Final Exam APML - 67750

Amir Bar 200864627

Lior Ziv 305742611

Board representation:

We have considered multiple representations of the board as inputs for the models:

- The board as given, 6 by 7 matrix each cell contains 1 for player 1, 2 for player 2 and otherwise 0. At first we thought this will give the learner all the information it needs in order to understand Q* policy, but it turned out to be too simple model which make it harder to our learner to separate between his moves to the rival moves, therefore the winning percentage wasn't satisfying (~60%) against the random player.

- Our observation in the previous representation led us to try and help the learner distinguish between the players. Therefore, we separated between our board and the opponent's board by sending two concatenated matrices of size 6 by 7 containing only one player moves (represented as ones). This approach gave us better winning rate (~70%) but still not what we hoped for against the random player.

- As we have seen that giving a plain representation of the board does not yield great results we decided to represent the board in action type of manner. That is, instead of giving the board, we represent the board as the effect of a certain action. For that, we represented the state as two vectors of size 7 (one for each possible action). As a first attempt we marked in the vectors by one the winning actions and the blocking actions. This representation reach about ~80% win percentage against the random player and 60% against the MinMax algorithm with depth 1. This representation, is the one we finally used.

- By now we viewed some game replays and we have seen that our learner approximately plays as follows - block or win if you can otherwise put a disc in some column. Thus, we turned to look for a representation that would encourage the learner to develop a strategy when it does not need to block or win. To do so, we thought of multiple properties. First, we looked at each possible action proximity, e.g. we computed for each possible action the amount of connections it would yield. This would encourage the learner to diverge from column filling and create connected bulks that would later become opportunities. Another property we tried is "threat map", a 6 by 7 matrix where each cell contains the maximal potential sequence for a player and if an opponent occupies the cell. However, both properties did not seem to improve the learner.

We also tried combinations of those representations that include the winning and blocking moves vector however, they also did not show better results.

Learned Model and Reasons Behind:

We considered multiple topologies to process the different board representations but as we decided on the 2 by 7 vector representation of winning/blocking moves we could simply use a multi-layer perceptron (MLP) with 2 fully connected (FC) hidden layers with ReLU and a read-out layer of size the actions corresponding to each $Q(s_t, a_i)$. This representation should be able to be learned with such a model and there is no reason to use more complex network for it. Please see the other solutions section for other topologies for other representations we considered.

Learning Algorithm chosen:

We had a long discussion on what our model of learning should be. After we went over the lectures and the exam limitations, we came to conclusion that we won't be able to use an implementation that might explode with the number of states, take too much time to act or learn. Guided by those, we decided to implement the deep-Q learning model that use a neural network to approximate Q and showed relatively good results in pong with a little amount of time. The Deep-Q-Learning model is based on using gradient descent in order to minimize δ_{s_t,a_t} which is composed of :

- The Q function at the previous state and action.
- The reward for the previous action plus the optimal Q value for the new state with some action times the discounted reward constant.

$$\delta_{s_t,a_t}(Q) = Q(s_t, a_t) - \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') \right)$$

By minimizing $\delta_{s_t,a_t}{}^2$ we are minimizing the gap between our approximation for Q to Q* and we converge to Bellman's equation which means we are improving our policy.

In the exam we get as an input the previous state, new state, action and reward which are all the needed parameters for our chosen model. We implemented a memory replay database to allow ourselves at the learning phase to extract random batches for learning.

Exploration-exploitation trade-off:

We first distinguish between two cases:

1. Training time.
2. Testing time.

In the first case, we want at first to have high values of randomness as our policy is not trained and we want it to see as much cases as possible. Later on as we improve, we want to be less random however, we still do want some chances to random moves. To achieve this, we chose to start completely at random defining epsilon 1 and used decay of 0.9999 (approximately 0.9 factor for every 1000 games) to a minimum of 0.05 (which is expected to be one random move every two games for a random board initializer). This way we ensure to see many random games before we act according our policy (1 to 4 random/policy ratio). At the second case, we

assume we have a winning policy and want to act according to it and thus we set zero probability for randomness.

A short description of the tests and results you got when you trained your policy

We trained our data with different parameters and by the end decided to stick with the submitted once :
Batch size - 200 (decreases by 10 per learning delay)
Number of batches - 10
Learning rate - 0.0001
Initial epsilon for training - 1
Gamma - 0.9999
Our test results were:
~80% against the random player
~60% against the Maxmin with depth 1
~8% against the Maxmin with depth 2

Other Solutions We Did Not Submit:

As suggested previously, we considered many board representations and topologies networks. Each network we considered we tested multiple layer sizes depending on the inputs. Here we represent the notable models that we did not hand:

1. A CNN with single convolutional layer of 3 by 3 patches and 16 channels followed by 2 FC layers with ReLU and a read-out layer of size of the actions. We used this model when we represented complete boards (6 by 7) either in the state we were given or in the separated representation. The idea was that in the separated representation the convolutional network with patches of size 3 by 3 could recognize columns, diagonals or rows that might be a threat or a win.

2. Same as the previous but with a residual network structure where the output of the CNN and the original input is going through 2 FC layers with ReLU and a read-out layer of size of the actions. This did not show improvement, at least with the amount of rounds we used.

3. Two MLPs as in item 1 with a 2 FC layers over their concatenated output and a read-out layer of size of the actions. This used us to combine the new properties mentioned in the board representation. The idea was that we might want to reach a conclusion based on each “hint” independently and then weight the ratio between them. This should not be worse as the model can ignore one of the networks. As it didn’t show improvement, most likely due to low round count, we abandoned this implementation.

Another issue we did not get to submit is to add drop in learning rate at the late stages to fine tune our model.