Q-Learning in the Evolutionary Pricing Game

November 19, 2022

Abstract

We describe a possible set-up of Q-learning of an agent in the "duopoly with demand inertia" pricing game. We discussion in Section 2 various issues such as the reward function or visiting states of the Q-table.

1 Q-learning

We use the Q-learning model described in Watkins and Dayan (1992), which also proves a convergence result under certain conditions. We have finite sets X of *states* and A of *actions*. At each time step t, the agent knows that she is in state x_t and chooses an action a_t , with a probabilistic next state and a current reward r_t, which has a mean value $R(x, a)$. The probability for the next state y is determined by the environment (which in our setting may change, see below) and depends on the state x and chosen action a according to

$$\text{prob}[y \mid x, a]\,. \tag{1} \qquad \boxed{\texttt{prob}}$$

A *policy* π maps states to actions. The rewards are discounted with a weight factor $\gamma < 1$ for future rewards. That is, if r_t is the reward for period $t = 0, 1, 2, \ldots$, then the total reward is

$$\sum_{t=0}^{\infty} \gamma^t\, r_t\,. \tag{2} \qquad \boxed{\texttt{total}}$$

The value of an initial state x under policy π is

$$V^{\pi}(x) = R(x, \pi(x)) + \gamma \sum_{y \in X} \text{prob}[y \mid x, \pi(x)]\, V^{\pi}(y)\,. \tag{3} \qquad \boxed{\texttt{value}}$$

An optimal *stationary* (that is, time-independent) policy π^* fulfills

$$V^*(x) := V^{\pi^*}(x) = \max_{a \in A} \left(R(x, a) + \gamma \sum_{y \in X} \text{prob}[y \mid x, a]\, V^{\pi^*}(y) \right). \tag{4} \qquad \boxed{\texttt{opt}}$$

For a policy π, the Q-value (or action-value) is the expected discounted reward for action a at state x and following policy π thereafter:

$$Q^{\pi}(x,a) = R(x,\pi(x)) + \gamma \sum_{y\in X} \mathrm{prob}[y \mid x, \pi(x)]\, V^{\pi}(y)\,. \tag{5}$$ defq

The object in Q-learning is to estimate the Q-values for an optimal policy π^*. Define them as $Q^*(x,a) := Q^{\pi^*}(x,a)$. Hence, by (3), $V^*(x) = \max_{a\in A} Q^*(x,a)$, and if a^* is an action at which the maximum is attained in state x, then an optimal policy can be formed as $\pi^*(x) = a^*$.

In Q-learning over time steps t (assuming some initialization of the Q-values $Q_0(x,a)$), the agent assumes the best she can do from state y is $\max_{b\in A} Q_{t-1}(y,b)$. The Q-values are only updated for the current state x_t and action a_t and with a learning rate α_t , which decreases with t, according to

$$Q_t(x_t,a_t) = (1-\alpha_t)\, Q_{t-1}(x_t,a_t) \ + \ \alpha_t\Big[r_t + \gamma \max_{b\in A} Q_{t-1}(y_t,b)\Big], \tag{6}$$ wd1

where y_t is the *observed* subsequent state after taking action a_t in state x_t . Note that this is about *learning* the Q-values, assuming the agent will choose the best possible action b in the future, based on the evidence $Q_{t-1}(y_t,b)$ of the past for the observed next state y_t . The action a_t is not chosen according to an optimization method. However, this can be combined with ε-*greedy* choices of a_t where a random action is taken with probability ε, and the currently considered optimal one with probability $1-\varepsilon$, where ε is tending to zero over time.

Watkins and Dayan (1992) proved that Q-learning converges to an optimal strategy if the rewards r_n are bounded, every state-action pair is visited infinitely often, and the learning rate is both slow enough to allow for learning ($\sum_{t\ge 0}\alpha_t = \infty$) and fast enough to accumulate knowledge ($\sum_{t\ge 0}\alpha_t^2 < \infty$).

2 Application to the duopoly pricing game

s-appl

We want to apply Q-learning to learn strategies, represented by agents, in the oligopoly game by Selten (1965) for the special duopoly case of the strategy experiments by Keser (1992). In this game, the learning environment is in several respects non-stationary, which we will discuss and address.

As studied by Keser (1992), the game is played over fixed number T of rounds, here $T = 25$. Each firm i has a *demand potential* D_i that determines its number of $D_i - p_i$ of sold units of a good when setting a price p_i (firm i's decision in each period), with a profit of $p_i - c_i$ per unit for the firm's production cost c_i. The firms have different costs, $c_1 = 57$ and $c_2 = 71$. The myopic monopoly profit maximizes $(D_i - p_i)(p_i - c_i)$ when $p_i = (D_i + c_i)/2$.

