Report for the Assignment: Implementing a Machine Learning Model to learn how to play the game Othello

Vladimir Ilievski

April 10, 2017

1 Introduction

The task of this assignment was to implement a Machine Learning model to learn how to play the game Othello. That means, the solution includes an intermix of the classical supervised and reinforcement learning methods.

Since playing of a game consists of an intelligent agent acting in an environment such that its performance is measured by the accumulated reward, there are few methods to maximize the agent's reward. In this task, I decided to use the off-policy Q-learning following an epsilon-greedy strategy, which is maximizing the expected reward. As a function approximator, I decided to use a Multilayer Percepton.

2 Theory

The Q-function is a function of a state-action pair, such that its approximation is expressed by one of the Bellman equations:

$$Q^{new}(s_t, a_t) \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a). \tag{1}$$

That means, the input of the Multilayer Percepton will be the current state s_t, and the output will be the value of the Q-function for every possible action in that state. However, the nature of the data in the Reinforcement Learning tasks is sequential, i.e. we don't know the next input, until an action is taken. On the other side, the Neural Networks work when all of the data is known a priori. For this reason, we have to adapt the Neural Network's way of working.

First, we observe a state s_t and the Neural Network will calculate $Q(s_t, a_t)$ for all possible actions a_t. Then, according to the epsilon-greedy policy it will chose and execute an action. After that, using the Bellman equation for the previous state-action pair we will calculate $Q^{new}(s_{t-1}, a_{t-1}) = r_{t-1} + \gamma \max_a Q(s_t, a)$. In the same time, we will recalculate the Q-value for the previous state-action pair $Q(s_{t-1}, a_{t-1})$. This will give us the correction of the Q-value, i.e. the error $Q^{new}(s_{t-1}, a_{t-1}) - Q(s_{t-1}, a_{t-1})$. Finally, we will propagate this error only once and move to the next state, repeating the same procedure until the end of the episode.

Because in this game two agents are acting, I decided to train the Neural Network, such that both agents share the same Neural Network to make a decision.

3 Implementation

Due to the fact that the goal of this assignment is the Machine Learning part, I took some parts of the code in this GitHub Repository for simulating the game, i.e. to display the board and execute the Neural Network's actions or in other words, the environment of the game. All the agent related code and its interaction with the environment, was developed by me and it is in Python Script named *agent.py* in this GitHub Repository. Most of the implementation is done using the TensorFlow and NumPy libraries for Python.

In the game Othello (or Reversi), the environment is consisted of an 8×8 uncheckered board, such that every square is either free or occupied by a black or white disk. For this reason the input is represented as an array of 64 elements, such that -1 is denoting a black disk occupancy, 1 is denoting a white disk occupancy and 0 means unoccupied square. There are 64 possible actions, i.e. putting a disk on a specified square, but not all of them are legal. Given all of that, the Multilayer Percepton is having 64 input and output units and one hidden layer with 50 units.

In order to adapt the network for the reinforcement task, I had to write a low-level implementation of the Neural Network, i.e. matrix multiplication and derivation. A sigmoid function is used on both the hidden and the output layer. The learning rate in the backpropagation step is set to 0.01.

Moreover, we should handle the set of legal actions, i.e. the output for all illegal actions should be zero. For this reason, a vector of size 64 is created, such that it is having zeros on the positions of the illegal actions and ones on the positions of legal actions. In the end, we multiply element-wise the output of the Neural Network with this vector.

Since the two agents share the same network, we keep track of the relevant information and execute the same algorithm for both of them. We keep track of their previous perceived state, legal actions and the action they took following the epsilon-greedy policy. In one of the turns of the agents, first we forward-pass the current state and calculate its epsilon-greedy and greedy actions. We use the epsilon-greedy action to calculate the next state. Using the greedy action we calculate the new Q-value for the previous action-state pair, according to the Bellman equation. After that, we forward-pass the previous state and select the value associated with the previous action. Now, we create the error vector, such that it contains zeros everywhere except on the position of the previous action, where it contains the correction. After two forward passes, we make the backward pass and adjusting the Neural Net. Finally, we save the current state, legal actions and the epsilon-greedy action for the next iteration.

In the training process, the value for gamma is set to 1 and the number of episodes is 10 thousands. Moreover, after each iteration we play game against an opponent using random moves, in order to test the performance of the Neural Network.

4 Conclusion

This solution proves that for a short amount of time, the classical supervised methods can be applied in the field of Reinforcement Learning. Due to the time restrictions I was not able to fully train the model, which is to play around 2 million episodes. However, after careful checking the correctness of the solution and running it for short time, I believe it will reach the goal, and the algorithm will learn how to play the game efficiently. I hope the presented solution fulfills the requirements of the assignment.