Answers

January 27, 2018

1 Answers - Exam

Name: Katya Chirko
Id: 321273351
cs: katyac

1.0.1 Board Representation:

I chose to represent the board as a vector with length 84, that consists of two concatenated vectors:

The first half is the flattened game board with 100 in slots with the current player's pieces, and the second half is the flattened game board with -100 in slots with the other player's pieces.

I chose this representation because my learning algorithm is a three-layered fully connected neural net, and a flattened vector is the most suitable input for it.

I wanted to distinguish between out player's pieces and the other player's pieces, hence the 100 and -100 marks. At first I wanted to pass a vector with length 42 as input with the same marks for the players, but ampirically this representation showed poorer results:

In a game between agents with those different representations, the 42-long vector started with a lead of 55%, but after 3000 iterations (in training mode) it's performance decayed to 38%. This representation also stops improving when playing against the school's agents after a while.

1.0.2 The Learning Model:

The learning algorithm is a three-layered fully connected neural net, with output of length 7 which is supposed to represent Q(a,s) for each action a, when the state s is given as an input. The output of the nn is multiplied pointwise with a boolean vector of tength 7, when v[i] = 1 iff i is a legal action. The maximum of this multiplication is returned as the chosen action.

At first, my intention was to implement a convolutional network and pass the gameboard to it as a matrix. The idea was that a convNet could learn useful patterns on the board, however the performance of the convNet was barely better than the performance of the linear classifier I implemented just to make sure something worked, so I decided to turn the linear classifier into a multilayered netword.

1.0.3 The Algorithm:

The model I chose to submit implements the deep-Q algorithm.

It was obvious that the model should be learning while playing, and between the different algorithms we learned deep-Q seemed like the most suitable for the specifications of the problem:

on evaluation the models will be trained large amount of episodes, but we have little time for each. The Actor-Critic and the deep-P algorithms converge faster but take more time for each iteration.

I also added the success-learning element - the network only learns the states from games that the agent won. Such agent wins 70% of the time against an agent with same parameters that learns from all games, and also the agent that learns from all the games stops improving at some point.

I also implemented the deep-P algorithm, and indeed it taked more time to act and learn because it needs to evaluate both the Q net and P net, but it doesn't seem to perform better than the deep-Q agent although the network has the same parameters:

After 10000 learning episodes, deepQ wins 70% of the games against deepP.

1.0.4 Exploration - Explatation:

This is an epsilon-greedy algorithm. The epsilon starts with 0.1 and decays linearly w.r.t. the number of rounds and the game duration, until it reaches 0.05 and stays there - in order to avoid local maxima after large amount of rounds.

1.0.5 Results:

A short descriptions of the results against the models I chose not to submit are found in the sections above.

Results against school's agents: The submitted model reaches 30% sucess against Minmax agent with depth = 2 during a 10,000 episodes long training session. The results of this model are:

- 18% success against Minmax agent with depth = 2
- 55% success against Minmax agent with depth = 1
- 80% success against Random agent

I think that the test results against the minimax with depth 2 agent are worse than the training results because the minimax agent has more time during the test.

The results of deepP agent while training against the Minmax agent with depth = 2 are 13% success, and it doesn't seem to improve with time. I understand that the results of deepP should be better than deepQ, but I chose to improve the model which initially gave better results.

Remark: At first, I used the relu activation function which gave much better results - about 18% success against the minimax with depth 2 after 100 episodes, but the success percentage decayed with time. I found out that the relu function "killed" all the neurons after several iterations and with time all the output neurons became 0. Instead the relu, I used tanh activation function, which gave worse initial results (about 13% success against the minimax with depth 2 after), but now the network is improving with time.