities. We are more likely to choose an action that has a higher expected utility. In the limit, when T approaches 0, the distribution approaches a point mass and we would be choosing the best action with probability 1.

Strategy three: Instead of changing the probability of choosing an action, we can also encourage the agent to explore by changing the utility estimates. This is the third idea: using optimistic utility estimates to encourage exploration. We'll look at this strategy in more detail shortly.

4.2 Optimistic Utility Estimates to Encourage Exploration

Previously, I introduced some ideas to encourage exploration. The last idea was to use optimistic utility estimates. Let me describe this strategy in more detail.

Example:

We will learn $V^+(s)$ (the optimistic estimates of $V(s)$).

$$V^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)V^+(s'), N(s,a)\right)$$

$$f(u,n) = \begin{cases} R^+, \text{ if } n < N_e \\ u, \text{otherwise} \end{cases}$$

$f(u,n)$ trade-offs exploitation and exploration.

- R^+ is the optimistic estimate of the best possible reward obtainable in any state.
- If we haven't visited (s,a) at least N_e times, assume its expected value is R^+.
- Otherwise, use the current $V^+(s)$ value.

The agent is more likely to try an action that leads to a higher expected utility. To encourage the agent to explore, we can set all the utility estimates to be large in the beginning. These utility estimates are called optimistic utility values. Once the agent has tried a state-action pair a certain number of times, we can revert back to computing the utility estimates based on our observations.

To do this, we have modified the Bellman updates. Let V plus denote the optimistic utility estimates. In the original Bellman update, after the max a, we are calculating the expected utility of taking action a in state s. The expected utility is the sum of the transition probability multiplied by the V value. In this modified Bellman update, we are calculating the expected utility of taking action a using a f function.

This f function is the exploration function. It can help us trade-off exploration and exploitation.

f takes two parameters: the actual expected utility given the observations so far and the number of times we have visited this state-action pair. If we haven't visited a state-action pair at least N_e times, then we assume that the expected utility of taking the action is R plus. R plus is the optimistic estimate of the best possible reward obtainable in any state. In general, R plus is a large value, making the state-action pair very attractive for the agent. Once the agent has tried this state-action pair at least N_e times, f will return the actual expected utility of taking the action a computed using our observations.

In short, the optimistic utility estimates encourage the agent to try the state-action pair at least N_e times.

For this strategy to work, the exploration function f should be increasing in u and decreasing in n.

4.3 Active ADP Algorithm

Example:

1. Initialize $R(s), V^+(s), N(s,a), N(s,a,s')$.

2. Repeat steps 3 to 7 until we have visited each (s,a) at least N_e times and the $V^+(s)$ values converged.

3. Determine the best action a for current state s using $V^+(s)$.

$$a = \arg\max_a f\left(\sum_{s'} P(s'|s,a)V^+(s'), N(s,a)\right),\ f(u,n) = \begin{cases} R^+, \text{ if } n < N_e \\ u, \text{otherwise} \end{cases}$$

4. Take action a and generate an experience $\langle s, a, s', r' \rangle$

5. Update reward function: $R(s') \leftarrow r'$

6. Update the transition probability.

$$N(s,a) = N(s,a) + 1, N(s,a,s') = N(s,a,s') + 1$$
$$P(s'|s,a) = N(s,a,s')/N(s,a)$$

7. Update $V^+(s)$ using the Bellman updates.

$$V^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)V^+(s'), N(s,a)\right)$$

Let's look at the steps of the active ADP algorithm.

Active ADP algorithm combines the passive ADP algorithm and an exploration strategy. For example, let's combine the passive ADP algorithm and the third exploration strategy: making use of the optimistic utility estimates.

We start by initializing some values arbitrarily, the reward function, the optimistic utility estimates, the counts N for estimating the transition probabilities.

Next, we will go inside a loop. We will keep going through the loop until the optimistic utility estimates converge.

Inside the loop, the algorithm alternates between two tasks:

(1) Take the optimal action based on the current utility estimates.

(2) Update the utility estimates by using the new experience.

For steps 3 - 4, we take the current optimistic utility estimates and determine the optimal action for the current state. The agent takes the action and generates an experience.

For steps 5 - 7, given the experience, we are going to update a few things. First, we'll update the reward function, if we haven't observed the reward before. Second, we will update the counts for the state-action pair for calculating a transition probability. Finally, we will update the optimistic utility estimates iteratively using Bellman updates until the utility estimates converge. Note that we are still using the optimistic utility estimates when we are performing value iteration or policy iteration.

In this algorithm, we will check convergence in two places. One, we need to check convergence during value iteration. We will perform value iteration until the utility estimates converge. Two, we need to check convergence during consecutive iterations of the loop. After each iteration, we will check how much the utility values differ from those in the previous iteration and whether they converged.