Faculty of Mechatronics
WARSAW UNIVERSITY OF TECHNOLOGY

Machine learning in contract bridge

Anna Sztyber*, Michał Kocon, Filip Kołodziej, Patrycja Smaga, Maciej Romaniuk
* anna.sztyber@pw.edu.pl

ML in PL Virtual Event 2020

Contract bridge

Bridge is a card game played by two pairs. It consists of two phases:

- Bidding - during this phase players can use only predefined bids (like 1 ♥ or 2 ♠) or announcements (pas, double, redouble). The goal of bidding is to find declarer, trump suit, and contract level.
- Declarer's play - in this phase declarer tries to make contract collecting as many tricks as possible.

Despite recent developments of AI in games, in bridge human players are still superior. Bridge beauty and difficulty lies in need of cooperation between partners based on incomplete information. The goal of this poster is to present a series of our recent projects investigating machine learning application in contract bridge.

Hand vectorization

Introduction

Bridge hand is a set of 13 card held by one player.
The main goal of the following part was to achieve a vectorized bridge hand, which could deliver more information than considering single card itself. Desired form is expected to allow for comparison between different hands taking into account their power, suit etc. The idea was defined as machine learning problem and an attempt to solve one based on deep artificial neural networks. Inspiration comes from natural language processing technique - word embedding, which consists in mapping words into numbers. Both approaches enable to treat non-numeric data as mathematical structures.

$$\begin{bmatrix} 2♣ & 3♣ & \cdots & Q♠ & K♠ & A♠ \\ 0 & 0 & \cdots & 1 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1.03 \\ 0.74 \\ \vdots \\ -2.54 \end{bmatrix}$$

Bridge solver implementation

It has been already proven that computer can do better at logic games than human on the example of chess. Appearance of other bots playing even more complicated games seems to be just a matter of time. Currently, there is no fully developed system, which plays contract bridge at master level. The reason why it causes so much trouble is the fact, that bridge is a game of incomplete information, which means players are not fully aware of the cards distribution between others.
Since contract bridge remains unsolved, there is a simplified version with cards open on the table. From this point of view, players are assumed to be familiarized with all cards, so partial information problem disappears. On this basis, programs called double dummy solvers (DDS) can compute, how many tricks each pair is able to collect depending on trump, but it takes some time, which AI techniques cannot afford.

