Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Agent-Based Modeling and Social System Simulation

Project Report

Deep Reinforcement Learning for Markets

Shanshan Huang & Qifan Guo
& Till Karbacher& Gauzelin Vidovic

Zurich
December 2019

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Shanshan Huang
Qifan Guo
Till Karbacher
Gauzelin Vidovic

Contents

1 Abstract

The aim of this project was to simulate an economic system. The said economic system is the Walrasian double auction model. The *"Decision Making Laboratory"* (*DeSciL* for short) made different double auction experiments with humans with various market settings and matching mechanism and made the data from these experiments accessible. Based on this data an agent should be trained to act like a human in a double auction environment using Deep Deterministic Policy Gradient Learning (DDPG), which is a Deep Reinforcement Learning algorithm. Out of the different settings from the experiments of the *DeSciL*, the Full information setting and the First price mechanism were chosen for this project. It turned out the datas were not convenient enough for an agent to train with it.
Therefore, an agent was trained using DDPG with no data in a market composed of one seller and one buyer. The market settings were Random match and Full information (which is the same as opposite information in the case of just one buyer and one seller).

2 Individual contributions

In this project, Till and Gauzelin works on preprocessing the dataset[1], implementing a `FirstMatch` mechanism. Shanshan and Qifan works on implementing the DDPG framework and conducting experiments.

3 Introduction and Motivations

3.1 Research context

The project aims at modeling an agent able to participate and play in a double auction game. Two type of players interact on the market, the *buyers* and the *sellers*. Each have a price limit, also called *valuation* v, respectively their budget or their production price. Agents are supposed to act rationally, meaning that they avoid losses. Bids b_i and asks a_j can be submitted at discrete time steps t and are valid for 10 time steps. For a match to happen, we must have a pair of agents at a given time for which:

$$a_j \leq b_i$$

The exact deal price $d_{j,i}$ is chosen in this range by a *matching mechanism* (more on that later). One game consists of K rounds. Each round ends after T time steps or when no more matches are possible, as agents stop playing once they got a deal. The duration T is a random variable sampled in a uniform distribution to avoid end game behaviours. At the end of a round, the environment resets and a new round starts. The same agents can then start over, with the same valuations. During a game, the *market setting* is assumed to remain constant.

This experiment is inspired by an experiment ran with human players on a large scale by the *DeSciL*. They used that environment to evaluate the effect of information feedback on agents' offers. They played games with a default market of 10 buyers and 10 sellers lasting through 10 rounds. First price matcher and black-box information setting were used. They then varied that default setting with asymmetric pull of agents, longer games, or different market settings. Results show that despite the market setting, deals realized by the players always end up converging to the competitive equilibrium, that is the equilibrium price that maximizes overall utility in a given market. More details on *DeSciL*'s results can be found in *"Draft.pdf"*[1], attached as an annex.

Our primary goal is to model an agent capable of rational behaviour in a double auction game. As a secondary objective, we would like our agent to exhibit the same human behaviour, i.e. converging towards the competitive equilibrium.

3.2 DDPG an overview

An agent in the double action system has a continuous action space or a high dimensional action space if discretized. This is a problem for a lot of reinforcement learning algorithms as for example in the Deep Q Network algorithm (DQN) since the action-value function needs to be minimized at each time steps with respect to the action space. Because of this the deep deterministic policy gradient algorithm (DDPG[2]) was used for this project. DDPG uses ideas of DQN and combines it with the actor-critic approach.

To calculate the maximal action-value function (which is the problem in continuous action space) the DDPG uses the target policy network (actor). This network computes the argument which maximises the action-value function[3]. The advantage of the off-policy part is that the exploration is independent of the learning and an exploration policy can be created through adding noise to the actor policy.
The replay buffer is used since the input data of a network should be iid (independent and identical distributed), since they were sampled from the environment this is not the case. So, a finite number of samples are stored in this buffer and for each update of the two networks a fixed number is uniformly sampled from this buffer and used as input data. The size of the buffer stays the same over all time steps and the buffer

is updated at each time step according to first in first out principe.

