Mathematics Laboratory Final Project

Reza Arabpour, 610397136

Fall 2022

Abstract

In this project we first do a review on the paper *"A Recurrent Neural Network for Game Theoretic Decision Making"*, by Sudeep Bhatia and Russell Golman, 2014, introducing a computational model to solve the outcome of strategic games. Then we go through the codes which has been impelemented in Matlab. There we will see the code to generate a game, "*game_creator.m*", and then the "*main.m*" code which runs the simulations in a parallel using two other functions named "*stage_update.m*" and "*loop_check.m*". After that, we will see my own idea to model a real world problem and try using the taught materials in this course to face the problem, which is a market simulation with many players who are making random decisions for many round and making supply and demand which results in changes in price.

1 Introduction

Strategic decision making is an important feature of high-level cognition. From coordinating meeting times to cooperating on research projects or negotiating household chores, interdependent scenarios –in which the choices of an individual are contingent on the choices of others– are ubiquitous in everyday life. Game theory provides a mathematical framework with which rational decision making in interdependent scenarios can be formally represented and analyzed. It is an important area of research in economics, political science, computer science, and philosophy, and it is widely used for studying behavior in social settings, where it describes and predicts the types of cooperation, conflict, and coordination observed in groups of human decision makers.

In this paper we present a connectionist model that is able to reason through game theoretic problems in two-player games. Our model is based on Kosko's (1988) Bidirectional Associative Memory (BAM) network, which is a minimal two-layer recurrent neural network (RNN) with binary activation functions and binary connection weights. We assume that the two layers in our network represent the strategies available to the two players, with connections between these layers encoding best-responses to any given strategy.

2 Theory

Note : here we are just telling the theorems and writing a report, not an article! Thus, if we intentionally avoid bringing the proofs and the fine-grained detail,

but all the material have been extracted from the paper mentioned in abstract. Thus, more details and proof of the assumptions made here can be found there.

2.1 Game Theoretic Decision Making

We define a finite-strategy two-player game with a set of pure strategies for each player, $S_1 = \{S_{11}, S_{12}, ..., S_{1N}\}$ and $S_2 = \{S_{21}, S_{22}, ..., S_{2N}\}$ respectively, and a pair of payoff functions u_1 and u_2 that give each player's utility for each profile of pure strategies (S_{1i}, S_{2j}). Thus if player 1 selects S_{1i} and player 2 selects S_{2j} the utility for player 1 is $u_1(S_{1i}; S_{2j})$ and the utility for player 2 is $u_2(S_{2j}; S_{1i})$. We use the notation u_{ij} as a shortcut for$(u_1(S_{1i}; S_{2j}), u_2(S_{2j}; S_{1i}))$.

The most standard solution concept for a strategic game is Nash equilibrium, which relies on common knowledge of rationality and accurate (so-called "rational") expectations. A Nash equilibrium is a strategy profile in which no player can obtain higher utility by unilaterally changing her strategy; each player is already playing a best response to the equilibrium strategy profile. We define the set of best responses for player μ to an opponent's strategy $s_{-\mu}$ as $BR(s_{-\mu}) = arg\ max\ u_\mu(s_\mu; s_{-\mu})$. Then a pure strategy Nash equilibrium can be defined as a strategy profile (s_{1i}, s_{2j}) $s.t.$ $s1 \in BR(s_{2j})$ and $s2 \in BR(s_{1i})$.

2.2 Bidirectional Associative Memory

The Bidirectional Associative Memory Network (BAM) is a particularly powerful (and mathematically tractable) constraint satisfaction neural network. It consists of two layers with binary connections between their respective nodes and binary activation functions for any given node. When the connections between its nodes are symmetric then BAM is guaranteed to reach a stable pattern of activation, regardless of its starting state. This property has been used by scholars to solve a variety of practical tasks involving associative memory and pattern completion and also model human decision making and the biases that it often involves. In this paper, we model how an individual decision maker reasons through two-player finite strategy games, using the BAM network. Strategies in these games for each of the two players can be represented in each of BAM's two layers. We will assume that the first layer in the BAM network represents strategies available to the decision maker (or self). If the decision maker can choose from the set of strategies $S_1 = \{S_{11}, S_{12}, ..., S_{1N}\}$, then the first layer in our network consists of N nodes, with node i representing strategy S_{1i}. The activated nodes in this layer represent the strategies that the decision maker considers playing in the game.

