Learning to Make Decisions With and Without a Model of the World

Session 3 Lecture Notes*

Leo Klenner† Henry Fung† Cory Combs†

1 Introduction

In this section, we will present an algorithm (in its simplest version) called the "Q-Learning Algorithm" to learn the Q values in a six-grid world.

Suppose we have a drone. We want to incorporate an reinforcement learning algorithm in its flight management system (FMS) so that it can automatically plan the shortest flight path to the enemy base for a reconnaissance mission. We divided the operational region of this drone into six sectors (similar to our six grid world problem). In addition, for the sake of illustrating the Q-Learning Algorithm, we make the following simplifying assumptions:

- The location of the enemy base is known and is labelled as the goal state (G on Figure 1).
- We are only dealing with deterministic reward and subsequent states. In other words, if the drone flies to the right from grid A in Figure 1, it will always end up in grid B. We assume that there are no cross-wind that can perturb the drone as it flies to the right and causes it to end up in another state. Similarly, we assume deterministic reward– the drone will always receive 100 if it reaches the goal state, and 0 otherwise.
- In order for the Q-Learning algorithm (in its "vanilla" form) to work, we must assume bounded immediate reward values. In other words, for all state-action pairs, $r(s, a) < c$, where c is a constant.
- The drone chooses actions in such a way that it visits every possible state-action pair infinitely often. Overtime, the drone must execute action a from state s repeatedly with non-zero frequency.

Armed with these assumptions, we will begin by briefly presenting the Q-learning problem again from Session 1. Next, we will present the Q-Learning Algorithm and illustrate how it works with our drone example. Finally, we will examine the conditions in which the Q-Learning algorithm will "converge" to a solution (i.e., find an optimal policy after n training episodes), and strategies that we can deploy to accelerate convergence (i.e., shorten the amount of training episodes that the drone needs to find an optimal policy).

The Q function $Q(s, a)$ is a evaluation function that is used by the drone to evaluate each of its possible actions from each possible state (what we called state-action pair). It is important to make the distinction between actions and "state-action" pairs. This is because the consequence of performing an action (say "up") is different when the drone is in grid B in Figure 2, than when it is in grid A. Thus, we need to evaluate state-action pairs rather than actions.

*Written using XeLaTeX.

†Johns Hopkins University School of Advanced International Studies (SAIS), Washington, D.C.

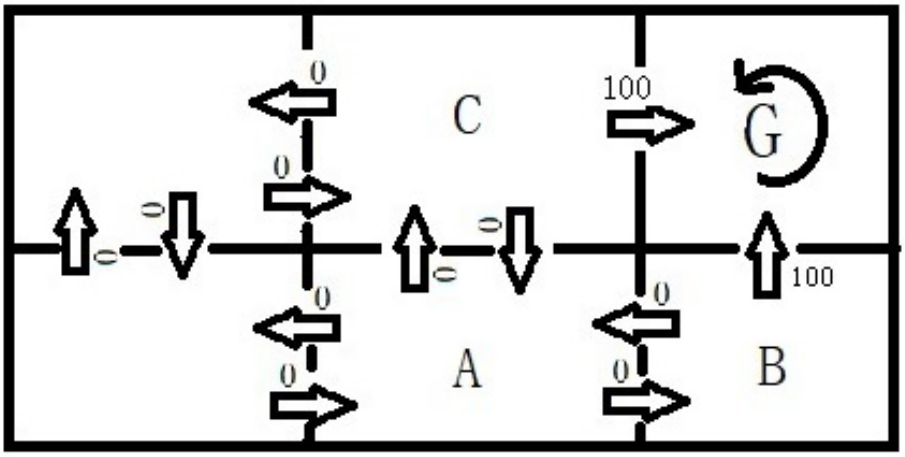

Figure 1: The six-grid world: immediate reward values.

In Session 1, we derived the Bellman equation that assigns a "Q value" to each state-action pair(s, a):

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a') \tag{1}$$

From the above equation, we see that the Q value is the the reward that the drone received immediately upon executing action from state s, plus the value (discounted by *delta*) that it receives by choosing the state-action pair with the highest Q value thereafter.

Let's illustrate with an example. Suppose the drone is currently in grid A in Figure 2. We want to determine the Q value for moving to the right ($a = right$) from grid A ($s = A$) using the Bellman equation. As before, we assume $\delta = 0.9$, and all immediate rewards are 0 except for actions that would lead to the goal state G.

From the Bellman equation, we know that the immediate reward $r(s, a)$ is zero since the drone does not move to the goal state. Now, let's look at the second term $\gamma \max_{a'} Q(\delta(s,a), a')$. We know that after moving right, the subsequent state of the drone is grid B. From grid B, the drone performs the action "up" since it has the highest Q value (100). Thus $a' = up$, $\delta(s, a)$ is grid B, and $\max_{a'} Q(\delta(s,a), a')$ is 100. Putting it all together, the Q value of the drone at grid A, moving to the right is as follows:

$$Q(s = A, a = right) = 0 + 0.9 * 100 = 0.9 \tag{2}$$

As shown in Figure 2, $Q(s = A, a = right)$ is 90 (as indicated on the red arrow).