4 Description of the Model

4.1 Simulation setting

In order to efficiently model an agent we opted for deep deterministic policy gradient (DDPG). The modeled agent plays as an extra buyer in the default experimental environment of 10 buyers and 10 sellers over 10 rounds, using full information setting and first price matcher.
It's reward function is defined as

$$r = \begin{cases} 0 & \text{if no deal happened} \\ v - d & \text{if deal happened} \end{cases}$$

The duration of each round is directly taken from the data set.

4.2 Data set

In order to train our DDPG algorithm, we needed to use an initial data set. It seemed logical to use the results from *DeSciL*'s large-scale experience. These results are compiled in one file as a list of all bids submitted by the agents. It is important to underline the fact that only bids submissions are reported, so every agent does not necessarily have an input at every time step (indeed, they usually have less than 5 per round, roughly 200 time steps). The pseudo-code to preprocess the dataset used in the analysis are included in the appendix.

The output files we obtained this way presented many strange behaviors, ultimately leading us to look for alternatives to train our DDPG algorithm. Let's summarize all these discrepancies:

- Some agents keep submitting bids during rounds when they already got matched
- Some matches happen between a buyer and a seller with a higher bid than the said buyer. To put it otherwise, some sellers are making deals with buyers who are not paying enough for the deal to normally happen
- Some matches that should occur don't. That means that at some points in time, there are pairs of buyers and sellers for which the buyer is willing to pay enough money to the seller for a deal to happen, but for some reason none of them match

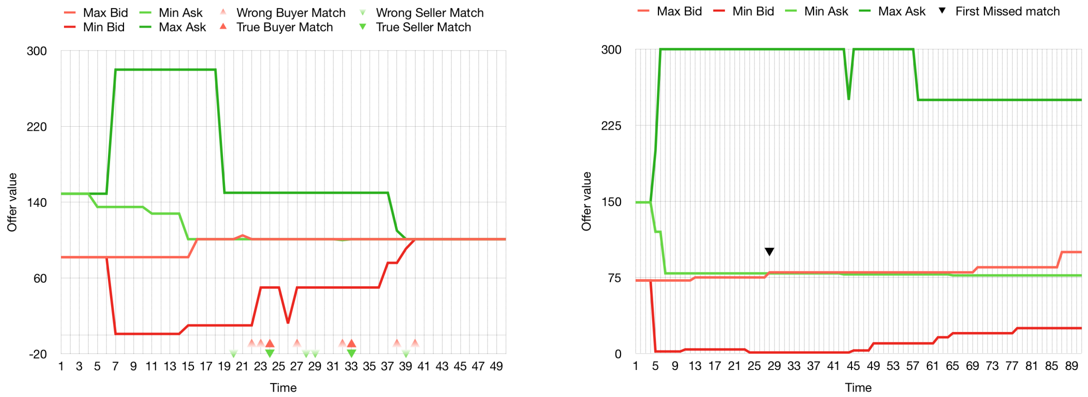
- Some one-sided matches happen. Randomly, some buyers are set as ”matched”, but no seller does at that same time step. The opposite applies too
- Some bids pending for over 10 time steps still gives corresponding agent matches. This directly violates the idea that a bid expires after 10 time steps, stated on the paper ”Draft.pdf”
- Some deal prices are negative. Yet it is defined by the research paper as the absolute price of the deal defined by the price matcher
- Although technically not a wrong behaviour, some agent will occasionally not submit a single bid over an entire round. This can get to the extreme where only 3 agents play in a given round. The uninitialized data generate a lot of noise, which confuses the DDPG during the learning phase

All of the above lead us to ultimately discard the data set and change our approach to the situation. There were too many discrepancies to consistently correct the datas. Instead, we let the agent learn by playing many games against a predetermined opposite-side actor.

Before moving on, let's quickly discuss the Figures 1a and 1b, corresponding to full information and black-box information settings data sets respectively. Each figure corresponds to one round in the given environment. In both graph, the green lines correspond to the seller side of the market, and the red lines to the buyers. The lighter curves are the the most likely offers to break a deal on both side. The whole buyers bids spectrum lives between the two red curves. Similarly the green curves embrace the sellers asks spectrum at any given point.

In the former graph, red and green arrows indicate an agent matching, respectively a buyer or a seller (their y-coordinate on the graph carries no meaning). The shaded arrows explicitly show the one-sided matches behaviour. On top of that, the minimum bid curve suggests that the buyers matching are not necessarily the one meeting the sellers asks.