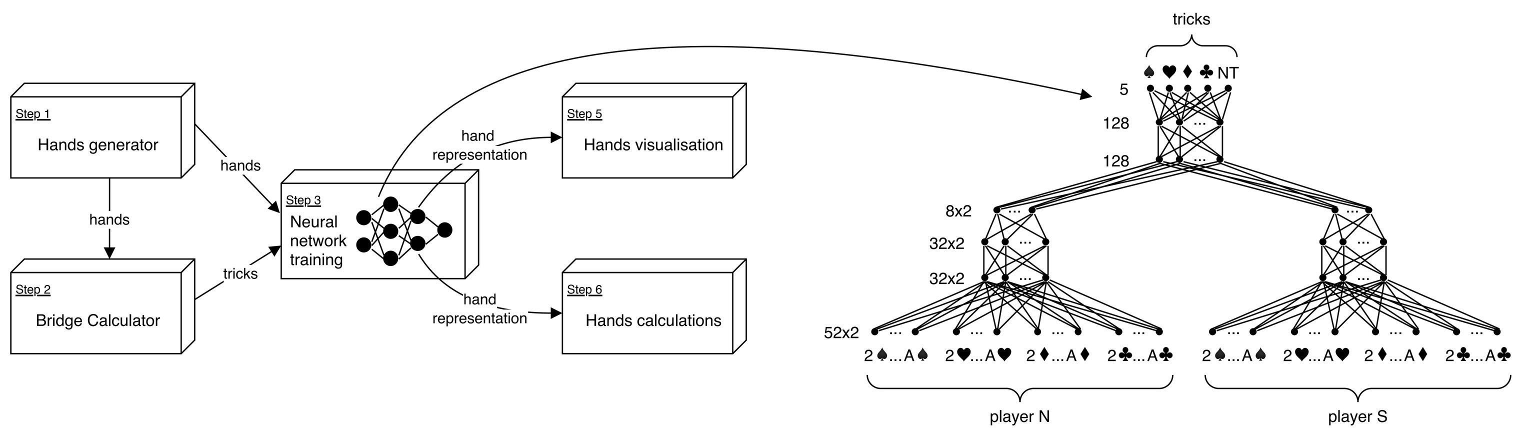

Fig. 1: Diagram of the system with data flow and neural network architecture

Trick error	0.5	1	2
All	79.0%	97.5%	99.9%
Suit	82.6%	98.8%	100.0%
No-trump	64.5%	91.2%	99.5%

Tab. 1: Neural network accuracy in specified contracts and error

Figure 1 presents a system, which overcomes two problems mentioned above (on certain level) - incomplete information aspect and long computing time. The core constitutes an artificial neural network (MLP type), which is trained to predict tricks like DDS, but based only on one pair of player's hands. Training data was generated using two external programs. First one [1] drew hands for N-S pair and then ten sets for E-W pair taking into account, that N-S cards have been already used. Second program was classic DDS - Bridge Calculator (`http://bcalc.w8.pl/`), which computed ten deals for each N-S hands configuration. The aim of the additional E-W hands was to average score as an analogy to Monte Carlo algorithm. The tricks obtained could be used as labels for training. After successful learning process, neural network was able to compute deals much faster than classic DDS with accuracy given in table 1.

Hand representation

Hand representation (as 8 real numbers) was obtained from layer 8x2 presented in figure 1, which was being reduced until accuracy didn't deteriorate significantly. To check if given form of hand is intuitive, further reduction to two dimensional vector has been applied using t-SNE algorithm. Then it became possible to visualize hands in 2D space - plane. Figure 2 (left) presents 5000 hands coloured in a following manner: the more blue point is the more high card points the hand contains.
It easy to notice, that strong hands are presented by points in the upper part of the plane. On the other hand, second visualisation depicts groups of hands depending on majority of cards in certain colour. Hands dominated by spades have a tendency to be to the left, hearts - to the right, clubs and diamonds - more and less centrally. Summing up, the vectorized bridge hand can be assumed logical.

