[DT0171] ARTIFICIAL INTELLIGENCE 2024/2025

HOMEWORK 2: Learning Agent with Minimax

Student: Tien Dung Nguyen – 301142 – tiendung.nguyen@student.univaq.it

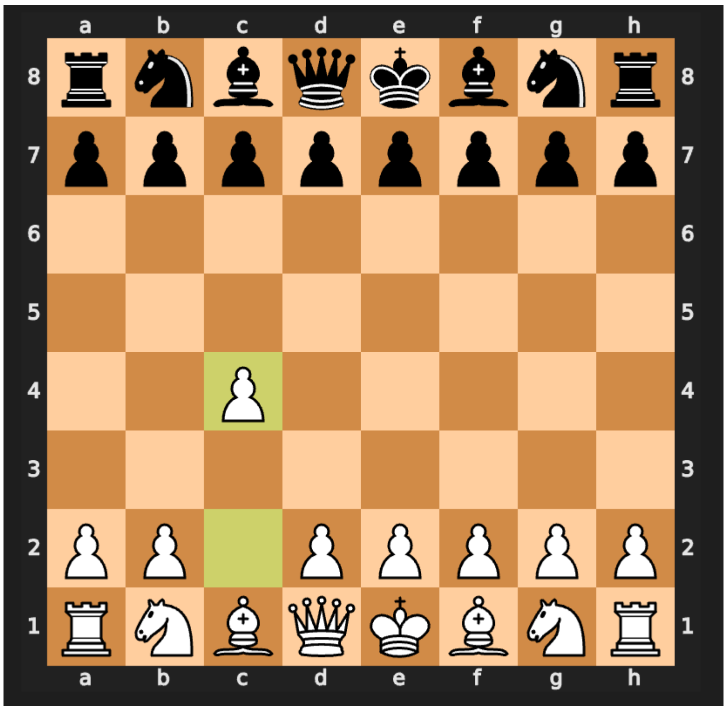
Part of this implementation is referenced from this GitHub repository.

1. Project structure

This project contains the following classes and files:

Classes:

- **Game:** An abstract class with abstract methods that represent different process of a game
- **ChessGame:** An extension of Game class, where the evaluation functions of chess game are defined
- **Player classes:** Represent different type of players with different decision strategies, including RandomPlayer and many MinimaxPlayers
- **GameVisualize:** Display the chess board and players' movements in real-time



- **FlexibleMLP:** Define the multiple-layer perception model to train the prediction model in later tasks

Other files:

- **config.py:** Contains global configuration
- **train.ipynb:** Use to generate train data and train models

- **test.ipynb:** Use to run scenarios and visualize experiment results

2. Task 1: Implement branch-limited Minimax

a. Implementation

In this part, the **Basic Material Evaluation** is used, which evaluates the board state based on the material (piece count) of both players.

$$H = h_0 = \sum_{i \in P} v_i - \sum_{j \in O} v_j$$

where P and O indicate the pieces left of current player and opponent, respectively.

Modify the original Minimax algorithm to take a new parameter ***choice_limit***, with default value -1 (considering all the possible moves).

```
class MiniMaxPlayer(Player):
    def __init__(self, player, game: Game=ChessGame(), depth=DEFAULT_DEPTH,
                 choice_limit=-1, verbose=False, generate_data=False):
        super().__init__(player, game, f"minimax, depth = {depth}, limit = {choice_limit}")
        self.depth = depth
        self.verbose = verbose
        # limit number of choices
        self.limit = choice_limit
        self.generate_data = generate_data
```

Modify the Game class to limit the number of possible moves according to the sorted score.

```
def sorted_moves(self, board: Board, player: bool, limit: int=-1) -> List[str]:
        moves = list(board.legal_moves)
        scores = []
        for move in moves:
            copy_board = board.copy()
            copy_board.push(move)
            scores.append(self.game_score(copy_board, player))

        moves = sorted(zip(moves, scores), key=lambda x: x[1], reverse=True)
        return moves if limit == -1 else moves[:limit]
```

b. Experiment with original abMinimax

In this experiment, the players played 10 games, the average time for each move and the number of wins are recorded. The figures Figure 1 and Figure 2 describe the results.

From Figure 1, it can be concluded that the average step time of original abMinimax algorithm is around 1 second, much higher than that of blMinimax algorithm (around 1ms). This can be explained by the fact that while abMinimax algorithm explores all possible moves, blMinimax only consider the limited number of moves and only the heuristic value H^0.