Initially, the Q values (the values on top of the arrows in Figure 2) are unknown to the drone. The objective of the Q-Learning Algorithm precisely is to estimate the unknown Q values through multiple attempts to reach the goal state from an arbitrary starting grid (what we called training episodes). Let's suppose for the moment all the Q values are successfully estimated by the algorithm (values in Figure 2). How can we use the Q values to determine the optimal policy π^* for the drone? Recall that an optimal policy assigns an action a to the drone at every state s such that it will move to G through the shortest possible path.

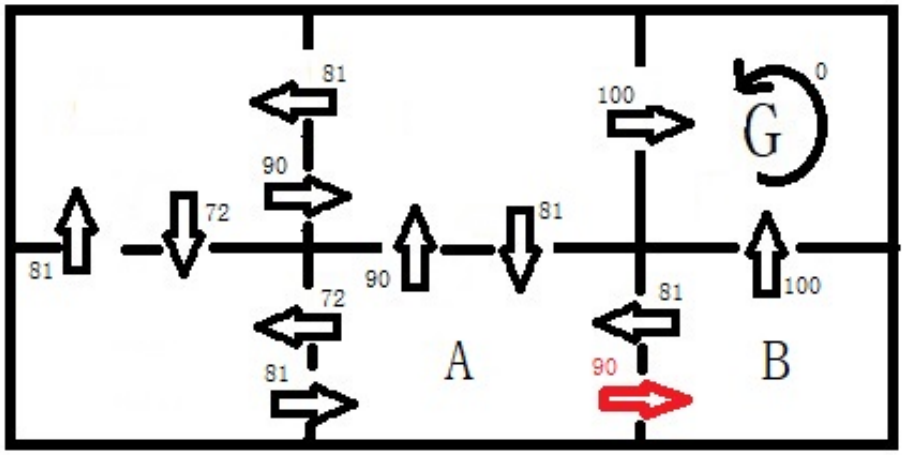

Figure 2: The six-grid world: Q values.

The optimal policy is defined as follows:

$$\pi^*(s) = \underset{a}{\text{argmax}}\, Q(s,a) \tag{3}$$

The optimal policy "dictates" that the drone should always perform action a with the highest Q value. If the drone is in grid A, then, it should choose up (or right) since these actions are associated with the highest Q value. Next, it should choose right (up) to reach the goal state. You can experiment this yourself using Figure 2.

In summary, finding the Q values in Figure 2 corresponds to finding the optimal policy. In the next section, we will present the Q-Learning Algorithm for finding the Q values, and an illustrative example to demonstrate the operation of this algorithm.

2 The Q-Learning Algorithm

The Q-Learning Algorithm is an iterative process (i.e., we "loop" through this algorithm many times until a solution is reached). At every iteration, we have a set of estimates for the Q value that we denote $\hat{Q}$. For our example, the set $\hat{Q}$ contains the estimates for the 12 initially unknown Q values in Figure 2. You can imagine that the drone jots down these 12 estimated values in a table at each iteration, and then updates these values as it make more observations with each action. The values that you see in Figure 2 are the "true" Q values that the drone strives to estimate.

The table of $\hat{Q}$ values can be initially filled with random values (ex: the drone can set all values to 0). The drone then repeatedly observes its current state s, chooses some action a and then observe the resulting immediate reward $r(s,a)$ and subsequent state $s' = \delta(s,a)$. It then updates the table of the 12 $\hat{Q}$ values as follows:

$$\hat{Q}(s,a) = r + \gamma * \underset{a}{\text{argmax}}\, \hat{Q}(s',a') \tag{4}$$

From the above "update rule", we see that the drone updates $\hat{Q}$ of its previous state with the $\hat{Q}$ of its new state. Thus, there is some sort of back propagation that occurs with this algorithm as we will see in our illustrative example.

The steps of the Q-Learning Algorithm is presented below. Using this algorithm, the estimated $\hat{Q}$ will converge to the actual Q, provided that several conditions are met. The issue of convergence will be address later in this document.

The Q-Learning Algorithm:

1. Initiate the $\hat{Q}(s, a)$ values of all state-action pairs to 0.
2. Observe the current state s.
3. Select an action a and execute it.
4. Receive immediate reward r.
5. Observe the new state s'
6. Update the table entry for $\hat{Q}(s, a)$ using $\hat{Q}(s, a) = r + \gamma * \underset{a}{\text{argmax}}\, \hat{Q}(s', a')$
7. Set state s' to s, repeat steps 2 to 6 until $\hat{Q}$ converges to actual Q.