At the start of the T periods, both firms have the same demand potential $D_1 = D - 2 = 200$. After each period, the cheaper firm gains demand potential from the more expensive firm in proportion to their price difference, according to

$$\begin{aligned} D_1^{t+1} &= D_1^t + \tfrac{1}{2}(p_2^t - p_1^t)\,, \\ D_2^{t+1} &= D_2^t + \tfrac{1}{2}(p_1^t - p_2^t)\,. \end{aligned} \tag{7}$$ demand

The total profits are summed up (there is also a discount factor of 1 percent per time period that favors early profits, which we ignore).

An agent learns a *strategy* in this game, which chooses the price for each period, in principle based on the full history of the player's own and the opponent's prices for all past periods.

2.1 States and actions

To simplify the state set, we propose the demand potential D_i of firm i as its *state*, discretized for example to multiples of 10. As its action a, we take much its price is *lower* compared to its myopically optimal price $(D_i + c_i)/2$, that is, $p_i = (D_i + c_i)/2 - a$, because it is never optimal to ask a higher price, and because this will be a less variable choice of the firm's action than the price itself. A first discretization of a could be in multiples of 5, that is, $a \in \{0, 5, 10, \ldots\}$. Recall that a firm is not allowed to charge a price below its cost, with the constraint $p_i \geq c_i$, which also restricts the possibilities for a depending on its demand potential.

2.2 Learning environment

s-env

We want the agent to learn how to play over T periods. This is done by playing against another, fixed agent, who does *not* learn at the same time, over T periods. Rather, that opponent is drawn from a pool of existing strategies, to which we will add a newly successfully trained agent. The probability of the opponent's choice is given by a mixed equilibrium among the existing agents.

Because the agent is meant to play well over the T periods, this should also be the scenario how the state (the demand potential) evolves. That is, the agent cannot freely or randomly choose the next state. Rather, the next state is determined by the game rules, and depends on the current state, the action a chosen by the agent, and the price of the opponent. The price of the opponent is not known (and presumably random depending on which opponent has been chosen). However, given the opponent's price, the game rules imply that choosing $a + b$ rather than a means the next state (demand potential) is $y + b/2$ rather than y.

There is a natural tension between short-term reward and future reward. If x is the current state and c the agent's cost, then the myopic profit is $(x-c)^2/4$. If the price is reduced by a from the myopic price $(x+c)/2$, then the current profit is $(x-c)^2/4-a^2$. However, higher values of a favor better states in the future because they increase x.

The Q-values should reflect the higher future reward.

2.3 Reward update

As the game is defined, what counts in an interaction is the player's total reward over T periods. The values of the Q-table should reflect that total reward. This is not easy to do consistently unless the current time period is added to the state, that is, a state is a pair (d,t) for a (discretized) demand potential and time period t. However, this multiplies the number of states by T.

Instead, consider now the *discounted* version of this game over infinitely many periods with discount factor γ, which can be implemented by taking γ to be a continuation probability from one period to the next. For example, if $\gamma = 0.96$, then the expected number of periods is $1/(1-\gamma) = 25$. The actual length of interaction is random and has exponentially small probability for very long runs, so it can be simulated without difficulty. The discounted version has the advantage of a stationary environment because the future looks the same at each time period.

In this version, consider the discounted overall reward in (2). If the reward r_t in each period is constant, say given by r, then the total reward is $r/(1-\gamma)$. It makes sense to normalize this by multiplying it with $(1-\gamma)$ to represent then the per-period reward, which is independent of γ if that reward is constant. This normalization is useful for the comparison of different discount factors.

The Q-values in (6) can be scaled by multiplication with any constant factor. If that factor is $(1-\gamma)$, then this equation becomes

$$Q_t(x_t,a_t) = (1-\alpha_t)\,Q_{t-1}(x_t,a_t) \;+\; \alpha_t\Big[(1-\gamma)\,r_t + \gamma \max_{b\in A} Q_{t-1}(y_t,b)\Big]. \qquad (8) \quad \boxed{\texttt{Qnorm}}$$

That is, the per-period reward r_t is multiplied by $(1-\gamma)$, which for γ near 1 is relatively small. This is important if the Q-values are meant to represent average payoffs per period.

One could adapt (8) to the case of fixed run lengths over T periods, by letting γ change with t. For example, the reward r_0 in the first period (if we number the periods $t = 0,\ldots,T-1$) has weight $1/T$ compared to the rest of the game. The reward r_1 has weight $1/(T-1)$, and in general r_t has weight $1/(T-t)$, which is the full weight 1 for the last period $t = T-1$, in agreement with how the game is played. The overall reward is then the average reward $\frac{1}{T}\sum_{t=0}^{T-1} r_t$ as intended. By writing (8) with $\gamma = 1 - 1/(T-t)$, this could be the update rule for the values of Q even "by stealth", that is, with the agent not informed about t. The assumption is that the learning takes place over the fixed number T of periods, and is re-started (but keeping the computed values) when playing against a new opponent.