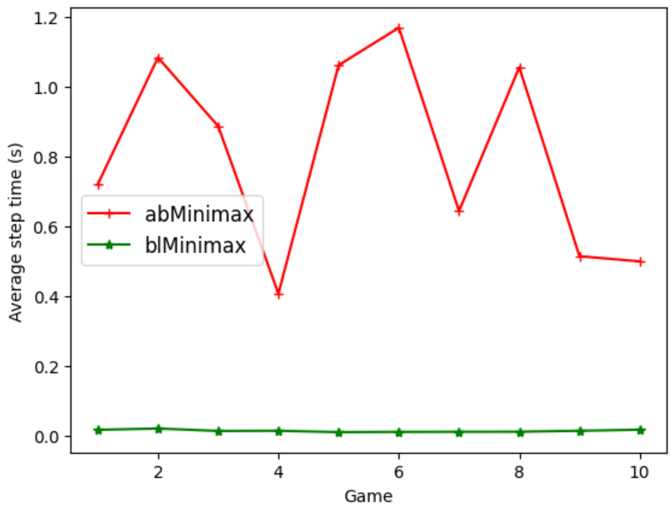


Figure 1. Average step time comparision between abMinimax and blMinimax

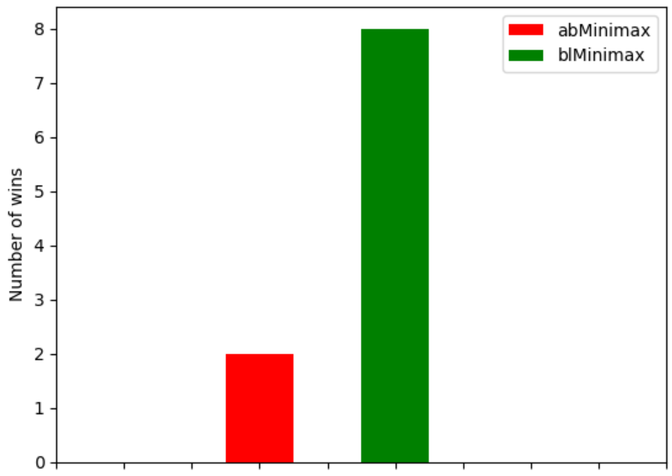


Figure 2. Number of wins comparision between abMinimax and blMinimax

In Figure 2, the number of wins of blMinimax player is higher than abMinimax's. Although blMinimax does not consider all possibilities, this might be the case where the value of H^0 truly reflects the competitiveness of a move, reducing the number exploration of moves while not compromising exploring moves without a good result.

3. Task 2: Optimized Minimax algorithm with H^l evaluation

a. Implementation

For this experiment, different players with H^l value are tested with the H^{l-1} player to get the best l value. Each pair of players played 10 games

Due to hardware limitation, only the l values of $l < 5$ are considered.

