Homework 2 - Artificial Intelligence

Gianluca Rea
Matricola: 278722

Introduction

For this homework, the adaptive min max was implemented to play the chess game. The following are shown and explained the heuristics implemented, the min-max algorithm with alpha-beta pruning, the Multi-Layer Perceptron repressor with its analysis and finally a final analysis of the games played.

Heuristics

1. **Material**
   The sum of the pieces times their value is returned, minus the sum of the opponent's pieces times their value (n=0, p=1, b=3, k=3, r=5, q=9).

2. **Piece Square**
   A score is assigned to all the pieces of the chessboard on the basis of the matrices in which the values of the various elements on the chessboard are reported (figure 1), obviously, this value is calculated by making the value of the player who has the turn minus the value of the opponent.

```
pawntable = [
    0,  0,  0,  0,  0,  0,  0,  0,
    5, 10, 10,-20,-20, 10, 10,  5,
    5, -5,-10,  0,  0,-10, -5,  5,
    0,  0,  0, 20, 20,  0,  0,  0,
    5,  5, 10, 25, 25, 10,  5,  5,
    10, 10, 20, 30, 30, 20, 10, 10,
    50, 50, 50, 50, 50, 50, 50, 50,
    0,  0,  0,  0,  0,  0,  0,  0]

knightstable = [
    -50,-40,-30,-30,-30,-30,-40,-50,
    -40,-20,  0,  5,  5,  0,-20,-40,
    -30,  5, 10, 15, 15, 10,  5,-30,
    -30,  0, 15, 20, 20, 15,  0,-30,
    -30,  5, 15, 20, 20, 15,  5,-30,
    -30,  0, 10, 15, 15, 10,  0,-30,
    -40,-20,  0,  0,  0,  0,-20,-40,
    -50,-40,-30,-30,-30,-30,-40,-50]

bishopstable = [
    -20,-10,-10,-10,-10,-10,-10,-20,
    -10,  5,  0,  0,  0,  0,  5,-10,
    -10, 10, 10, 10, 10, 10, 10,-10,
    -10,  0, 10, 10, 10, 10,  0,-10,
    -10,  5,  5, 10, 10,  5,  5,-10,
    -10,  0,  5, 10, 10,  5,  0,-10,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -20,-10,-10,-10,-10,-10,-10,-20]
```

```
rookstable = [
    0,  0,  0,  5,  5,  0,  0,  0,
    -5,  0,  0,  0,  0,  0,  0, -5,
    -5,  0,  0,  0,  0,  0,  0, -5,
    -5,  0,  0,  0,  0,  0,  0, -5,
    -5,  0,  0,  0,  0,  0,  0, -5,
    -5,  0,  0,  0,  0,  0,  0, -5,
    5, 10, 10, 10, 10, 10, 10,  5,
    0,  0,  0,  0,  0,  0,  0,  0]

queenstable = [
    -20,-10,-10, -5, -5,-10,-10,-20,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -10,  5,  5,  5,  5,  5,  0,-10,
    0,  0,  5,  5,  5,  5,  0, -5,
    -5,  0,  5,  5,  5,  5,  0, -5,
    -10,  0,  5,  5,  5,  5,  0,-10,
    -10,  0,  0,  0,  0,  0,  0,-10,
    -20,-10,-10, -5, -5,-10,-10,-20]

kingstable = [
    20, 30, 10,  0,  0, 10, 30, 20,
    20, 20,  0,  0,  0,  0, 20, 20,
    -10,-20,-20,-20,-20,-20,-20,-10,
    -20,-30,-30,-40,-40,-30,-30,-20,
    -30,-40,-40,-50,-50,-40,-40,-30,
    -30,-40,-40,-50,-50,-40,-40,-30,
    -30,-40,-40,-50,-50,-40,-40,-30,
    -30,-40,-40,-50,-50,-40,-40,-30]
```

Figure 1 - Matrix for Piece Square Heuristic

3. **Check Status**
   The value 100 is assigned and returned if the move allows the king to check, otherwise, 0.

4. **Moves Number**
   Returns the number of legal moves the state allows.