The later graph shows evidence of matches not happening. Here the black arrow indicate the first match opportunity (once again, y-value is meaningless). This can be observed as the maximum bid curve rises above the minimum ask curve. However, over that whole round no match ever happens.



(a)

(b)

Figure 1: Dataset discrepancies illustration: The red and green arrows show time steps when agents are matched. Full arrows indicate a pair of agents matching simultaneously. The black arrow illustrates the first match not happening when bids and asks meet. From there on, there are mismatches until the end of the game.

5 Implementation

5.1 Experiment Details

In order to make the model insensitive to the scale of prices, all prices including valuations and proposed offer prices are normalized in the pre-processing step.

Actor is a two layer dense neural network, with only 15 neurons in the hidden layer to avoid overfitting. Both dense layers adopt Exponential Linear Unit (ELU) as the activation function. We experimented with other activations such as tanh, signmoid and relu, but in terms of convergence and stability, elu outperforms them across our experiments. Different from other activation functions, ELU has an extra alpha constant which mitigates the dead RELU problem.

Similarly, the critic is also a 2 dense layer NN. Different from actor, it takes both states and actions as the input, hence in the first layer there are two sets of weight associated with each input respectively, they are then concatenated to the 10 unit hidden layer. The activation function is again elu.

In order to prevent from gradient explosion and stabilize gradient propagation, the gradients are first clipped in range 0 to 1.

To investigate in every component of DDPG, we decided to model in a single-agent setting. As the reinforcement learning is tricky to train, we further simplified the setting in which there will be only two agents in the market, one buyer and one seller. Initially, the seller is proposing a varying price, however, the multiple varying factors in our configurations results is a extremely non-robust model. To isolate each factors, we decided to let the other agent propose a fixed price and simulate the behavior in the other side.

This is no evidence that current price depends only on the previous steps even for transparent settings (if we take the rationality of the agents into account), hence the problem could not be formulated as a Markov Decision Process. However, DDPG can only deal with transitions of consecutive time steps. To solve this problem, we add an extra parameter history length. At each time step, a fixed length of history prices are fed into DDPG network. We assume the current price only depends on given history prices. To evaluate how history length influences the accumulative rewards, we experimented with history length of 1, 10 and 50.

In the original setting, one epoch will terminate immediately if there is a deal. However, this setting doesn't behave well in our experiment. Instead, we designed to run only one epoch, in which the agent will continue even if there is a deal for a total of 5000 time steps. To find a balancing point to deal with exploration and exploitation dilemma, we added an exponentially decaying noise.

In the beginning of each epoch, we initialize the buyer with price 0.4. For the seller, it's a little bit more complicated.

$$P_s(t) = \frac{100 - t}{100}$$

for $t \in$ [0,1,2,...,history length-1]

5.2 Design of Reward Function

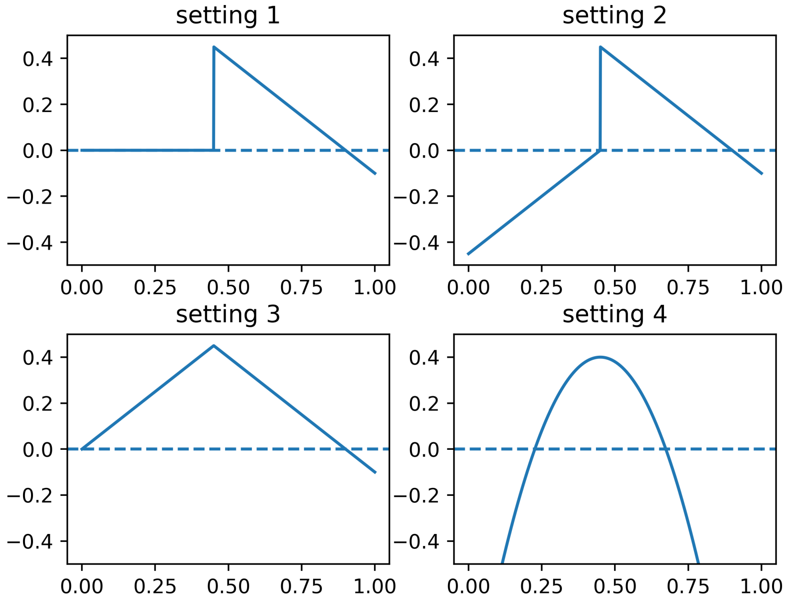


Figure 2: Different Rewards functions

The reward function implemented in the `RandomMatcher` is current offer price - deal price if they reach a deal or zero otherwise. Such reward function is prone to small errors and unstable to learn for the reason that it completely ignores the valuation price in the calculation.