b. Experiment result

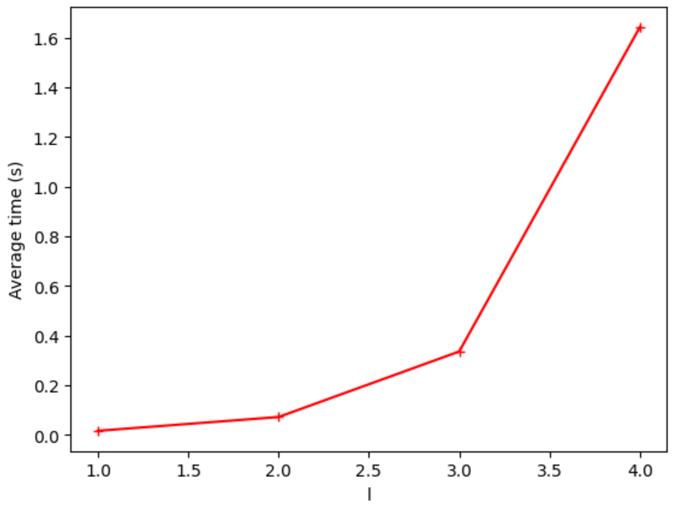


Figure 3. Average time for each step of blMinimaxl algorithm

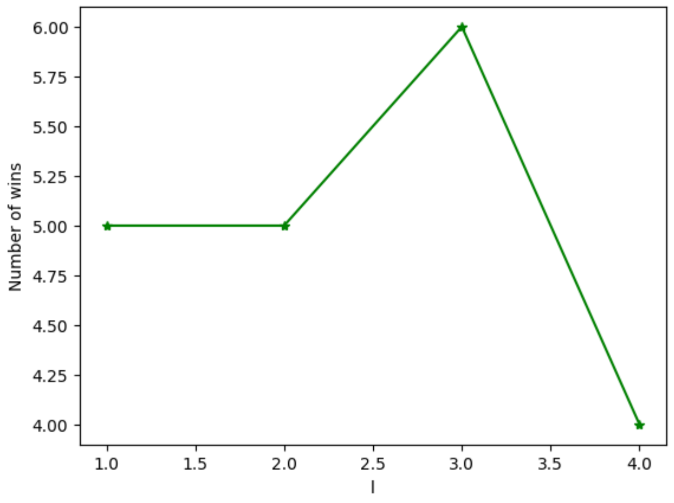


Figure 4. Number of wins compared with previous value of l for each of the blMinimaxl algorithm

The average time for each move increases gradually when l goes from 1 to 3 according to Figure 3. When $l = 4$, this value surged to over 1.5 second/move. Considering the number of wins compared to the previous value of l from Figure 4, there is a balance between blMinimax0, blMinimax1 and blMinimax2 when they shared the same number of wins/loses. However, this number increased slightly to 6 with H^3 then drop to 4 with H^4. Therefore, we can choose the optimal value of $l = 3$, which is the most suitable in terms of both time and performance.

4. Task 3: Train a prediction model and implement *pred*Minimax algorithm

a. Data generation

Games between an original Minimax player considering H^3 value with a random player are set up to generate dataset used for training. Each data contains the H^0

value as the input and H^3 as the output (ground truth). Around over 33,000 rows are generated in the **data.csv** file.

b. Model training

The generated dataset is divided in to train and test set randomly with the ratio of 8:2. A multiple-layer perception model with 5 hidden layers is used to train the predictive model, the architecture is defined as follow:

```
FlexibleMLP(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=64, bias=True)
    (1): Softmax(dim=1)
    (2): Linear(in_features=64, out_features=32, bias=True)
    (3): Softmax(dim=1)
    (4): Linear(in_features=32, out_features=16, bias=True)
    (5): Softmax(dim=1)
    (6): Linear(in_features=16, out_features=1, bias=True)
  )
)
```

Other parameters for training are as follow:

- Number of epochs: 1,000
- Optimizer: Adam optimizer
- Learning rate: 0.01
- Loss function: Mean Absolute Error Loss (L1), computes the mean of the absolute differences between predicted and actual values, defined by:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

However, the loss value fluctuates in a high amplitude and shows no sign of decreasing as the model keeps training. One possible explanation is because of the nature of the input with just one float input value that makes the model struggle to get the correct prediction.

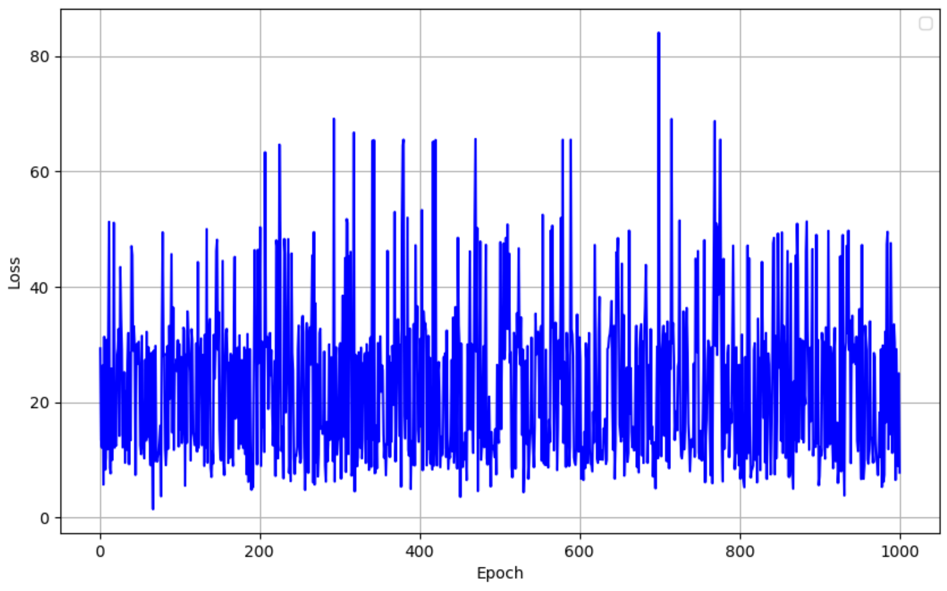


Figure 5. Loss value per epoch while training model predicting H^3 from H^0

c. Modify the Minimax algorithm

In the first steps, the predMinimax algorithm shows fast and diverse moves. However, it gets stuck at some points where the best move it could find is the same piece, moving around 2 cells like Figure 6 since the evaluation function only considers the value of the pieces left on the board, leads to the best move sometimes unchanged.