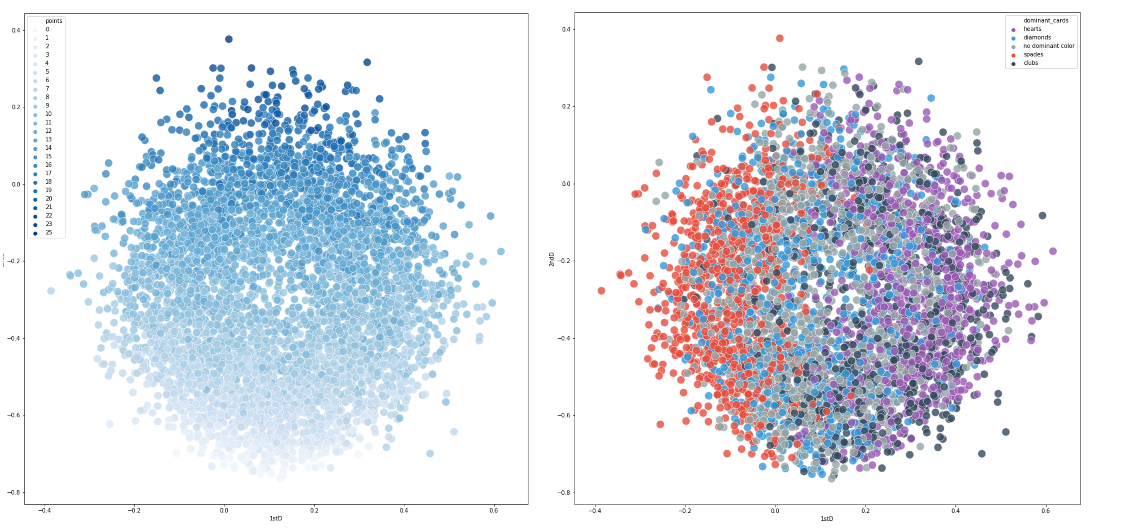

Fig. 2: Visualisation of 5000 bridge hands using t-SNE algorithm

Hands as mathematical structures allow for calculations on them. Table in Fig. 3 presents five most similar hands (from 100000 test set) to the one in the first row, which can be defined by user. Example in Fig. 4 involves basic operations on vectors. As an analogy to famous word embedding equation $king - man + woman = queen$, an attempt was made to solve $H_{strong} - H_{weak} + S_{weak} = x$. It turns out, that $x = S_{strong}$.

	Label	Points summary	Points in trump	Cards in trump
Base	32 KJ2 Q8764 KT9	9	[0, 4, 2, 3]	[2, 3, 5, 3]
1	97 KT4 J9763 KJ9	8	[0, 3, 1, 4]	[2, 3, 5, 3]
2	52 KT3 J9853 KJ5	8	[0, 3, 1, 4]	[2, 3, 5, 3]
3	J2 KT4 Q8764 K73	9	[1, 3, 2, 3]	[2, 3, 5, 3]
4	64 KT2 QT987 K53	8	[0, 3, 2, 3]	[2, 3, 5, 3]
5	82 KQ8 Q7654 KJ9	11	[0, 5, 2, 4]	[2, 3, 5, 3]

Fig. 3: Hands calculations

	Label	Points summary	Points in trump	Cards in trump
Strong Hearts	432 A432 432 432	4	[0, 4, 0, 0]	[3, 4, 3, 3]
Weak Hearts	432 J432 432 432	1	[0, 1, 0, 0]	[3, 4, 3, 3]
Weak Spades	J432 432 432 432	1	[1, 0, 0, 0]	[4, 3, 3, 3]
1	A654 T43 975 642	4	[4, 0, 0, 0]	[4, 3, 3, 3]
2	A764 982 953 876	4	[4, 0, 0, 0]	[4, 3, 3, 3]
3	A643 842 752 854	4	[4, 0, 0, 0]	[4, 3, 3, 3]

Fig. 4: Hands calculations

Application of reinforcement learning to bridge bidding

Problem description

In case of standard problems solved by reinforcement learning, a single agent is being trained thanks to experience gained by interaction with environment. When agent performs an action, he gets "rewarded" by the environment. The reward for same action performed by agent being in a certain state should be predictable with a certain possibility (stochastic environment). In case of bridge bidding we do not possess this kind of luxury. States are described by series of previous bids of which only a half of has been directly chosen by the agent in question. The other half has been chosen by another agent whose policy is unknown. Moreover during the learning phase, policies of agents are not constant and change with every learning step. It means that the environment changes and old experiences are not necessarily usable in the future.

Algorithm and network structure

In standard Q-learning implementations, Q-values cascade down the network with every step and are calculated as a sum of reward in current step and discounted expected highest Q-value in next step. During bridge bidding we do not gain any rewards until the phase is finished and the game is concluded. Therefore the only ingredient remaining for calculation of Q-values is the expected Q-value in next step. To speed up the process algorithm of "penetrative Q-learning" was proposed in [2]. Instead of waiting for Q-values to cascade down the network, the reward gained by ultimately chosen bid is used as a reward for every action performed on every step of bidding phase. Although it may seem counterintuitive (agent which chooses irrational bidding path will get same reward as if he chose the most optimal path), it proved to deliver interesting results.