We then proposed a similar rewards as a linear function as depicted in Figure 2 setting 1. The agents earns no reward when he is not matched, and earns positive rewards as valuation price - proposed price.

In the evaluation phase, the sparse rewards when there's no deal makes the agent irregardless of adding noise or not. Hence, we proposed two more as setting 2, 3 and one more quadratic rewards setting 4.

6 Simulation Results and Discussion

To evaluate a model, we sum all the discrete rewards function, which is setting 1 mentioned in section 5.2. A larger sum means a better model. We tried history length 1,10 and 50. The table 1,2,3,4 show that some reward function prefer a simple history length 1 while other reward function models prefer a history length of 10. It also shows that increasing history length does not necessarily improve the performance. In most experiments, we can see that no noise has better performance. The best result is achieved at history length of 10 with no noise in setting 3. We use a random agent as the baseline. The random agent ouputs price between 0 and 1. It has reward 482.36.

	noise=0.5	noise=0
history length 1	25.98	**579.27**
history length 10	32.72	0
history length 50	-174.42	-198.46

Table 1: setting 1

	noise=0.5	noise=0
history length 1	-235.76	-238.93
history length 10	1040.42	**1165.25**
history length 50	-160.93	-382.47

Table 2: setting 2

	noise=0.5	noise=0
history length 1	471.67	620.51
history length 10	-0.67	**1429.45**
history length 50	-163.13	-299.80

Table 3: setting 3

	noise=0.5	noise=0
history length 1	577.14	362.88
history length 10	215.07	**954.40**
history length 50	-308.92	-307.87

Table 4: setting 4

6.1 Visualization

We plot the figure of the proposed price in different settings(see Figure 3). Y-axis means proposed price and X-axis means time step. The blue line is the price of the seller. The green curve is the price output by DDPG. The red curve is the price with noise after clipping. In our case, the noise is 0 so the red curve is just the price after clipping. We also plot the behavior of the random agent. The history length is 10 and noise is 0 for each figure.