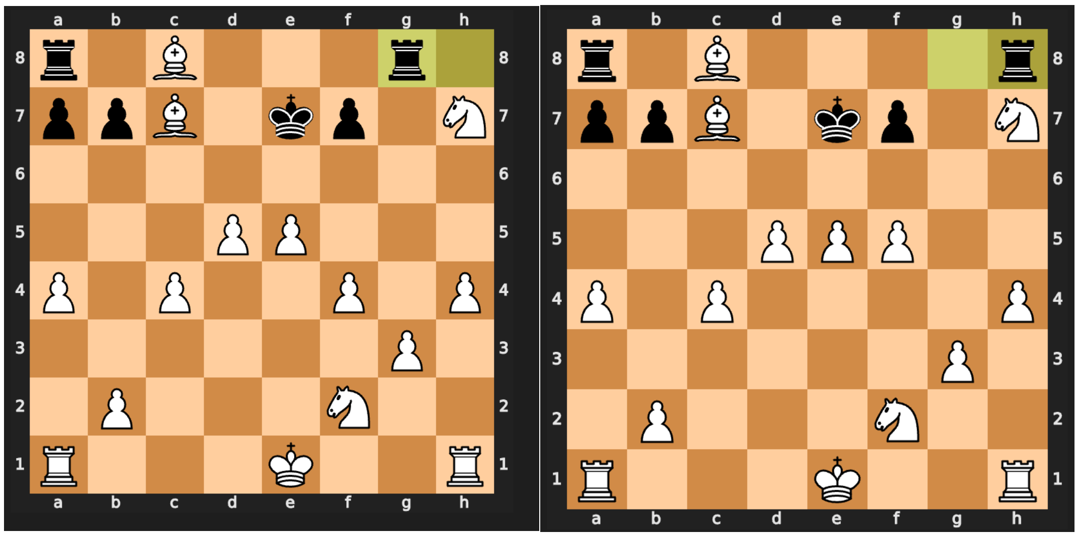


Figure 6. predMinimax problem

To avoid such situation, there are some ways like improving the value function to introduce penalties for repeating one move, or shuffle the entire move set at some points. However, an implemented workaround introduces here is having a **piece factor**, indicating the value of the opponent's piece being **captured** by using the move.

```
def sorted_moves_prediction(self, board: Board, player: bool, limit: int=-1) -> List[str]:
        moves = list(board.legal_moves)
        scores = []
        for move in moves:
            copy_board = board.copy()
            copy_board.push(move)
            piece = board.piece_type_at(NAME_TO_SQUARE[self.square_name(move)])
            if piece == None:
                piece = 0.5
            scores.append(piece * self.predict_h(self.game_score(copy_board, player)))

        moves = sorted(zip(moves, scores), key=lambda x: x[1], reverse=True)
        return moves if limit == -1 else moves[:limit]
```

d. Experiment with predictive Minimax

In this experiment, a player of abMinimax played 10 games against a player of predMinimax algorithm, both players used H^3 value with no limitation of moves to explore.

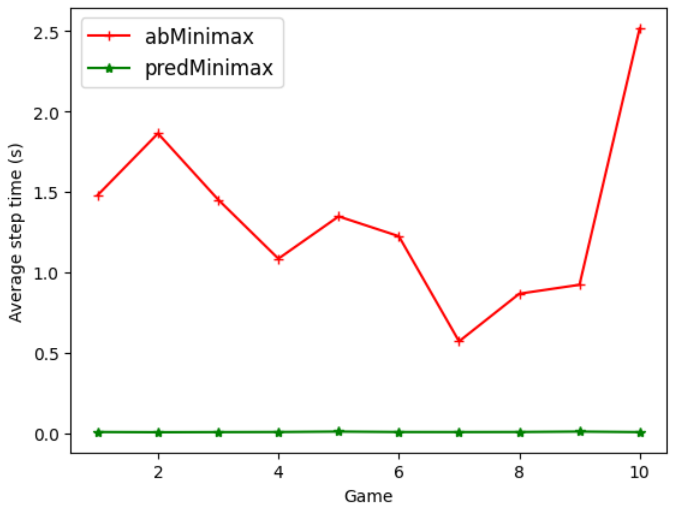


Figure 7. Average step time comparision between abMinimaxl and blMinimax

From the experimental result, it can be seen that the average time for each step of the abMinimax algorithm fluctuates, from 0.5 second per move till up to 2.5 seconds. In contrast, the average time of predMinimax is stable and approximate to 0. The reason for this is the predMinimax algorithm does not need too much time to exploit the further moves to calculate H^3 value as the abMinimax algorithm does.