- $Q^*(s,a)$ - optimal Q-value achieveable by choosing action a in state s.
- $Q^*(s',a') \mid s,a$ - optimal Q-value achieveable by choosing action a' in state s', which is achieved by choice of action a in state s.
- r - reward gained by choice of action a in state s.
- γ - discount coefficient

$$Q^*(s,a) = r + \gamma \cdot \max_{a'} Q^*(s',a') \mid s,a$$

$$Q^{(i)*}(s,a) = \max_{a'} Q^{(i+1)*}(s',a') \mid s,a$$

$$Q^{(i)*}(s,a) = Q^{(i+1)*}(s',a^{*\prime})$$

$$Q^{(i+1)*}(s',a^{*\prime}) = Q^{(i+2)*}(s'',a^{*\prime\prime}) \mid s',a'$$

$$Q^{(i)*}(s,a) = r(a^{*(t)})$$

The structure of the implemented network has been based on the architecture described in [2]. Players cards were represented by a vector consisting of 52 bits, each representing the presence of one card. Appended to it was current bidding history, encoded in similar manner (bids marked by 1 were called already). During each phase of bidding, networks have checked the end result of every action (36 possible actions in the beginning). Their bid was being propagated through the system with greedy exploration policy. After the end result of every possible move has been calculated, "player 1" chose his bid using UCB1 exploration algorithm and "player 2" (armed in information about his own cards and previous bid) repeats same steps.

Rewards and results.

The rewards were calculated based on the contract bridge IMP scale and normalized between 0 to 1. Three experiments were performed. In the first one (which served as a baseline), each layer of a system consisting of 4 networks (two players without rivals participating place up to 2 bids for a total of 4) were trained separately as if their bid was a final one. In second experiment, reinforcement learning was used and in third, reinforcement learning with recurrent networks. Mean reward in the first experiment was equal around 0.891, in second 0.905 and in third 0.9.

	Separate learning layer by layer					Reinforcement learning					Reinforcement learning RNN				
	♠	♥	♦	♣	Times	♠	♥	♦	♣	Times	♠	♥	♦	♣	Times
1♣	1.89	2.01	2.17	4.85	3373	3.28	3.16	3.04	3.16	8319	2.02	2.51	2.70	4.67	3175
1♦	1.86	2.08	4.76	2.17	4341	1.91	1.64	5.59	2.09	1013	3.92	2.91	3.17	2.59	6487
1♥	2.01	4.46	2.25	2.26	5397	2.13	5.64	2.69	2.67	1847	2.01	5.46	2.86	2.79	2137
1♠	4.25	2.12	2.25	2.35	5112	6.08	2.67	2.69	3.09	853	6.07	3.10	2.83	3.36	1244
1NT	3.39	3.37	4.01	3.48	973	4.62	4.70	4.57	4.56	804	5.44	4.99	5.07	4.98	286
2♣	0.0	3.0	0.0	8.0	1	2.79	1.25	4.64	6.75	48	3.03	3.90	4.40	6.02	333
2♦	5.0	2.2	7.2	1.6	20	4.85	7.14	2.59	2.85	42	3.14	4.25	6.46	3.52	307
2♥	-	-	-	-		2.65	7.05	3.71	5.21	126	3.06	6.99	4.15	3.66	164
2♠	-	-	-	-		7.14	3.54	5.11	4.42	70	7.76	4.8	4.6	3.24	25
2NT	-	-	-	-		6.42	7.71	2.42	4.57	7	6.33	7.0	7.0	5.33	3
3♣	-	-	-	-		-	-	-	-		5.28	6.07	4.92	7.28	14
3♦	-	-	-	-		7.0	10.0	6.0	7.0	1	4.0	4.0	9.5	1.5	2
3♥	-	-	-	-		5.94	5.82	2.24	2.11	70	5.2	6.8	5.32	5.2	25
3♠	-	-	-	-		7.30	5.23	2.15	3.61	13	6.6	3.0	6.0	6.4	5
3NT	4.36	4.29	5.07	4.73	1248	5.75	5.14	6.09	5.11	114	4.0	5.0	5.33	8.33	3

Tab. 2: Hand strength comparison (High Card Points) for the opening bid [1]