To illustrate the Q-Learning Algorithm, consider the scenario in Figure 3. In the initial state, the drone is in grid D ($s_1 = D$). Note that some of its current estimates $\hat{Q}$ are shown in the left figure in Figure 3. Next, based on its current $\hat{Q}$, the drone chooses to fly right $a = right$ and ends up in grid E ($s_2 = E$). It then applies the training rule (Equation 4) to update the $\hat{Q}$ of the state-action transition that it had just executed (highlighted in red in the right figure) as follows:

$$\hat{Q}(s_1, right) = r + \gamma * \underset{a}{\text{argmax}}\, \hat{Q}(s_2, a') \tag{5}$$

Substituting values to the above Equation, we have:

$$\hat{Q}(s_1, right) = 0 + 0.9 * max(63, 81, 100) = 90 \tag{6}$$

The new $\hat{Q}$ of the state-action transition ($s_1 = D, a = right$) is the sum of the received immediate reward (0), plus the highest $\hat{Q}$ out of all possible state-action pairs from grid E. In this manner, each time the drone flies forward from an old state to a new old, the algorithm propagates $\hat{Q}$ from the new state to the old. The immediate rewards is used to "augment" the propagated values of $\hat{Q}$. For each training episode, the drone begins at some randomly chosen state and is allowed to execute actions until it reaches G. When it does, a new episode starts and the drone is initialized (most likely in a simulation or a Python program) to a new state.

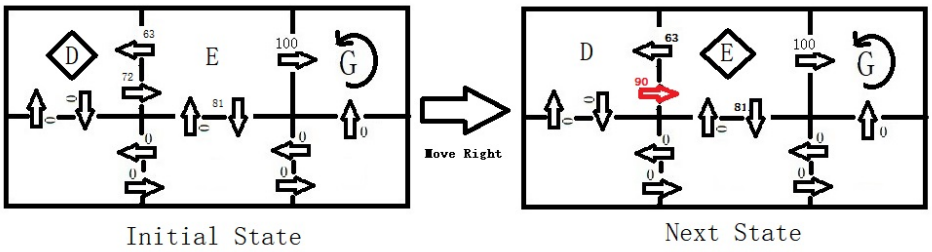

Figure 3: Updating estimates of the Q value.

In this example, all 12 estimates $\hat{Q}$ will be set to 0 initially. The algorithm will not make any updates to the $\hat{Q}$ values until it reaches to the goal state (where it receives a non-zero reward 100). This will

update the state-action pair that leads to the goal state to 100 (ex: s=E, a=right). On the next episode, if the drone passes through grid D (or any other grid that is adjacent to E), the non-zero $\hat{Q}$ in grid E will propagate to a state action pair in grid D (as in our example, $\hat{Q} = 100$ in grid E is used to update $\hat{Q}$=90 in grid D). Assuming that we have sufficient number of training episodes, the information will propagate from non-zero state-action pairs to the entire state-action space available to the drone. In other words, all 12 estimates in the $\hat{Q}$ table will be non-zero and be filled with the true Q values in Figure 2.

3 Conditions for Convergence

The question remains: is there a guarantee that the $\hat{Q}$ values in the Q-Learning Algorithm will converge to the true Q values after running many iterations? The answer is yes, but only under the following conditions:

- (A1) Deterministic MDP, meaning that the immediate rewards and the subsequent states are deterministic. Performing action a at state s will always yield the same reward and subsequent state.
- (A2) Immediate reward values are bounded. For all state-action pairs, $r(s, a) < c$, where c is a constant.
- (A3) The agent visits every possible state-action transition infinitely often. Overtime, the drone must execute action a from state s repeatedly with non-zero frequency.

The Q-Learning Algorithm allows for negative immediate rewards. In addition, any number of state-action pair may produce non-zero immediate rewards. However, one of the more restrictive assumptions above is (A3): the drone must perform every possible action from every possible state with non-zero frequency overtime. Imagine instead of a six-grid world, we have a 10000 grid world. In large (or continuous domains), (A3) might not be satisfied and thus the Q-Learning Algorithm will not estimate $\hat{Q}$ that converges to Q overtime.

4 Strategies for Experimentation

The Q-Learning Algorithm does not specify how the drone chooses its actions. In the illustrative example, the drone chose action a in state s that maximizes $\hat{Q}(s, a)$, and this is certainly a rule that drone can follow to choose its actions. By doing so, the drone is exploiting its current approximation of $\hat{Q}$. However, there are two problems with following this "exploitation strategy", and this harkens back to our discussion of the trade-off between exploitation vs. exploration in Session 1:

- The drone will risk "overcommitting" to actions that are found early on in the training that has high $\hat{Q}$ values, while failing to explore other state-action pairs that might have even higher $\hat{Q}$ values.
- The conditions for convergence (A3) requires the drone to visit each state-action pair infinitely often. This will not occur if the drone always selects state-action transitions with the highest $\hat{Q}(s, a)$.

For these reasons, a better alternative is for the drone to use a probabilistic approach to choose its actions. Actions with higher $\hat{Q}$ will be assigned higher probabilities, but all actions have non-zero probability of being selected. By favouring actions with higher $\hat{Q}$ (i.e., assigning higher probabilities to these actions), the drone will exploit what it has learnt and seek actions that will maximize its rewards, based on its current estimates. In contrary, by assigning less probabilities on actions with high $\hat{Q}$, the drone would then be allowed to explore actions that do not currently have high $\hat{Q}$ values. This exploitation-exploration trade-off can vary with the number of iterations so that the drone will do more exploration during the early stages of training, and focus more on exploitation in the latter stages.