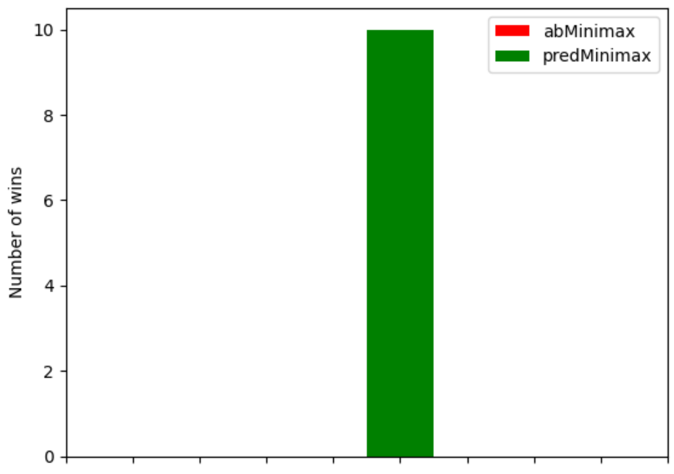


Figure 8. Number of wins comparision between abMinimax and blMinimax

Furthermore, the number of wins of predMinimax is 100% of the games. Although this might be somehow contradicted to the training result because the model is not expected to be good, this is expected because of the modification made while introducing the piece factor. This factor makes decision-making process more diverse, thus increasing the number of wins accoridingly.

5. Task 4: Comparision between *pred*blMinimax$_l$ and blMinimax$_l$

As experimented in task 2, the optimal value of l is l = 3, therefore this task only evaluates *pred*blMinimax$_3$ and blMinimax$_3$.

Experiment results are shown in Figure 9 and Figure 10.

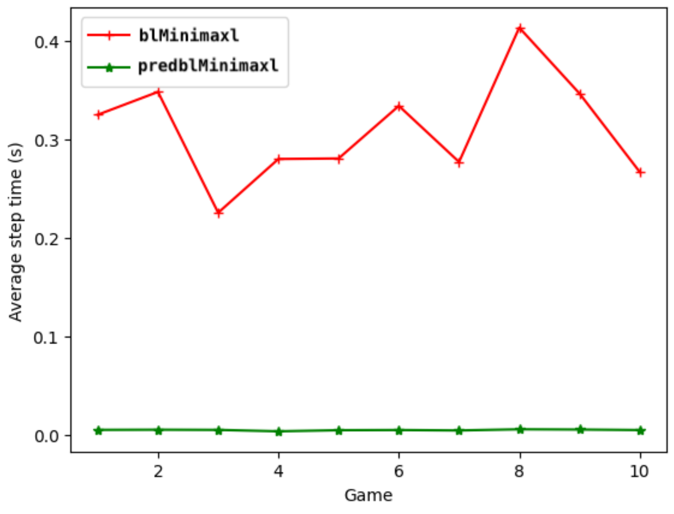


Figure 9. Average step time comparision between blMinimaxl and predblMinimaxl

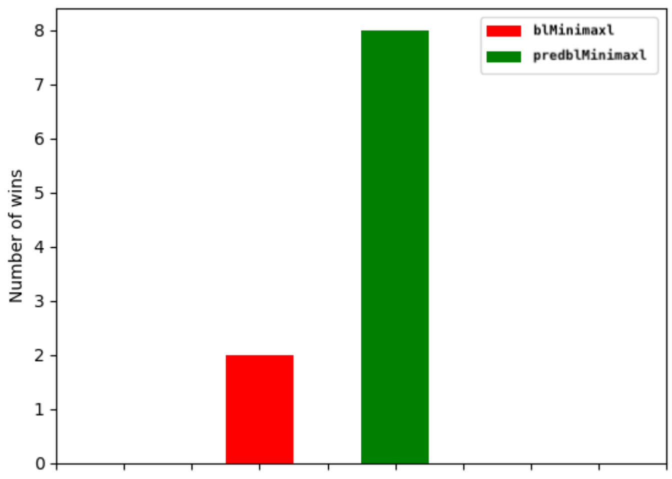


Figure 10. Number of wins comparision between blMinimaxl and predblMinimaxl

Each move in *pred*blMinimaxl algorithm takes much less than blMinimaxl algorithm due to the predictive model. This reduction of time allows the *pred*blMinimax explores more game state in less time, leading to faster decision-making. Moreover, the predictive algorithm shows effectiveness as it outperforms the blMinimaxl algorithm.