Player N	♠8632 ♥AQ532 ♦9 ♣K96			
HCP of player N	♠:0	♥:6	♦:0	♣:3
Player S	♠AKQT ♥ ♦KQJ532 ♣QT4			
HCP of layer S	♠:9	♥:0	♦:6	♣:2
Bidding history (N starts)	1♣-2♦-3♦-3NT			
Highest possible bid	4NT			
Player N	♠AKQJ95 ♥AK ♦AT6 ♣KJ			
HCP of player N	♠:10	♥:7	♦:4	♣:4
Player S	♠- ♥JT652 ♦8543 ♣AT54			
HCP of layer S	♠:0	♥:1	♦:0	♣:4
Bidding history (N starts)	2♠-3NT			
Highest possible bid	6NT			
Player N	♠Q3 ♥KT543 ♦J876 ♣K2			
HCP of player N	♠:2	♥:3	♦:1	♣:3
Player S	♠AJ9764 ♥A7 ♦AT ♣AJ6			
HCP of layer S	♠:5	♥:4	♦:4	♣:4
Bidding history (N starts)	Pas-2♣-3♦-3NT			
Highest possible bid	5NT			

Tab. 3: Bidding sequences performed by system trained with reinforcement learning. Visible tendency to avoid risky bids [1]

OpenAI Gym bridge bidding environment

`https://github.com/patrysma/gym-bridge-auction`

Environment description

A large number of algorithms needs testing to solve the problem of bridge bidding using reinforcement learning. In order to compare them a multi-agent environment simulating the bidding was created in Python. The implemented environment has the following features:

- The environment is designed for four agents.
- The requirement was to use the OpenAI Gym library interface, which is an open source toolkit for developing and comparing reinforcement learning algorithms (`https://gym.openai.com/docs/`).
- The environment has both a graphical (Fig. 5) and console interface. The graphics option is well suited for demonstration purposes. However, for the algorithms development it is better to use the console version, which works faster.
- The bidding was simulated according to the rules of bridge. The action space includes bids ([7NT, 7♠, 7♥, 7♦, 7♣, ... , 1NT, 1♠, 7♥, 1♦, 1♣]) and announcements such as: pass, double and redouble.
- The environment is to be used for future development and comparison of reinforcement learning algorithms, thanks to the appropriate definition of the agent's action space, the environment state after the action taken and the reward function.

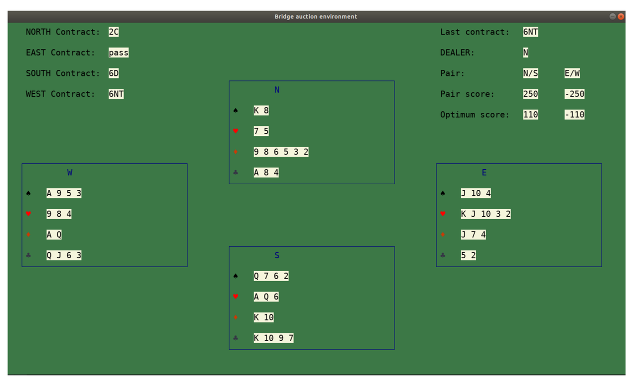

Fig. 5: Graphical interface of the bridge bidding environment [3]

- Double Dummy Solver (DDS) tool (`https://github.com/dds-bridge/dds`), that offers various types of bridge calculations, was useful in determining the reward function. The following functions were used: calculating the maximum number of tricks each player with a partner would take and determining the optimal score for pairs, assuming that all players bid perfectly.

Environment in action

OpenAI Gym provides a simple environment interface that allows the agent to interact with the environment. The following code (Listing 1) shows the use of the environment with random agent actions without implementing any training system.

Listing 1: Sample program code presenting the environment with console interface in action[?]

```
import gym_bridge_auction
import gym

PLAYERS_NAMES = ['N', 'E', 'S', 'W']

for i_episode in range(5):
    env = gym.make('BridgeAuction-v0')
    observation = env.reset()
    hands = {}
    for number, player in enumerate(PLAYERS_NAMES):
        hands[player] = observation['Players hands'][number]

    env.render('console')
    print('Observation space:')
    print(observation)

    for i in range(100):
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        env.render('console')
        print('Observation space:')
        print(observation)
        print('Reward: ' + str(reward))

        if done:
            print("Episode {} finished after {} timesteps".format(i_episode + 1,
                  i + 1))
            break

env.close()
```