As mentioned earlier, node activation in BAM is binary, with each node being on or off. We will assume that every node has the same binary activation function, with activation triggered by strictly positive input. For any node k (in either layer of the network) with input I_k, the activation function f_k is specified by the following equation:

$$f_k(I_k) = \begin{cases} 1\ if\ I_k > 0 \\ 0\ if\ I_k \leq 0 \end{cases} \tag{1}$$

Connections between nodes are also binary, with each node in the first layer either connected or not connected to each node in the second layer, and each

node in the second layer either connected or not connected to each node in the first layer. There are no connections between two nodes in one layer.

3 Model Results

3.1 Unique Nash Equilibrium

First, suppose the network reaches a stable state corresponding to a pure strategy profile, that is, a stable state of activation in which only one node is activated in each layer of the network. Our first result characterizes that stable activation state as corresponding to a Nash equilibrium. The example of the *"traveler's dilemma"* in the section below illustrates how the network converges on a pure strategy Nash equilibrium.

3.2 No Nash Equilibrium

The game of *"rock-paper-scissors"* discussed in the section below illustrates that the network may not converge to a stable state. Even if the network does not reach a stable state, however, we can characterize the nodes which may experience recurrent activation. In the long run, activation is restricted to the set of rationalizable strategies, R_1 and R_2 respectively. Which means that we should expect strategically sophisticated individuals only to play rationalizable strategies. The network may never converge to a state with stable activation, so we may not be able to identify a single strategy that will necessarily be chosen, but we can make testable predictions about what will not be chosen.

3.3 Multiple Nash Equilibrium

We can also recognize that the lack of a point prediction creates space for contextual factors to matter. An individual's eventual decision may depend on which strategy she considers first, which could in turn depend on the salience of different options, how the options are framed, or how the decision maker's attention is anchored. The *"military dilemma"* discussed in the section below illustrates how the starting point determines which of the multiple Nash equilibria is eventually selected by the network. It is straightforward to observe that any pure strategy Nash equilibrium would constitute a stable state in our network. The strategies in the Nash equilibrium would, due to the nature of the connection edges, reinforce each other and, once activated, sustain their activation.

4 The Code

4.1 Game Creator

Using this code one can create and inputs his own game and saves it for later use in the main program. Here we will give the computer the strategies and their best responses, which will be used later for creating the neural network.

4.2 Main

This code loads a game created using game creator program, and computes the calculations needed to find out in which of the above-mentioned scenarios this game will end ('Unique Nash Equilibrium' or 'Multiple Nash Equilibrium' or 'No Nash Equilibrium').

4.2.1 Stage Update

This function inputs the current status of the network, and outputs the network after executing calculations on the activation function and applying resulted changes.

4.2.2 Loop Check

This function will compair the current state of the network with all the previous states, to prevent getting stuck into a loop.

4.2.3 Stage Plot

This function was writen in order to plot the network graph, but since my PC had a problem with Matlab software and I did all the codes in Octave, I couldn't use the *"Digraph()"* function to create the graph and plotting it.

5 Experiment Results

5.1 Traveler's dilemma

In the original parable, two travelers have lost identical things and must request compensation between $2 and $10. The airline (which is responsible for the lost luggage) will accept the lower claim as valid and pay that amount to both players, and, to deter lying, will penalize the higher claimant with a $2 fee and will reward the lower claimant with $2 bonus.