6. Task 5: Define new H^0 function and train a prediction model

a. Other evaluation functions

Besides the **Basic Material Evaluation**, some different evaluations are also taken in consideration.

- **Material and Board Control Evaluation:** Adds control of the center (d4, d5, e4, e5 squares) to the material evaluation. If the center positions are taken by a player’s pieces in, some points are added, points are subtracted otherwise.

$$h_1 = \sum_{i \in P} v_i - \sum_{j \in O} v_j + \gamma(p - o)$$

- **Position-Based Evaluation h_2:** Uses additional weight for the positional advantage of pieces (e.g., pawns near promotion, knights in the center).
- **Material and Mobility Evaluation h_3:** Combines material and mobility (number of legal moves) to evaluate the board state.

For simplicity, the new H function is defined by:

$$H = h_1 + h_2 + h_3 + h_4$$

In general, the H function can be defined by:

$$H = \sum \gamma_i h_i$$

where γ_i marks the importance of evaluation h_i

b. Comparision between the original H function

In this experiment, 10 games are set between the blMinimax algorithm with the original H function and the blMinimax algorithm with the new H function.

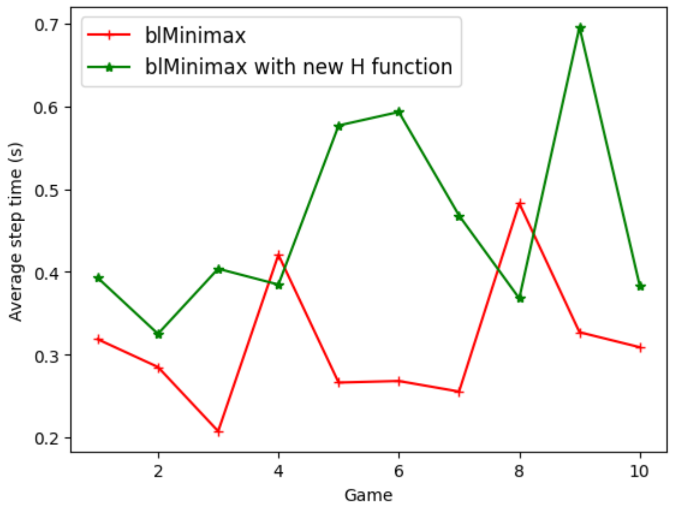


Figure 11. Average step time comparision between blMinimax and blMinimax with the new H function

In general, the average time per step of the blMinimax algorithm with the original H function is lower than that of the algorithm with the new H function, only two games those values are higher but not much. This can be explained by the fact that the calculation step of the new H function with different evaluation functions is more complex than the original H function.

In terms of performance, the blMinimax algorithm with the new H function is slightly better than the original blMinimax algorithm with 6 wins compared to 4 wins of total 10 games. The new evaluation function has taken in to consider different complex approaches to evaluate the game, which makes it somehow more effective.

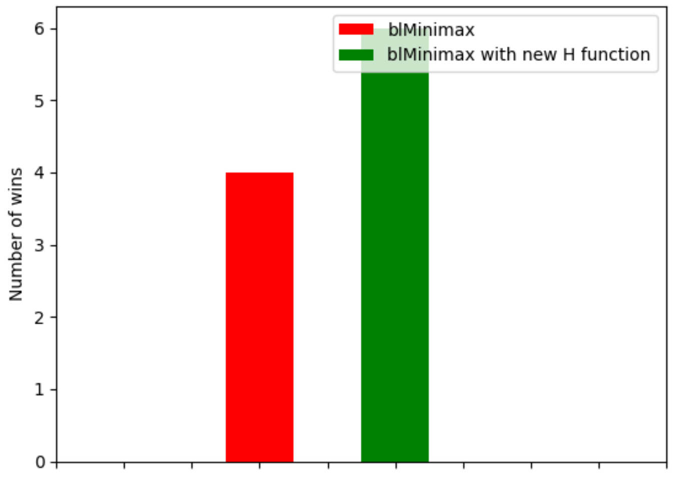


Figure 12. Number of wins comparision between blMinimaxl and predblMinimaxl

c. Data generation and training new predective model

With the same approach as the previous model, games between an original Minimax player considering the new H function value with a random player are set up to generate dataset used for training. Each data contains the $H = \{h_i\}$ value vector as the input and H^3 as the output (ground truth). Over 30,000 rows are recorded in the **data2.csv** file.