5. **Linear Combination**
   This heuristic is the linear combination of the above heuristics.
   In particular, the formula is:

   $$material * \alpha + mobility * \beta + boardState.legal\_moves.count() * \gamma + \\ + check\_status * \delta + checkmate\_status * \epsilon$$

   Where $\alpha = 10$, $\beta = 0.1$, $(\gamma, \delta, \epsilon) = 1$

   Several tests were carried out with different *alpha* and *beta* values, while *gamma*, *delta* and *epsilon* were always left at one.
   These values have been chosen in order to place the heuristics on the same decimal level, thus giving the same importance to all and not giving one the possibility of being preponderant in some way over the others. Besides this, it was found that the material heuristic is stronger than Piece Square, also for this reason more importance was given to Material.

A small note should be made, when a move allows you to checkmate that move is taken into consideration with respect to the heuristic

Min-Max with alpha-beta pruning:

This algorithm creates a tree in a dfs fashion, where each node of the tree represents the board in one state. Once the leaves have been reached, the evaluation of the heuristics is carried out and, depending on whose turn it is, the minimum or maximum value of the various leaves is taken into consideration. In fact, in the algorithm, there is a maximizing and a minimizing player, in the specific case of the root the maximum value of the next level is associated.
The maximizing players and the minimizing alternate in minimum and maximum until the leaves.

The alpha-beta pruning algorithm has also been implemented in the min-max algorithm. This algorithm tries to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree and thus saves CPU and memory time.

The min-max algorithm was run several times with various heuristics and with depths from 1 to 3.

Within the implementation we can find the instruction random.shuffle(moves), this instruction shuffles the list of possible moves that can be made every time. This instruction is used to not receive the same move every time the heuristic returns equal values in the roots. In fact, the heuristic returns equal values especially at the beginning when white has to make the first move. This way you will have a different move every time.

Adaptive MIN-MAX (MLPRegressor)

In order to create an Adaptive Min-Max a Multi-Layer Perceptron regressor (MLPRegressor from now on) has been implemented able to predict the Min-Max value with depth 3 and Linear Combination heuristic.

For this purpose, the boards and the min-max evaluation were collected in order to create the dataset and train the MLPRegressor.
In particular, 4807 games were played with white implementing the Min-Max alpha-beta pruning with the Linear Combination heuristic and black being implemented by an agent with the greedy algorithm and the Material heuristic.

Through the 4807 games played, a dataset of 82463 rows was created. Within the dataset, the first 64 columns represent the squares of the chessboard with the pieces (returned by the min-max) and the last column, in our case val, contains the evaluation of the heuristic (always returned by the min-max) for that state.

Before carrying out the training of the neural network, the "One-Hot-Encoding" was performed on the dataset. In particular, the first 64 columns have been encoded using this algorithm. Subsequently, the MLPRegressor was trained with three "Hidden Layers", each with 783, 512 and 128 neurons. This configuration had a test set score of around 0.73.

Both the MLPRegressor and the OneHotEncoder were backed up via the "joblib" python library. This library allows you to save configurations of neural networks or other things like the OneHotEncoder so that they can be imported into other notebooks/applications or whatever.

Analysis of the MLPRegressor

To be able to have a score of 0.73 many games have been played, probably playing double or triple or even quadruple the number of games can allow you to get an even higher score.

This analysis derives from the fact that several training tests were carried out.
In fact, the first training was carried out by creating a dataset of 301 games.
The second training with 1501 games played. The third training with 3003 games played. In the end, the 3 datasets were put together to create a single one and try to obtain a repressor with behaviour as close as possible to those of Linear Combination.

Furthermore, the OneHotEncoder allowed us to have an excellent representation for the regressor, in fact, other encodings were tried in addition to the OneHotEncoder, such as for example a simple encoding where the letters were replaced with numbers I decided or others more or less similar to this. In this regard, the highest score was obtained when the OneHotEncoder was used.

Analysis of the Games

In order to carry out an accurate analysis, 6225 games were played between the various agents with min-max at different depths and different heuristics.
All the statistics between wins-losses and draws have been reported in the "Match Stats.xlsx" file.
From the reported data it is possible to see that the agent with the most powerful Min-Max is the one with depth 3 and Linear Combination heuristic.
Below are agents with depth 3 and MLP Regressor and Material heuristic.
In particular, from the data in the excel file, it is possible to see the agents with MLP Regressor and Material are more or less on the same level.
Immediately after them is the Piece Square heuristic and finally the Moves Number and Check Status heuristics.