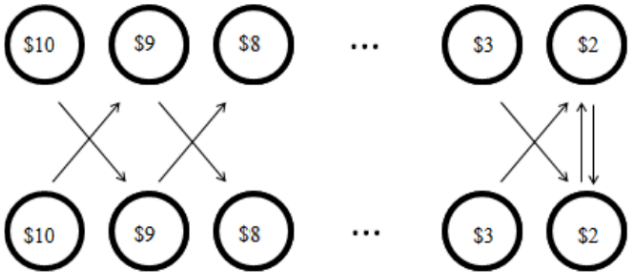

Figure 1: Traveler's dilemma result

5.2 Rock Raper Scissors

The classic game of rock-paper-scissors is the simplest symmetric, zero-sum game with non-transitive winning strategies. Each player has three pure strategies: rock, paper, or scissor. The loop is that rock "defeats" scissors, scissors

"defeats" paper, and paper "defeats" rock. If both players play the same strategy, then the game is a tie.

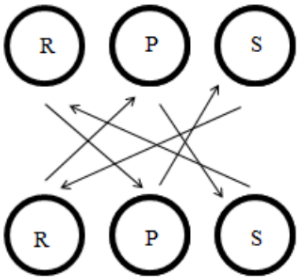

Figure 2: Rock Raper Scissors result

5.3 Military dilemma

In this game there are two neighbor countries, who are not friends. They are in a very tough competition and none of them wants to waste their money. But there is a problem, what if one of them attacks the other one? To avoid this from happening, if one of them starts to invest into their military, the second one will have to do so. But, investing on military is a time and resource consuming thing and if one of them does this, they will lose the progress competition! So they should decide what to do, and obviously this game has two nash equilibrium, one with both countries developing no military, and one with both countries invest on their military at the same time.

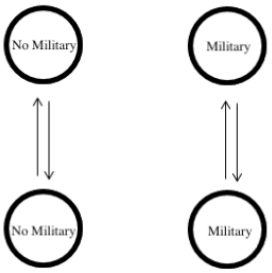

Figure 3: Military dilemma result

6 My Extension, The Market Simulator

When I was reading the article and did the coding, I came up with the idea of creating some thing out of it and model a real-world problem. The first thing that came into my mind was the market simulation, becuase it is has the game theoric properties and with some simplifications, It can be impelemented using the BAM neural networks. Thus, I coded this market Simulator in which there are some players and in each round they decide to whether to trade anything or not, them they decide about their order size and their desired price, finally

some orders from these people will be gathered and sent to an trading core which matches the buyers and sellers orders. Also, I have impelemented a feature which is the market sentiments and the user can adjust it to somehow see the effect of different levels of news-dependency in an asset. At the end of each round the price, people's assets ,and moeny will update. We do this for the number of wanted rounds and finally we output the market history and statistics in addition to plotting the price history and rounds trading data.

6.1 The Model

6.1.1 Inputs parameters

1. number_of_rounds : the number of rounds of simulation
2. number_of_players : the number of players
3. average_money : the average money people have at beginning
4. average_asset : the average shares poeple have at beginning
5. initial_price : the initial price for asset
6. sentimnet_effect : wheter to consider market sentimnet
7. sentiments_shift : the big picture sentiments or the bias
8. sentiments_multiplier : the sentiments effect multiplier

6.1.2 Outputs

1. brooker_profit = the amount of differences in matching prices which is the profit of the brooker
2. greatest_winner = the amount of mony that the most profitable person in this model had made
3. greatest_loser = the amount of mony that the most loser person in this model had lost
4. average_change = the average change that had happened to people's wealth

6.1.3 Outputs options

1. rounds_details = the details for each round, containing the change in price, number of round trades, and total volume of trades in round
2. more_stats = plots more market statistics, containing the history of volume, trades, and average volume per trade in each round
3. people_stats = plots more statistics about people, containing the average money and assets people had in each round and the changes happend in wealth distribution

6.2 The code

6.2.1 market_simulator.m

this function is the main function of the program and handles the input and output related things, plus creating the initial state of the model.

6.2.2 Order_Core.m

this function inputs a list of orders and does all the calculations for each round and apply changes to people's money and assets.

6.3 Results

Here we will see a sample of outputs of the model, for a market with 50 players who play 200 rounds and having about 1000$ and 100 shares per average at beginning. The initial price of the asset is 80 dollars (like AMD in Nasdaq), also, we include the sentiments with a -2 big picture (like the current recession expectation). We set the sentimnet multiplier to 1, to have a normal and not very sharp moving share.