The discounted version (8) of this game seems safer in this regard. The goal is that the Q-values approximate the expected future average reward *per period* for a given state (demand potential), in order to be comparable across different run lengths and possibly different discount factors. Varying the discount factor should be interesting with respect to the evolving behavior. This requires the current reward r_t to be weighted with $(1-\gamma)$ in (8).

2.4 Visiting all states

In order for Q-learning to converge, all state-action pairs need to be tried sufficiently often. However, many demand potentials are unrealistic to be encountered in typical play, even though the agent needs to be prepared what to do when such a state is encountered. The fact that actions a are relative to the myopic optimal price already leads to reasonable behavior.

As mentioned in 2.2, it makes sense to let an agent play over a number of periods against her opponent and thereby visit the states (demand potentials) typically encountered in actual play. This restricts the states that will be visited. For example, against a myopic opponent who on their side always plays $a = 0$, the agent will not lose own demand potential but only gain it (at least this holds for the low-cost firm).

However, we *expect agents to learn* to play more aggressively against such opponents by lowering their prices (with own higher values of a), which lowers their opponent's demand potential. Once such opponents are part of the population, they will lead to lower demand potentials of the agent herself, and thus give opportunities to learn.

Recall that if the agent plays over T periods (with T fixed or random) against her opponent, the next state is part of the trajectory of states and cannot just be chosen freely in order to learn.[1] The agent's free choice of actions does not imply that they can freely reach all states with their next move.

Another possibility to reach different states during the learning phase is a *random* rather than equal split of the initial demand potentials. In the extreme, such a restart could be made even after one or two time steps, in order to populate the

[1]This must be a common problem in machine learning where you learn "as you go".

Q-table more quickly. However, it is not clear whether this will then reflect realistic future rewards.

2.5 Initializing Q-values

Another question is how to populate the initial Q-values. Rather than letting them to converge from a random or zero initial value, one could try to use more realistic values. For example, if the future demand potential is constant at y and the agent plays a and has cost c, then their reward per period is $(y-c)^2/4-a^2$. This is a reasonable table entry, which could be modified with some random alteration of a. The assumption that a future demand potential is constant is justified if prices of the two players converge to each other, which has been observed in play. One could also compare this with the Q-values that eventually result.

At any rate, it should speed up learning significantly, but may risk leading to too constrained behavior.

2.6 Discretization

The actual state is a continuous real. It could be discretized to integers, but the number of states would still be very large. With discretization at multiples of 10, the state for the Q-update could be chosen randomly according to the closest states: E.g. if the actual state is 207, chose 210 and 200 with probabilities 0.7 and 0.3, respectively. Alternatively one could even do the update in (8) in a weighted manner.

An alternative are *tilings* of the state that use overlapping grids (essentially separate Q-tables) that are updated simultaneously with each action and later interpolated.

Deploying the agent, once she has "learned enough" (which in itself is to be determined) requires defining a chosen action by the agent as a function of the state. According to the learned Q-table, this should be the optimal action. This action can be interpolated from the optimal actions for the states, which serve as breakpoints in a linear interpolation. This will simplify the description of an agent substantially, although it may be of some interest to keep the learned Q-table.

Later extensions would also involve more complex states, like pairs of triples of demand potentials for the last two or three periods rather than just one period.

2.7 The population game and its evolution

Once an agent has been trained (as a low-cost or high-cost firm), it will be added to the population game as a new row or column. For that purpose, the new agent plays, as it will be deployed, against every existing agent and records a pair of payoffs, one for each player, in the bimatrix game.

If the length of the interaction is random, it may be necessary to play against the same opponent more than once and take averages (it would be useful to record the resulting variance in payoffs).

In particular, this determines the expected payoffs against the mixed equilibrium strategies used as the learning environment in 2.2. One would assume that the new agent does better than the other (say, row) players in the equilibrium. This can also be used as a criterion for declaring learning as successfull, because the new agent is a potential "entrant" in an evolutionary setting.

With the new agent added, new equilibria are determined. It could happen that new agent merely destroys or changes the equilibrium that it has been trained against, but does not participate in any equilibrium. To be tested if this occurs.

References

Keser1992 Keser, C. (1992). *Experimental Duopoly Markets with Demand Inertia: Game-Playing Experiments and the Strategy Method*, volume 391 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, Berlin.

Selten1965 Selten, R. (1965). Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit: Teil I: Bestimmung des dynamischen Preisgleichgewichts. *Zeitschrift für die gesamte Staatswissenschaft/Journal of Institutional and Theoretical Economics* 121(2), 301–324.

WD92 Watkins, C. J. and P. Dayan (1992). Q-learning. *Machine learning* 8(3), 279–292.