Q learning

N.D. Van Foreest

July 4, 2017

Contents

1 Intro

I want to apply Q learning to the EOQ model. First I want to get the tabular Q learning algorithm to work. For this, I copy from Bertsekas, Vol 2, Section 6.4 on Q learning. Then I'll apply it to the EOQ model.

2 Q factors

Define the Q-factors as

$$Q^*(i,u) = \sum_j p_{ij}(u)\left[g(i,u,j) + \alpha J^*(j)\right],$$

where $g(i,u,j)$ is the reward of taking action u in state i and end up in state j, and $J^*(j)$ is the value of continuing in state j. With this expression for $Q^*(i,u)$ the dynamic programming (Bellman) equations take the form

$$J^*(i) = \max_{u\in U(i)}\left\{\sum_j p_{ij}(u)\left[g(i,u,j)+\alpha J^*(j)\right]\right\} = \max_{u\in U(i)} Q^*(i,u).$$

From this we can get an optimality equation for the Q factors:

$$\begin{aligned}
J^*(j) = \max_{u'\in U(j)} Q^*(j,u') &\iff \\
\alpha J^*(j) = \alpha \max_{u'\in U(j)} Q^*(j,u') &\iff \\
\sum_j p_{ij}(u)\alpha J^*(j) = \sum_j p_{ij}(u)\alpha \max_{u'\in U(j)} Q^*(j,u') &\iff \\
\sum_j p_{ij}(u)\left[g(i,u,j)+\alpha J^*(j)\right] = \sum_j p_{ij}(u)\left[g(i,u,j)+\alpha \max_{u'\in U(j)} Q^*(j,u')\right] &\iff \\
Q^*(i,u) = \sum_j p_{ij}(u)\left[g(i,u,j)+\alpha \max_{u'\in U(j)} Q^*(j,u')\right].&
\end{aligned}$$

We can use this to compute Q^* by value-iteration. Start with $Q(i,u) = 0$ (or some other set of random values), and iterate according to

$$Q(i,u) := \sum_j p_{ij}(u) \left[g(i,u,j) + \alpha \max_{u' \in U(j)} Q(j,u') \right]. \tag{1}$$

Now we can do two things for Q-learning. The first is to simulate j and $g(i,u,j)$ from the pair (i,u) (see below how to do this), and take $\alpha = 1$ to obtain a total cost during a finite horizon, and replace the above expectation by the update rule

$$Q(i,u) = g(i,u,j) + \max_{u' \in U(j)} Q(j,u').$$

To explore and exploit and we can choose to use $u' = \arg\max Q^*(j,u')$ with, for instance, probability 0.9 and otherwise take u' uniform in $U(j)$.

The second thing is to introduce a damping factor $\gamma \in [0,1]$ in (1) so that we obtain

$$\begin{aligned} Q(i,u) &= (1-\gamma)Q(i,u) + \gamma \sum_j p_{ij}(u) \left[g(i,u,j) + \alpha \max_{u' \in U(j)} Q(j,u') \right] \\ &= Q(i,u) + \gamma \left(\sum_j p_{ij}(u) \left[g(i,u,j) + \alpha \max_{u' \in U(j)} Q(j,u') \right] - Q(i,u) \right). \end{aligned}$$

Now we simulate j and $g(i,u,j)$ from the pair (i,u) and replace the above expectation by the update rule

$$Q(i,u) := Q(i,u) + \gamma \left(g(i,u,j) + \alpha \max_{u' \in U(j)} Q(j,u') - Q(i,u) \right). \tag{2}$$

To explore and exploit we can use the same trick as before.

Remark 2.1. To simulate j from the pair (i,u), we can use the transition probabilities $p_{ij}(u)$ as follows. Let J be the next state (random variable) from state (i,u). Then set $\mathsf{P}_{(i,u)}\{J = j\} = p_{ij}(u)$. Now that we have generated the next state j from (i,u) we can compute the reward $g(i,u,j)$.

