Lecture 21
Reinforcement Learning, Part 2

Alice Gao

November 2, 2021

Contents

1 Learning Goals

By the end of the lecture, you should be able to

- Trace and implement the passive Q-learning algorithm.
- Trace and implement the active Q-learning algorithm.
- Compare and contrast ADP and Q-learning algorithms.

2 Temporal Difference Learning

2.1 Bellman equations for Q(s,a)

Previously, I introduced the ADP algorithm for reinforcement learning. ADP learns the utility values V(s) by using the Bellman equations. You can see the Bellman equations for the V values on this slide.

Example:

Bellman equations for $V(s)$:

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

There is a related quantity called the Q values. Q(s,a) gives the agent's expected utility of performing action a in state s. V and Q are closely related and we can define them recursively in terms of each other.

Since V and Q are closely related, we can write down the Bellman equations in terms of the Q values as well. Q(s,a) is equal to the immediate reward of entering state s plus the expected utility of performing action a in state s. If we perform action a, with some probability, we will reach the next state s'. In state s', we will perform the optimal action a' based on the Q values. max Q(s', a') gives us the agent's expected utility of performing the best action in state s'.

Example:

Bellman equations for $Q(s,a)$:

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

Since we have Bellman equations for both V and Q, learning the V and Q values are equivalent. If we have an algorithm for learning the V values, we can convert it into an equivalent one learning the Q values.

There are pros and cons of learning the V values or the Q values. For example, one advantage of learning the Q values is that we do not need to learn the transition probabilities. I'll explain this idea in more detail shortly.

2.2 Temporal Difference Error

I am going to introduce two related reinforcement learning algorithms: Q-learning and SARSA (S-A-R-S-A). Both algorithms belong to a class of algorithms called Temporal Difference Learning. Let me give you an example to explain the key idea behind temporal difference learning.

Assume that we have received an experience. Starting from state s1, we have received a reward of r1. We took the action a and reached state s2. Based on this observed transition, how should we update the Q value Q(s1, a)?

Let me start with the Bellman equation for Q(s1, a). It is equal to the immediate reward of entering s1 plus the agent's expected utility of taking action a. Action a may take us to the next state s' and the agent will perform the best action in state s' based on the Q values.

Example:

Bellman Equations:

$$Q(s_1, a) = R(s_1) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$$

Based on the Bellman equation, Q(s1, a) should be calculated by the expression on the RHS. There is one problem with this calculation — we don't have the transition probabilities. To solve this problem, let's make a simplifying assumption. Since this is the only transition we have observed so far, let's assume that this transition always occurs. In other words, let's assume that P(s2 — s1, a) is equal to 1. Given this assumption, we can simplify the RHS expression. Q(s1, a) is equal to the immediate reward of entering s1 plus the discount factor multiplied by the agent's expected utility of taking the best action in state s2. Note that the transition probability disappeared since we assumed that the transition from s1 to s2 occurred for sure.

$$R(s_1) + \gamma \max_{a'} Q(s_2, a')$$

This expression we just wrote down is our prediction of the Q(s1, a) value based on the observed transition. If we take our prediction and subtract the current value of Q(s1, a), this difference is called the temporal difference error. The key idea in temporal difference learning is to update the Q values proportional to the temporal difference error.

Example:

Temporal difference (TD) error:

$$(R(s_1) + \gamma \max_{a'} Q(s_2, a')) - Q(s_1, a)$$

3 Q-Learning

3.1 Q-Learning Updates

Let me introduce the Q-learning update rule. Q-learning is an example of temporal difference learning. The key idea is to update the Q values proportional to the temporal difference error.

Given an experience, a transition from state s to state s' by taking action a, we will update Q(s,a) as follows.

Example:

Given an experience $\langle s, r, a, s' \rangle$, update $Q(s, a)$ as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Bigg(R(s) + \gamma \max_{a'} Q(s', a') - Q(s,a) \Bigg)$$

Take the original Q value and add the temporal difference error. The temporal difference error is equal to the predicted Q value minus the current Q value. The predicted Q value is the immediate reward plus the discounted expected utility of taking the best action in the next state s'.

Alpha is a value between 0 and 1. Since we multiplied the temporal difference error by alpha, the change in the Q value is only a portion of the temporal difference error.

Let's re-arrange the terms and write the update rule in another way. This alternative version might appear more intuitive for you.

Example:

An alternative version of the Q-learning update:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \Bigg(R(s) + \gamma \max_{a'} Q(s', a') \Bigg)$$

In this alternative version, we are changing the Q value to a linear combination of two terms. The first term is the current Q value. The second term is the predicted Q value based on the observed transition. Alpha controls the weights of the terms. If alpha is large, the predicted value has more weight and we are potentially making a large change. If alpha is small, the current value has more weight and we are likely making a small change.

3.2 Passive Q-Learning Algorithm

Let's take a look at the passive version of the Q-learning algorithm. Recall that, in passive reinforcement learning, the agent has a fixed policy and the goal is to learn the expected utility of following the policy. In this case, our goal is to learn the Q value, which is the agent's expected utility of taking action a in state s.