Listing 2: Bidding fragment - first two bids[?]

```
Dealer: W
N hand:     E hand:       S hand:         W hand:
♠ 9 7       ♠ K Q 8 4 3   ♠ J             ♠ A 10 6 5 2
♥ Q 9 6 3   ♥ K J 10      ♥ A 8 7 2       ♥ 5 4
♦ 10 9 3    ♦ J 6         ♦ A 5 2         ♦ K Q 8 7 4
♣ A 8 7 5   ♣ Q 6 4       ♣ K 10 9 3 2    ♣ J

Observation space:
{'whose turn': None, 'whose next turn': 3, 'LAST_contract': None, 'Player_contract': None, 'winning_pair':
    None, 'double/redouble': 0, 'Players hand': [[0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
    1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0], [0, 0, 1,
     0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
    0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0], [1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
     0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0,
    0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1]]}

LAST_contract: 3NT              NORTH_contract: None
Pair: N/S E/W                   EAST_contract: None
Score: 100 -100                 SOUTH_contract: None
Optimum score: 420 -420         WEST_contract: 3NT

Observation space:
{'whose turn': 3, 'whose next turn': 0, 'LAST_contract': 21, 'double/redouble': 0, 'Player_contract': 21,
'winning_pair': 1}
Reward: [-320, 320]

LAST_contract: 6NT              NORTH_contract: 6NT
Pair: N/S E/W                   EAST_contract: None
Score: -300 300                 SOUTH_contract: None
Optimum score: 420 -420         WEST_contract: 3NT

Observation space:
{'whose turn': 0, 'whose next turn': 1, 'LAST_contract': 6, 'double/redouble': 0, 'Player_contract': 6,
'winning_pair': 0}
Reward: [-720, 720]
```

The bidding fragment - the first two bids - is presented in Listing 2. Agents in clock order (the dealer starts, in this case W) perform single actions selected from the available space. The full set of information about the game state is not available, but agents have access to the bidding history thanks to the `observation` variable, which is visible to all players. Additionally, they can still see their own cards. The players actions are evaluated by means of a reward that measures the effectiveness of the bidding. In each step, the difference from the ideal case is returned (e.g. `Reward:[-320, 320]`, where -320 is the difference for the N-S pair and the second value is for the E-W pair). Information about the pair's score at a specific time in the bidding phase and the optimal score for each side is also displayed. This data can be obtained from the `info` variable and can be used by developers to check the results. The goal of each episode is to establish a contract, which is a commitment that the winning pair will take a certain number of tricks (state `winning_pair`).

The speed of the environment depends on the time taken for the Double Dummy Solver to return results useful in determining the reward function. It is varied, most often it ranges from 0.4 to 0.8 seconds, depending on the distribution of players' hands and the number of possible playing options for a given player. The environment code was placed in the GitHub repository `https://github.com/patrysma/gym-bridge-auction`.

OpenAI Gym bridge play environment

`https://github.com/KylarStern7/gym-bridge`

Introduction

Conract bridge hasn't been yet dominated by AI players. Main reason is that bridge is a game of incomplete information which makes it very difficult to find optimal policy for reinforcement agents. Also the number of possible game states is huge starting with 5.36×10^{28} different card deals. In order to find best machine learning algorithm there is a need to compare many of them that is the purpose the environment simulating bridge play has been created.

Environment features

- The environment has been created to be used by three agents (declarer and two defenders).
- The environment has a modified OpenAI GYM library interface which allows multi agent scenario.
- The environment has text and graphical interface (Fig 6).
- Agents receives all necessary information about current game state: agent's and dummy's cards, history of all previously played tricks, etc.
- Agents can receive rewards in one of four modes depending on the purpose of agent's learning.
- The initial conditions of game can be set randomly or loaded by user. Loading real data can improvement agent's learning effectiveness.