Model architecture are as follows

```
FlexibleMLP(
  (model): Sequential(
    (0): Linear(in_features=4, out_features=64, bias=True)
    (1): Softmax(dim=1)
    (2): Linear(in_features=64, out_features=32, bias=True)
    (3): Softmax(dim=1)
    (4): Linear(in_features=32, out_features=16, bias=True)
    (5): Softmax(dim=1)
    (6): Linear(in_features=16, out_features=1, bias=True)
  )
)
```

The training parameters are like the previous model:

- Number of epochs: 1,000
- Optimizer: Adam optimizer
- Learning rate: 0.01
- Loss function: Mean Absolute Error Loss (L1)

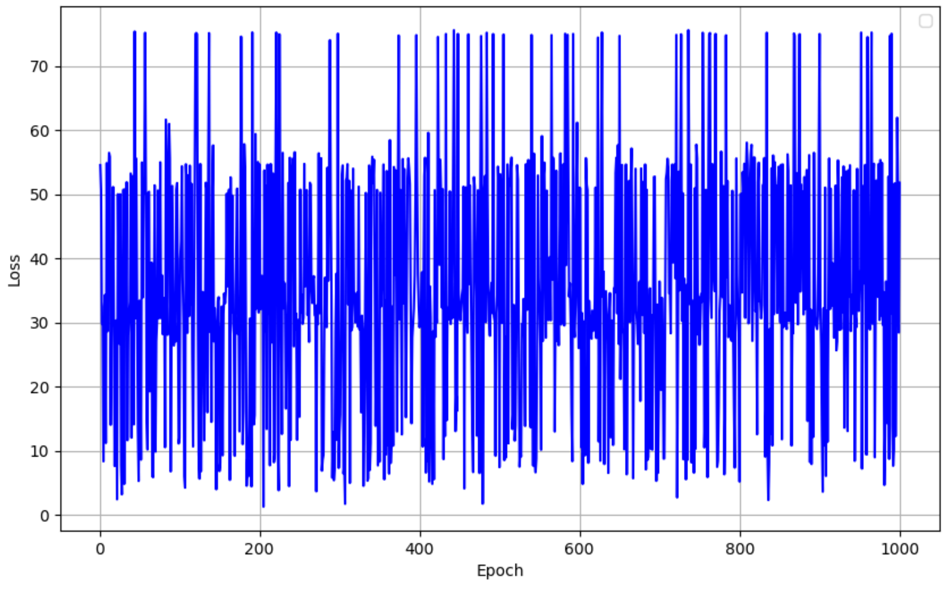


Figure 13. Loss value per epoch while training model predicting new H value

Compared to the previous model, this model experienced greater fluctuation while training and the loss function also shows no sign of decreasing as well. Besides the possibility of inadequate number of epochs, the model architecture might be an important factor that affects this training result.

d. Comparision with blMinimax using the original H function and blMinimax using the new H function

10 games are set up between the blMinimax algorithm with original H function and the blMinimax algorithm using the new H function predictor. Experimental results are presented in Figure 14 and Figure 15.

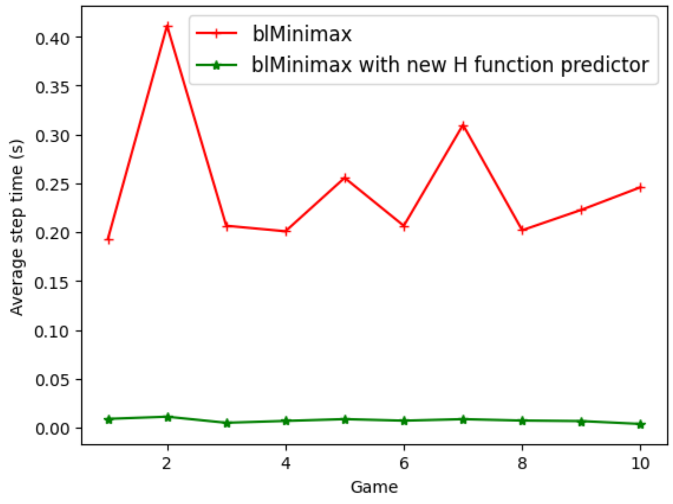


Figure 14. Average step time comparision between blMinimax and blMinimax with the new H function

Considering the average time, the value of the blMinimax algorithm with original H function varies from around 0.2 second to over 0.4 second, which is higher than the ***pred***blMinimax algorithm with new H algorithm. This is due to the simplicity of only using predictor to evaluate H function of the ***pred***blMinimax algorithm.

