Plicy Gradient methods in the Evolutionary Pricing Game

September 18, 2023

Abstract

The deep reinforcement learning model of an agent in duopoly multi-round pricing game and applying policy gradient methods to find the optimal strategy.

1 Multi-round pricing game

1.1 End effect

2 Naive Policy Gradient method

$$\begin{tikzcd} S_0 \arrow[r, bend left, "{a_0,\ r_0}"] & S_1 \arrow[r, bend left, "{a_1,\ r_1}"] & \cdots \arrow[r, bend left, "{a_{24},\ r_{24}}"] & S_{25} \end{tikzcd}$$

f1equi

Figure 1 environment model

In *action-value* methods such as Q-learning, the value of of each state or state-action pair is learned and based on these values, the policy would be determined. However, in *Policy Gradient* methods, the policy is learnt directly without using value functions for action selection.

As a first step, we implemented a naive policy gradient algorithm.

Since the number of states in our model is large, we need a function approximator to parametrize the action preferences. We use a artificial neural network for this purpose and we show the vector of connection weights in our network as θ.

The actions at each state are chosen in a way that action with higher valuation is more likely to be chosen.

2.1 Action space

The reward in each stage of the multi-round pricing game is determined by $(D_i - P) \times (P - C)$ in which D_i is the agent's demand potential at the start of stage i and C is the agent's product cost. $P^* = \frac{D_i + C}{2}$ maximises this quadratic concave function; we refer to P^* as *myopic price* or *monopoly price* at stage i. Playing the myopic price results in maximum payoff at the current stage but the demand potential for the next round would be affected, meaning it does **not** necessarily result in the maximum overall return.

We considered actions as the value below the monopoly price that should be played, in order to attract more demand and consequently more reward in later stages. In our model the action-space is discrete. In each stage, action $a \in \{0, 1, ..., 19\}$ with step= 3 that means the price can be $0, 3, 6, ...,$ or 57 units less than the stage's myopic price.

For example, for the low-cost player ($C = 57$), if $D_i = 180$ at the start of stage i and action $a = 5$:

$$P^* = \frac{180 + 57}{2} = 118.5 \rightarrow P = P^* - (a \times \text{step}) = 118.5 - 5 \times 3 = 103.5$$

The actions are determined by sampling from the probability distribution over the action space that is the output of neural network.

2.2 State representation

The state should provide all the information that is needed for making a decision at each state. In our model, the state includes following :

- current stage of game
- agent's current demand potential (d)
- agent's price in the last stage
- adversary's price history (last 3 stages)

encoding of stage	demand potential	last price	adversary's price history

2.3 Adjustments to the learning

After the basic implementation of the policy gradient algorithm, we noticed that the agent does not learn the end effect of the game. In the following, we explain the techniques we applied to our model to help it learn the end effect.

2.3.1 one-hot encoding of stage

In the basic model, stage was represented in the state as a real number in $[0, 1]$. To see the end effect more obvious, we used the one-hot encoding of stage in the state representation instead.

	i_1	i_2	...	i_{24}	i_{25}
stage 1	1	0	...	0	0
stage 2	0	1	...	0	0
...	...	...	...	...	...
stage 24	0	0	...	1	0
stage 25	0	0	...	0	1

The game has 25 stages so we use an array of size 25 and in each stage the corresponding index would be 1 and the other indices would be 0.

2.3.2 Learning backwards

Another technique we used to make the end-effect more clear to the learning agent, is learning backwards. The game is played as before but the difference is that for the first episodes, only the weights of last action (action for the last stage) in our neural network would be updated. Then, for some episodes, just the weights for last 2 actions would be updated, and so on. In the last episodes, the weights for all the actions would be updated.

In this way, the agent will have some time to optimize the action for last stage in the first steps which helps it understand the end-effect better.

2.4 Structure of Neural Network

2.5 Loss function

2.6 Using Adam optimizer

2.7 Structure of neural network

3 Monte Carlo Policy Gradient

4 Reinforcement with Baseline

5 Actor-Critic method