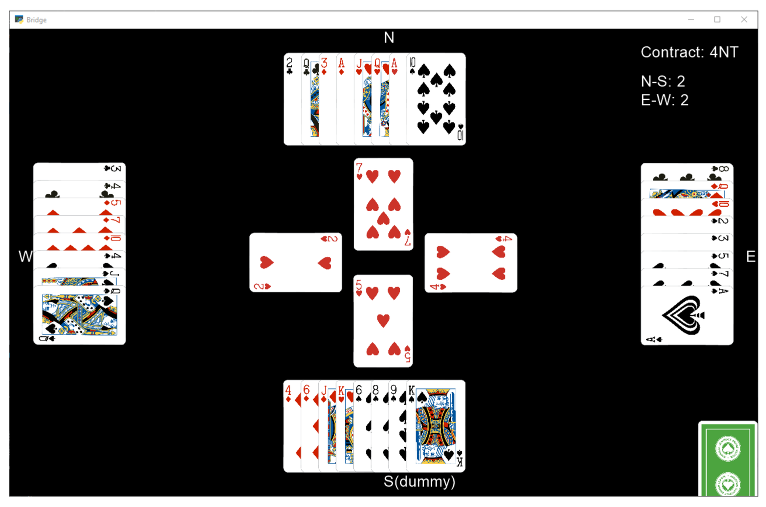

Fig. 6: Environment graphical interface [4]

Environment test

Environment has been tested in simple scenario with reward mode designed to learn basics of bridge. Agent received reward of value 1 each time he chose an action (card) that was complied with the rules otherwise agent got reward of value -2. Agent used only part of available observation containing his cards (vector of length 52) and suit of first card played in trick (vector of length 4). The Deep Q-Network Agent with fully connected neural network has been implemented (Fig. 7).

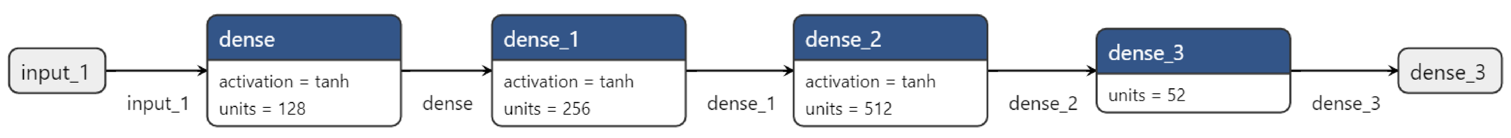

Fig. 7: Agent's Q-Network architecture

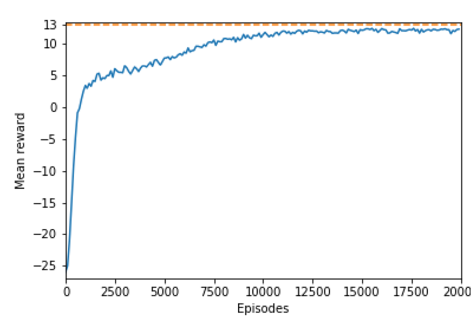

Fig. 8: Avarage reward during training [4]

During training agent has played 20000 episodes with initial conditions generated randomly. Agent has been collecting data at each trick and fitting his Q-Network at the end of each episode. Figure 8 presents average reward received by agent during training. At each point average has been calculated from 100 previous episodes.
Trained agent was evaluated on 1000 episodes with an average episode reward of 12.256 out of a maximum of 13. Table 4 presents percentage of correct actions taken by agent depending on trick number. The environment is available on github `https://github.com/KylarStern7/gym-bridge`.

Trick	1	2	3	4	5	6	7	8	9	10	11	12	13
Proper action [%]	97.6	92.8	91.0	95.2	93.7	93.1	94.6	93.1	95.5	95.5	93.7	93.7	96.1

Tab. 4: Percent of proper actions taken by agent

References

[1] Romaniuk M. Reinforcement learning of cooperative agents in partial information system. Master's thesis, Warsaw University of Technology, 2020.

[2] C. Yeh, C. Hsieh, and H. Lin. Automatic bridge bidding using deep reinforcement learning. *IEEE Transactions on Games*, 10(4):365–377, 2018.

[3] Smaga P. Creating a multi-agent environment using the gym library. Master's thesis, Warsaw University of Technology, 2020.

[4] Kocon M. Implementation of reinforcement learning environment. Master's thesis, Warsaw University of Technology, 2020.