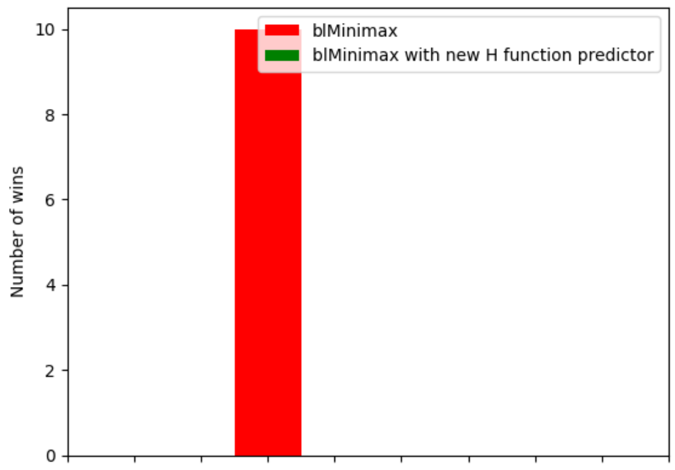


Figure 15. Number of wins comparision between blMinimaxl with original H function and predblMinimaxl

However, the original blMinimax outperformed the ***pred***blMinimax using new H function. This is expected based on the training result and no modification (like piece factor) is made like the previous model, the trained model is not effective enough to beat the original algorithm.

Ten other games are also experimented between the blMinimax algorithm with the new H function and the ***pred***blMinimax algorithm using the new H function. Results are as expected since the trained model only shows speed but not effectiveness compared to the original version of the algorithm **(**Figure 16, Figure 17).

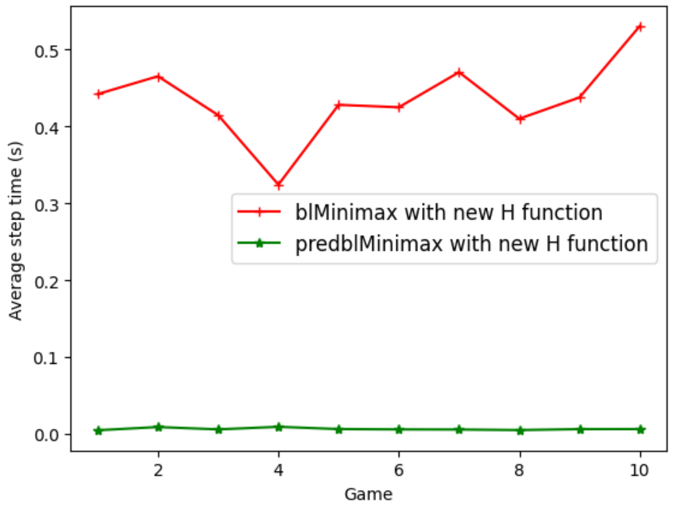


Figure 16. Average step time comparision between blMinimax and predblMinimax (both with the new H function)

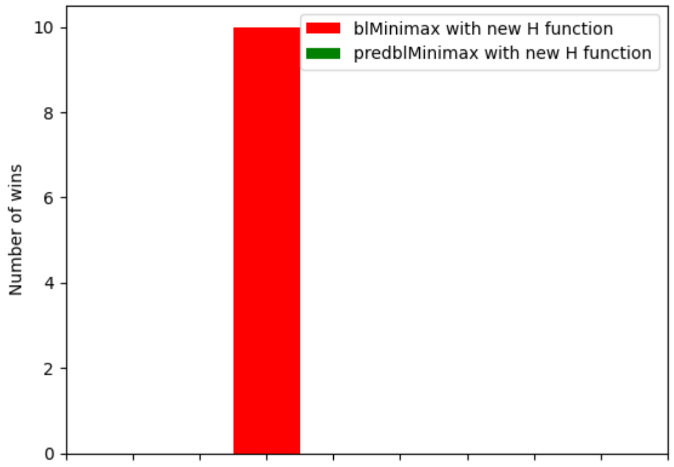


Figure 17. Number of wins comparision between blMinimaxl and predblMinimaxl (both using new H function)