3 Using Heuristics To initialize the Q values

Suppose we ask a human to solve the optimization problem at hand. Of course we don't expect that the human player obtains an optimal solution, but we like to use the actions of the human to give the optimizer a head start. There must be a number of ways to achieve this. So, lets assume the user played the game a few times. (I also assume for ease that the player took the same actions in the same state. This is not true of course, but we can tackle this issue later.). Now we have a trace $T = ((i_1,u_1),(i_2,u_2),\ldots)$ (so the user chose action u_1 in state i_1, and so on.). In other words, a trace is the set of states and actions that occured during the game played by the human. (Here a trace is an ordered set, i.e., a tuple, of how the game proceeded.)

Once we have a trace, or a few traces, we need to initialze the Q values to start the learning algorithm. One simple way is like this:

$$Q(i,u) = \begin{cases} \max_j g(i,u,j), & \text{if } (i,u) \in T, \\ 0 & \text{if } (i,u) \notin T. \end{cases}$$

Assume here that $g(i,u,j) \geq 0$ for all i,u,j. For inventory or scheduling problems with infinite horizon (so there is no ∂) but a reward per step (acceptance of a job), this initialization might be ok.

This choice is not ok if $g = 0$ for many i,u,j, and when the player only gets a reward at the end of the game, like in chess or go. So, assuming that the player plays the game until it finishes, then let (i_n,u_n) be the finishing state of the game, and n the finishing time. As the game stopped here, it must be that $g(i_n, u_n, \partial) > 0$, where ∂ is the coffin(finish) state). For instance, $\partial =$ mate for chess. Then set

$$Q(i,u) = \begin{cases} g(i_n, u_n, \partial), & \text{if } (i,u) \in T, \\ 0 & \text{if } (i,u) \not\in T. \end{cases}$$

a) How to handle different choices of the player in the same state (i,u)?

b) There is some explanation in Sutton's book on how to use traces to update the Q-values. I forgot where precisely, but I suspect it is in chapter 4.

c) Isn't there anything in Bertsekas, dynamic progamming volume 2, on this too? I don't think so, but it might be interesting to check.

4 An Application to an inventory system that satisfies the EOQ assumptions

Standard EOQ model under periodic review such that the review epochs are synchronized with the demand arrivals.. Orders are placed at the start of a period and arrive right away, i.e., immediate replenishments.

$D =$	Demand per period,
$h =$	Holding cost due at the end of the period,
$p =$	Selling price per unit,
$K =$	Ordering cost per order,

Let I_n be the inventory at the end of the period and Q_n the order size. Assuming that replenishments arrive at the start of the period, I_n satisfies the recursion

$$\begin{aligned} S_n &= \min\{I_{n-1} + Q_n, D_n\}, \\ I_n &= \max\{I_{n-1} + Q_n - s_n, I_{\max}\}, \end{aligned}$$

where S_n is the sales during period n. Thus, in the notation of Section 2, the inventory $I_{n-1} = i$, the action $u_n = Q_n$ and $j = I_n$. With this, it is easy to compute $p_{ij}(u)$.

The end-of-period reward is given by

$$R_n = p s_n - h I_n - K \mathbb{1}_{Q_n > 0}.$$

Thus, $g(i,u,j) = R_n$.

The state space is $I_n \in \{0, 1, \dots, I_{\max}\}$, the action space is $\{0, \dots, Q_{\max}\}$.

In the learning procedure we want to reduce the exploration rate while the number of episodes progresses. We take

$$\beta(i) = 0.8 + 0.2 \frac{i}{N},$$

if i is the current episode of N episodes in total. I also tried the rule

$$\beta(i) = 0.8$$

for all i. This also gave good results.

4.1 First test

TBD: Include results.

Figure 1: piet

4.2 (s,S) -policies

TBD: Include results.

4.3 Possible Next Steps

- One way that must give much better results is to use (1) for the iteration, rather than use (2). That must be a major improvement. Of course, this doesn't work for large problems, but is of interest to see how these two compare.
- (S,s) ?
- (Q,r) policy
- use neural network. Check the ice/hole/example on how to do this.