Example:

The Passive Q-Learning Algorithm

1. Repeat steps 2 to 4.
2. Follow policy π and generate an experience $\langle s, r, a, s' \rangle$.
3. Update reward function: $R(s) \leftarrow r$
4. Update $Q(s, a)$ by using the temporal difference update rules:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s,a) \right)$$

The passive Q-learning algorithm is quite similar to the passive ADP algorithm. You might want to compare them side by side. One major difference is that, for Q-learning, we do not need to update the counts to learn the transition probabilities. The other main difference is that, the last step is on updating the Q value using the temporal difference error rather than updating the V value using the Bellman equation.

I have made a special note about alpha. Alpha is called the learning rate. It is similar to the learning rate in the gradient descent algorithm. The magnitude of alpha controls the size of each update.

In the algorithm, I wrote alpha to be a fixed value. In practice, it is better to change alpha as we receive new experiences. Let N(s,a) denote the number of times the agent has taken action a in state s. Roughly speaking, if alpha decreases as N(s,a) increases, then the Q values will converge to the optimal values. One example of such an alpha function is alpha = 10 / (9 + N(s,a)).

3.3 Active Q-Learning Algorithm

Time to look at the active Q-learning algorithm.

Example:

The Active Q-Learning Algorithm

1. Initialize $R(s), Q(s,a), N(s,a), N(s,a,s')$.
2. Repeat steps 3 to 6 until we have visited each (s,a) at least N_e times and the $Q(s,a)$ values converged.
3. Determine the best action a for current state s using $V^+(s)$.

$$a = \arg\max_a f\Big(Q(s,a), N(s,a)\Big),\ f(u,n) = \begin{cases} R^+, & \text{if } n < N_e \\ u, \text{otherwise} \end{cases}$$

4. Take action a and generate an experience $\langle s, r, a, s' \rangle$
5. Update reward function: $R(s) \leftarrow r$
6. Update $Q(s,a)$ using the temporal difference update rules.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$$

Again, the structure looks quite similar to the ADP algorithm. However, I will argue that active Q-learning is much simpler than active ADP for one reason: The learning algorithm does not care about the policy that the agent is following. Let's take a closer look.

Similar to active ADP, we can decompose active Q-learning into two parts. The bottom part is basically a copy of passive Q-learning. Given an experience, we will update the reward function and update the Q value using temporal difference update. However, note that, this update does not reference the current policy in any place. In particular, for the next state s', the learning algorithm assumes that the agent is following the greedy policy. That is, the agent chooses the best action based on the Q values and achieves the expected utility of the max Q(s', a') over the action a'. However, the agent may not be following the greedy policy at all. This learning algorithm works regardless of the policy that the agent is following.

The top part of active Q-learning is to determine the agent's action given the current state. This part does depend on the current policy that the agent is following. This version uses the optimistic Q values to encourage the agent to explore. If the agent hasn't tried a state-action pair at least N_e times, then we assume that the Q value is R plus, which is the maximum possible reward we can obtain in any state. Having R plus as the reward makes the state-action pair super attractive to the agent. Once the agent has tried a state-action pair at least N_e times, we will use the current Q value instead.

Active Q-learning only needs to check convergence in one place. Every time we go through the loop, we will check whether the agent has visited each state-action pair at least N_e times and whether all the Q values have converged. If both are satisfied, we will terminate the algorithm.

4 Properties of Q-Learning

Thinking about Q-learning, the first thing that comes to your mind may be our version of Q-learning learns Q(s,a) instead of V(s). This distinction is not important. It's straightforward to write an equivalent Q-learning algorithm for learning the V values instead.

Second, Q-learning is a model-free algorithm since it does not require us to learn the transition probabilities. In contrast, ADP is a model-based algorithm. Because of this, Q-learning requires much simpler computation than ADP.

Q-learning is not guaranteed to converge to the optimal Q-values. If the agent explores enough, then Q-learning learns an approximation of the optimal Q-values.

How good is this approximation? Can we improve it? We can improve the convergence by adjusting the learning rate alpha. The smaller alpha is, the closer it will converge to the optimal Q values, but the slower it will converge. This makes intuitive sense. If alpha is small, the magnitude of each update is small. We would be adjusting the Q values very slowly and cautiously until they converge.

4.1 ADP v.s. Q-Learning

Let's compare ADP and Q-Learning. They are both reinforcement learning algorithms, but they are different in some significant ways.

Does the algorithm require the agent to learn the transition probabilities?

ADP is model-based and requires the agent to learn the transition probabilities. Q-learning is model-free and does not need to learn the transition probabilities.

How much computation is performed per experience?

Comparing the two, ADP requires more computation per experience. After receiving every experience, ADP tries to maintain the consistency in the utility values by adjusting them using the Bellman equations.

Q-learning performs a simple update based on the observed transition only. It does not try to keep the Q values consistent between neighbouring states. As a result, Q-learning requires less memory and computation time.

How fast does the algorithm learn?

ADP typically converges faster than Q-learning. Q-learning learns slower and shows much higher variability.

In general, a model-based algorithm like ADP is more efficient in terms of experience. They require fewer experiences to learn well.