6.3.1 Inputs parameters

```
please enter the number of rounds of simulation : 200
please enter the number of players : 50
please insert the average money people have at beginning : 1000
please insert the average shares poeple have at beginning : 100
please enter the initial price for asset : 80
Do you want to consider random market sentimnet ? (1 for yes, 0 for no) 1
enter the big picture sentiments (or 0 to ignore) : -2
enter the sentiments effect multiplier (or 1 to ignore) : 1
```

Figure 4: inputs

6.3.2 Outputs

```
The broker has made : 10349.7543dollars !
the greatest winner had made 38837.9486 dollars!
the greatest loser had lost 109.8392 dollars!
the average change people have made was 12416.5185 dollars!
```

Figure 5: outputs

6.3.3 Plots

- rounds_details

```
[1,190] = Round 190; change in price = -0.283664, number of trades = 9, total volume of trades 5.62381 dollars
[1,191] = Round 191; change in price = 2.68092, number of trades = 9, total volume of trades 5.73307 dollars
[1,192] = Round 192; change in price = 1.39623, number of trades = 16, total volume of trades 16.2369 dollars
[1,193] = Round 193; change in price = 1.63137, number of trades = 14, total volume of trades 30.8272 dollars
[1,194] = Round 194; change in price = -0.00881295, number of trades = 17, total volume of trades 17.3897 dollars
[1,195] = Round 195; change in price = -1.06541, number of trades = 9, total volume of trades 0.0309609 dollars
[1,196] = Round 196; change in price = -0.969584, number of trades = 10, total volume of trades 1.31419 dollars
[1,197] = Round 197; change in price = -1.36351, number of trades = 11, total volume of trades 10.1885 dollars
[1,198] = Round 198; change in price = -1.21813, number of trades = 14, total volume of trades 4.01641 dollars
[1,199] = Round 199; change in price = 1.19142, number of trades = 13, total volume of trades 23.2847 dollars
[1,200] = Round 200; change in price = 0.256789, number of trades = 20, total volume of trades 7.99262 dollars
```

