CS3243 **AY20/21 Sem1**

Lecture 9: October 21

Lecturer: MEEL, Kuldeep S. *Scribes: Song Qifeng*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

9.1 Value iteration

9.1.1 Background

In the previous lecture, we have obtained the Bellman equation

$$U(s) = R(s) + \gamma \; max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s') \tag{9.1}$$

Bellman equation serves as a recursive definition of the utility of a given state.

9.1.2 Aim

We want to obtain the optimal policy, therefore we need to first obtain the utilities of all states.

9.1.3 Problem

Bellman equation can be applied to every state, together they form a equation set.

While it is theoretically possible to straightly solve the equation set, its complexity is too huge because the **max** operation is non-linear, thus making the equation set non-linear, too.

9.1.4 Solution: Value iteration

The current problem we meet is much similar to the shortest distance problem we have met before in CS2040S, and a solution to it is Bellman-Ford algorithm. We can get some inspiration from Bellman-Ford and apply it to the current problem.

9.1.4.1 Inspiration from Bellman-Ford algorithm

Recall the Bellman-Ford algorithm we have learnt in CS2040S:

We needed to obtain the shortest path between two points but solving the problem by brute force is too expensive. Alternatively, we initialize all distances and iteratively update them.

9.1.4.2 Apply the idea of Bellman-Ford

We can initialize the utility of every state to 0, then iteratively update them until we meet a certain terminal standard.

Recall that in Bellman-Ford algorithm, the terminal standard is when no update is done throughout an iteration.

For this problem, we set the terminal standard to be when every update during one iteration is within a small threshold.

9.1.4.3 Pseudocode

```
ValueIteration($problem, \epsilon$)
    $U' \leftarrow EmptyDictionary()$
    for each state $s \in states$ do
        $U'[s] \leftarrow 0$
    $\delta \leftarrow \infty$
    while $\delta \geq \epsilon(1-\gamma)/\gamma$ do
        $U \leftarrow Copy(U')$
        $\delta \leftarrow 0$
        for each state $s \in states$ do
            $U'[s] \leftarrow R(s) + \gamma max_{a\in A(s)} \sum_{s'} P(s'|s,a)U(s')$
            $\delta = max(\delta, |U'[s] - U[s]|)$
    return $U'$
```

Note that as the iteration time approaches infinity, $U'[s]$ will converge to $U[s]$.

9.2 Policy iteration

9.2.1 Motivation

In order to get the optimal policy, the previous approach is to first calculate the utility of every state, then come up with a policy. Alternatively, we can also directly look for the optimal policy without calculating the utilities.

9.2.2 Inspiration from searching

Note that a policy is a **mapping** from **state** to **action**.

Unlike utility values which are continuous, policies are discrete, meaning that we can treat every policy as a state and our job is now transformed to search for the global maximum state (policy).

We start from an arbitrarily selected state (policy), then continuously moving to a better neighbor.

9.2.3 Policy iteration

Step 1:

Initialize π with an arbitrarily chosen policy.

Step 2:

Solve the linear equation set and get the utility of every state.

Step 3:

Use the following method to update the policy,

$$\pi_{i+1}(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a) U^{\pi_i}(s') \tag{9.2}$$

Iterate this method over and over again to approach to the optimal policy.

9.3 Q-Learning

9.3.1 Problem

In real world situations, the search space is often huge, e.g. chess has 10^{40} states. It would be too expensive, if not impossible, to obtain the whole transition model. However, without a complete transition model, both value iteration and policy iteration would be inapplicable.

9.3.2 Solution

Intuitively, if we can design a algorithm that obtains the utilities without using the transition model, that would be the solution. However, not using transition model means that we must find an alternative method to iteratively update, and one way is to manually find a balance between the current utility (**exploit**) and the utilites we compute in the next iteration (**explore**).

9.3.3 Algorithm

Firstly, for simplicity, let's do some mathematical manipulation.

$$\begin{aligned} U(s) &= R(s) + \gamma \ max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s') \\ &= max_{a \in A(s)} (R(s) + \gamma \sum_{s'} P(s'|s,a) U(s')) \end{aligned} \tag{9.3}$$

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

then we have $U(s) = max_{a \in A(s)} Q(s,a)$.

$$\begin{aligned} Q(s,a) &= R(s) + \gamma \sum_{s'} P(s'|s,a)U(s') \\ &= R(s) + \gamma \sum_{s'} P(s'|s,a) max_{a' \in A(s') Q(s',a')} \end{aligned} \tag{9.4}$$

Now we can go through the steps of the algorithm.

Step 1: Initialize

Initialize $\hat{Q}(s,a)$

Step 2: Choose an action

$\hat{a} \leftarrow argmax_{a \in A(s)} \hat{Q}(s,a)$

With a probability of β, we choose $\hat{a}$.

Otherwise, we randomly pick an action.

Step 3: Update

As mentioned before, we have to manually balance the current utility and the new utilities. Here we define a weight factor α, and the update equation will be

$$\begin{aligned} \hat{Q}(s,a) &= (1-\alpha)\hat{Q}(s,a) + \alpha(R(s) + \gamma max_{a'} \hat{Q}(s',a')) \\ &= \hat{Q}(s,a) + \alpha(R(s) + \gamma max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)) \end{aligned} \tag{9.5}$$

Note that α is not static, α should be high (close to 1) at the beginning and decreases to 0 as time passes by.

9.4 Approximate Q-Learning

9.4.1 Problem

While Q-Learning has got around with transition model, it require space to store all the pairs of (s,a), the space requirement can be huge in real world. To solve this problem, we need to find another solution.

9.4.2 Solution

We define another Q function:

$$Q(s,a) = \sum_{i=1}^{n} f_i(s,a)\omega_i \tag{9.6}$$

Here, $f_i(s,a)$ represents the ith feature of (s,a), and ω represents the weight of this feature because every (s,a) can have multiple features to evaluate.

Unlike Q-Learning, here we do not directly update Q(s,a), instead, we update the weights. With simple mathematical manipulation of the update function for Q(s,a), we obtain the update function for ω:

$$\begin{aligned} \hat{Q}(s,a) &= \hat{Q}(s,a) + \alpha(R(s) + \gamma max_{a'}\hat{Q}(s',a') - \hat{Q}(s,a)) \\ \hat{\omega}_i &= \hat{\omega}_i + \alpha(R(s) + \gamma max_{a'}\hat{Q}(s',a') - \hat{Q}(s,a))\frac{\partial \hat{Q}(s,a)}{\partial \hat{\omega}_i} \end{aligned} \tag{9.7}$$