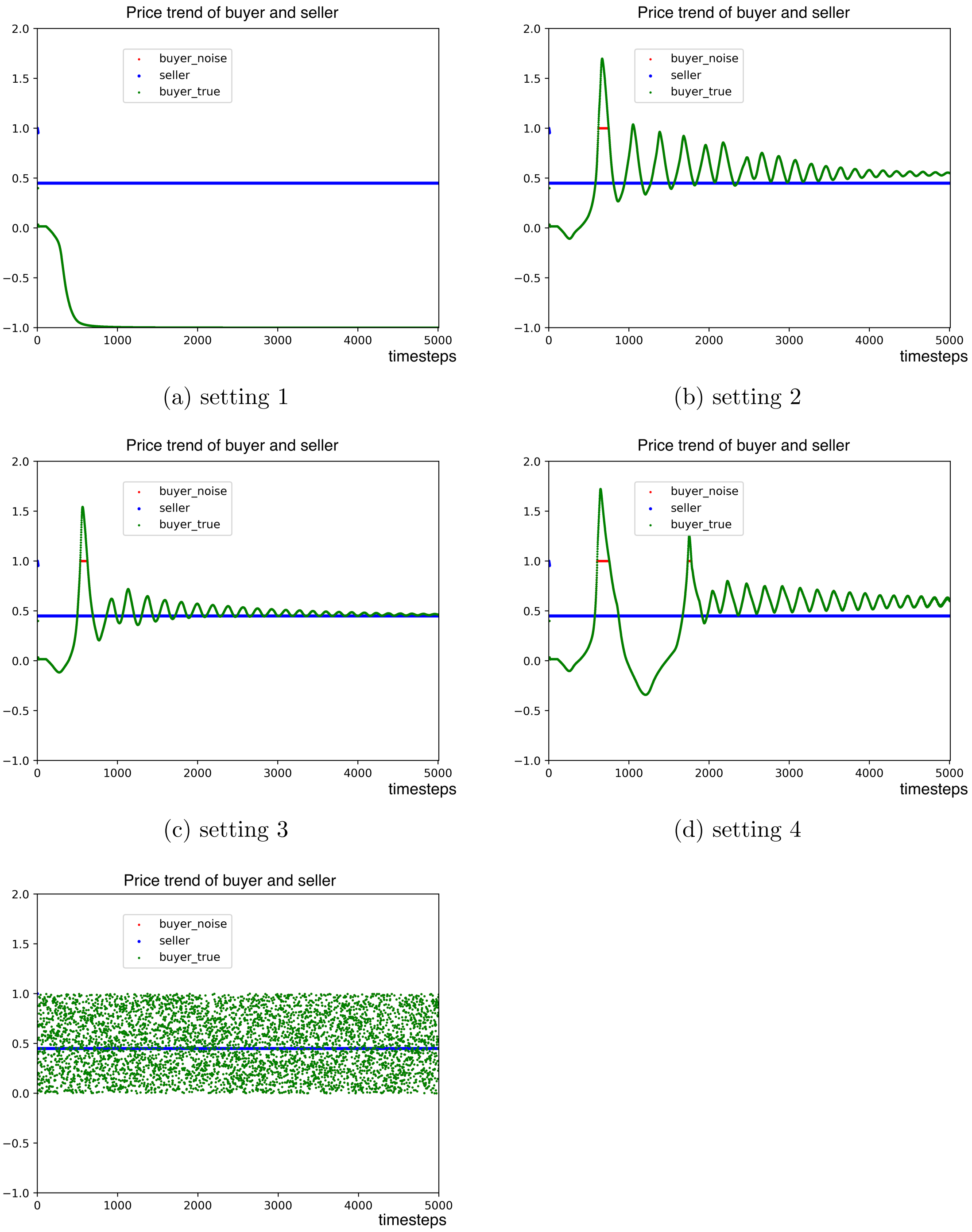
6.2 Discussion

In the discussion, we'd like to investigate various factors in our experiments.

It is interesting to see why some of the reward function fails, and others succeeds. In setting 1, the reward is zero everywhere before the seller price. Our agent earns the maximum rewards when he proposes exactly at seller's price, the reward decreases as he continues to raise his bid since the buyer is best off using least amount of money to get the product. In our experiments, the initial price of the agent doesn't seem to play an important role, but rather is determined mostly by the initial weights and bias inside both actor and critic networks. As a design choice, we initialized the weights to be small floating numbers so that it's within the valid range (0, 1). In this case, the agent is already in the zero-reward flat region. Such reward function has high risk of going out of range as the agent might choose to decrease price instead of increase. Even with the noise of variance 0.5 to start with, there's a huge chance it is trapped in the zero-reward region and ended up receiving no rewards as shown in the Table 1.

Inspired by setting 1, we proposed two similar but non-sparse rewards, which are setting 2 and 3 respectively. The only difference between the two is that in setting 2, when the agent bids less than seller price, the reward is negative for penalization; whereas in setting 3, the reward remains positive. From Figure 3 (b) (c), we generally observe the desired convergence in both setting 2 and 3. The buyer first explores the whole range of values it could take, and gradually start oscillations with less amplitude towards a convergence at the ask price from seller. Therefore, in dealing with reinforcement learning problems, the reward design is indeed a important aspect. By converting a sparse reward into a partially non-sparse one enhances learning to some extent.

We went further on investigating the influence of a non-differentiable point, which occurs in all setting 1-3 at seller's price. Setting 4 is then proposed as a smooth and continuous alternatives. To have a fair comparison with other settings, the maximum reward value is fixed at 0.4 at seller's price among all setting. The only parameter we could tune is the width of the parabolas, in other words, the range of prices which receives positive rewards. From a series of the experiments, we find it extremely tricky to tune the quadratic reward functions. If it's too wide, then the maximum reward



(a) setting 1

(b) setting 2

(c) setting 3

(d) setting 4

(e) Random Agent

Figure 3: The price trends of buyer and seller, all modelled using history length 10 and noise 0

point is not as distinct as other points, the system needs longer time to converge. In the opposite, if the range is small, then it would be similar to a spare reward function which is challenging for agents to learn. There was a considerable amount of effort going into fine-tuning the reward functions, hence we do think it is not well suited in our market setting.