Figure 6: round details sample result

- more_stats

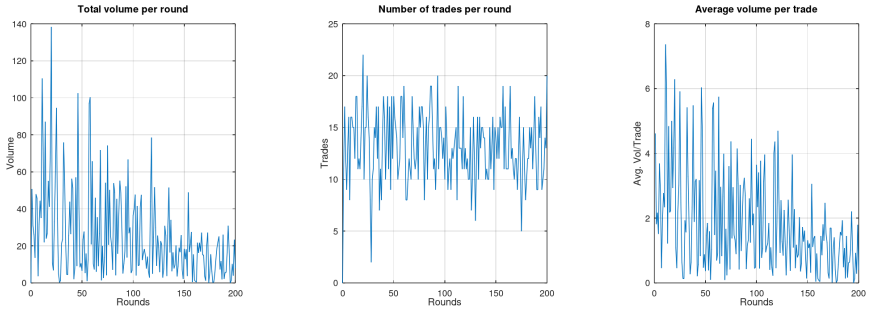

Figure 7: more stats result

- people_stats

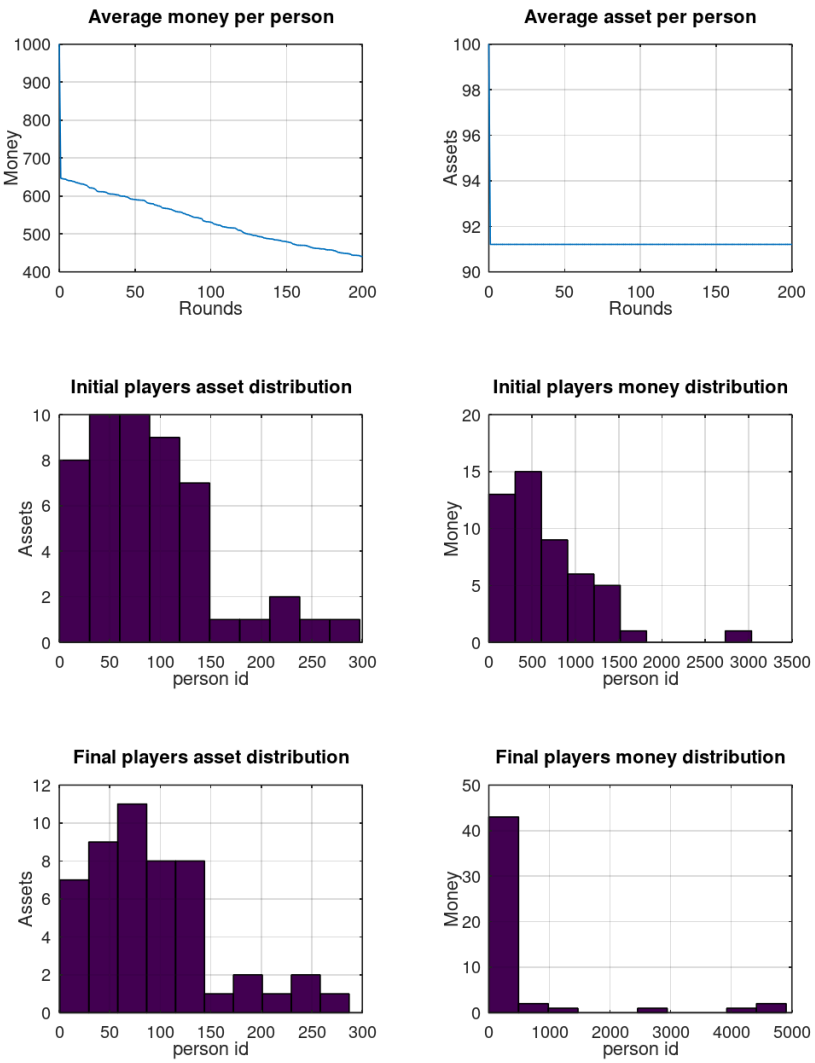

Figure 8: people stats result

- change_in_wealth

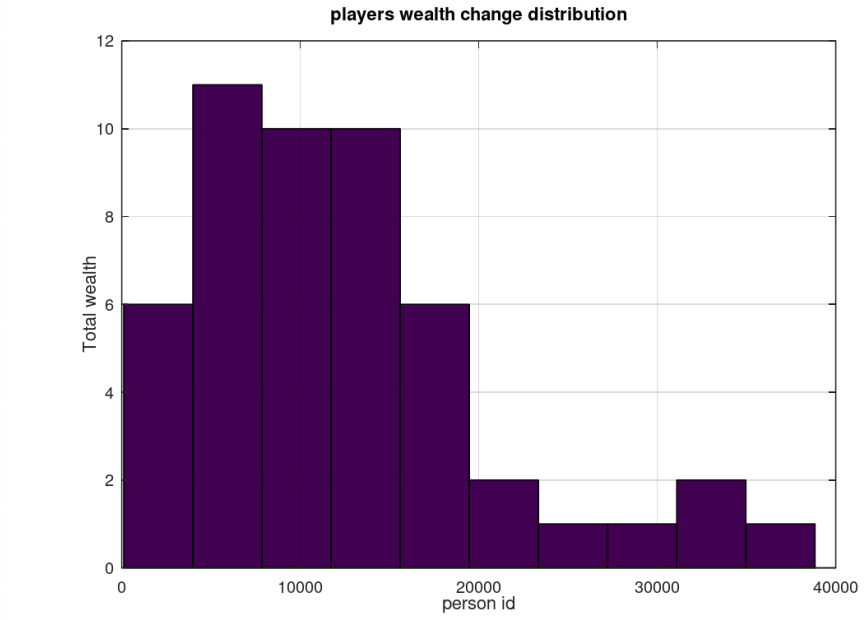

Figure 9: change in wealth result

References

[1] A Recurrent Neural Network for Game Theoretic Decision Making, *Sudeep Bhatia and Russell Golman*. Carnegie Mellon University, 2014