Surprisingly, when we introduce a decaying noise to allow exploration, the performance is generally not as ideal as their counterparts without noise. In theory, a right amount of noise is preferred as a trade-off between exploration and exploitation. Nonetheless, in our setting, the seller is always proposing a fixed price at 0.45, and thus there is only a single global maximum at 0.45. In this case, the agent will not be stuck in local maximum, since it is the global maximum at the same time.

Another interesting results we found is that using 10 steps in history performs the best for most of reward function setting in our experiments. Prior to running experiments, we are expecting to get best results using 50 time steps. A closer look at the results Fig.3 was able to answer this question. An important step in machine learning is feature selection. Ideally the input space should contain independent features which are strong indicators of the outcome in various aspects. In our controlled setting, we only have two agents, one from each side of the market. The history information from both sides are not really 50 independent samples. The seller only proposes a fixed price, the price history of the agent himself is also dependent on each other. Moreover, most of our proposed reward functions are either linear or quadratic, the degree of freedoms are therefore significantly less than 50, hence using smaller history length are sufficient to learn and prevents overfitting for the critic networks.

7 Summary and Outlook

Through this project, we trained an agent in a Walrasian double auction game to see if it could behave similarly to a human player. We scaled the experiment environment down to a single-buyer and single-seller setting to facilitate the training.

In this setting, we were only able to evaluate the convergence of a DDPG agent. We got it to converge to a competitive equilibrium, which is a first step towards human-like behaviour compared to a random agent. However, this isn't enough to attempt to reproduce the results from the reference paper. Due to the time constraints, we are restricted to that setting. In the future, we'd like to extend our work to include interactions from the seller side of the market as well. In such case, a good reward function which considers the gains from both the buyer and the seller are also essential to simulate a real system.

Further steps would be to widen the market to more agents so more interactions could be modeled. With our current setting, only full information and black-box settings can be distinctly emulated in their simplest form, but same and opposite side information settings would be redundant.

8 References

[1] DeSciL. *Feedback effects in the experimental double auction with private information - online ressources.* URL: https://osf.io/g84bt/.

[2] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: (2015). arXiv: 1509.02971 [cs.LG].

[3] OpenAI Spinning Up. *Deep Deterministi Policy Gradient.* Accessed: 2019-11-30. URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

9 Appendix

The pseudo-code to pre-process the dataset used in the analysis is provided as following,

```
for agent $i$ do
    for game $j$ do
        for round $k$ do
            for bid $l$ do
                Bid proposal of agent $i$ with side, time $t$, setting, and deal price (if applicable)
```

as the mock dataset (Table 5) illustrates. In regards with our DDPG algorithm needs, we divided that file in sub-files, one per game and per round, structured in the following fashion,

```
for time $t$ do
    for agent $i$ do
        Current bid of agent $i$
```

ID	side	game	round	bid	time	deal price
01	buyer	01	01	100	04	
01	buyer	01	01	115	09	115
01	buyer	01	02	105	81	
...						
01	buyer	02	01	098	06	
...						
02	buyer	01	01	107	11	
...						
10	seller	01	01	120	05	
...						
20	seller	05	10	110	962	112

Table 5: mock dataset

time	ID	bid
01	01	100
01	02	107
...		
01	20	130
02	01	100
02	02	107
...		
09	01	115
09	02	107
...		
90	20	115

Table 6: formatted data set

leading to the formatted mock data set (Table 6), valid for game01, round01 only. We extended the data set to have inputs for every agents at every time step. A few things are to be noted:

- Since in the original data set agents don't necessarily submit their first bid at the first time step, the undefined bids of agents are defined as follows:

min(valuations)-1 for buyers max(valuations)+1 for sellers

This is to ensure that no possible match could occur with these initial datas.

- Once an agent is matched, its inputs are forced to 0. This value is defined as an impossible value to refrain the matching criteria to match more buyers with an already matched seller. The point of using 0 for both sides is to avoid the neural network outputs to be affected by these dummy inputs. No error will propagate through the network this way.