UNIVERSITÀ
DI SIENA
1240

Neural Network for Chess Evaluation

Final Project for Neural Networks

- Dipartimento Ingegneria dell'informazione e scienze matematiche

Silas Ueberschaer, Benjamin Pöhlmann
SIENA 15/01/2024

UNIVERSITÀ DI SIENA 1240

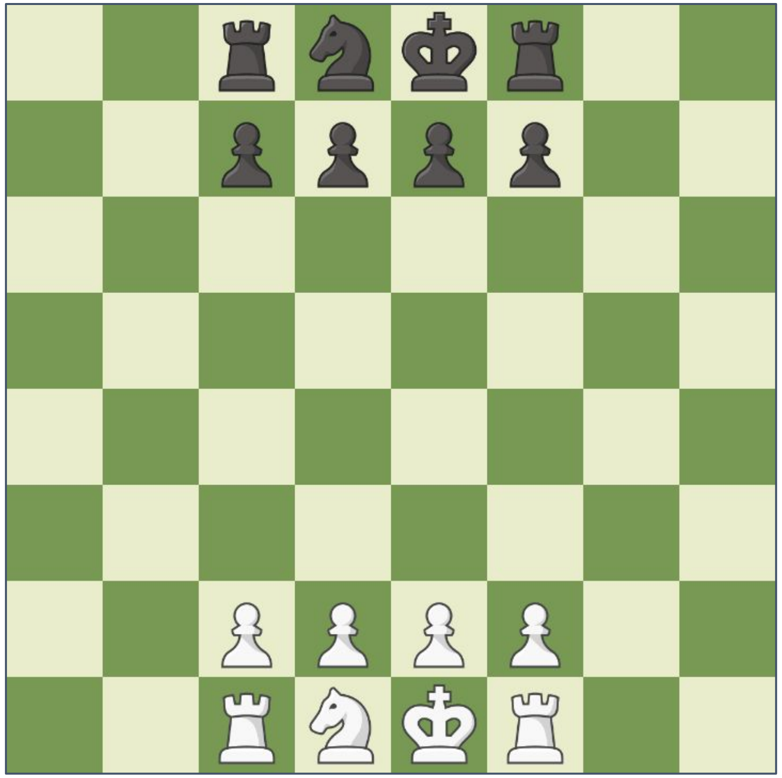
Goal for the Project

- **Creating a Neural Network that can predict who is currently winning**

- **Create our own dataset**

- **Play against the Neural Network, or let it play against another chess engine (Stockfish)**

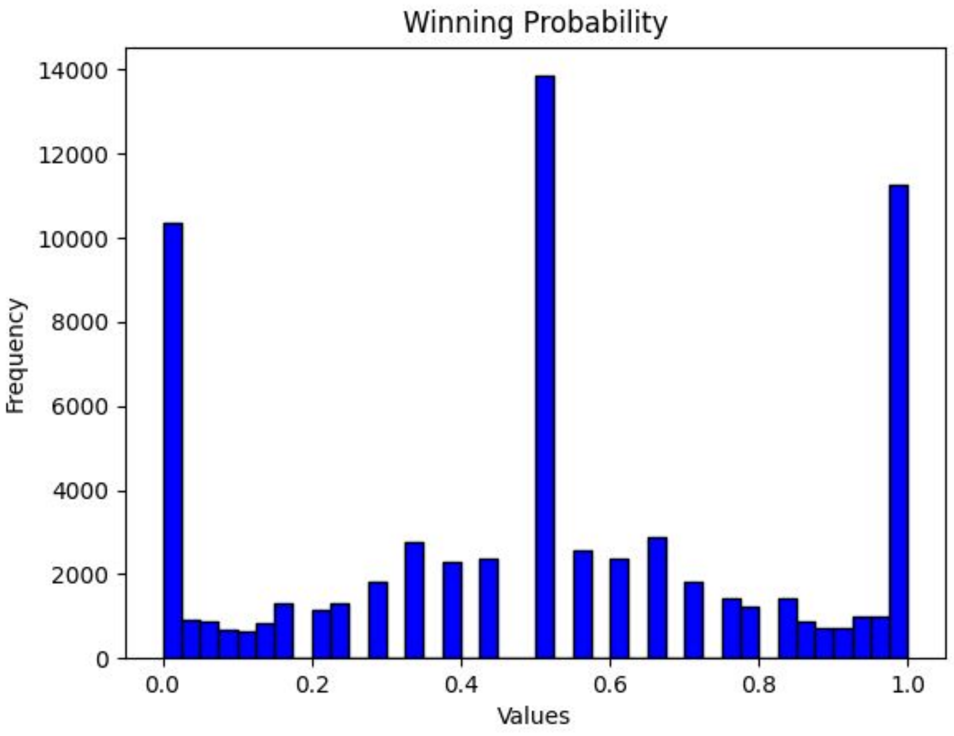
Creating our own data



Custom position

- Using custom start position with reduced amount of pieces to limit possible moves
  - Faster learning
  - Less data
- Simulating games and labeling them with Stockfish evaluation
  - >0.5 = white losing
  - <0.5 = white winning
  - $=0.5$ = draw
- Playing a mixture between good moves and random moves to create diverse dataset

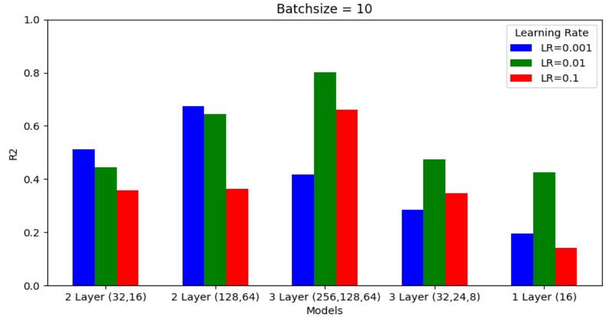
Dataset

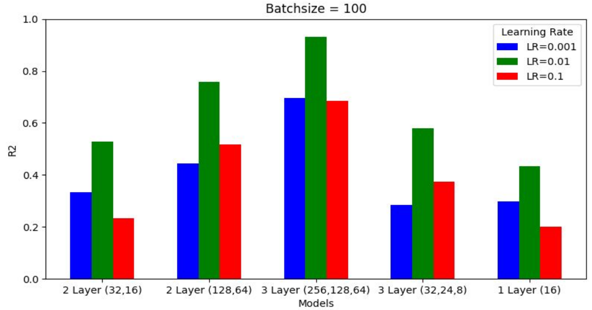


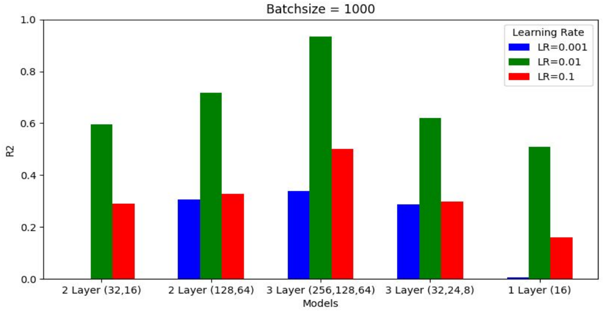
Dataset Distribution

- The dataset used for training consists of 70.000 positions
- Index 0 shows whose turn it is
  - 1 = White
  - 0 = Black
- Index 1 to 64 is a numerical representation of the chess board
  - 0 = empty, 1 = Pawn, 5 = Rook, 100 = King
- Index 65 shows the Stockfish evaluation of the position

Architecture

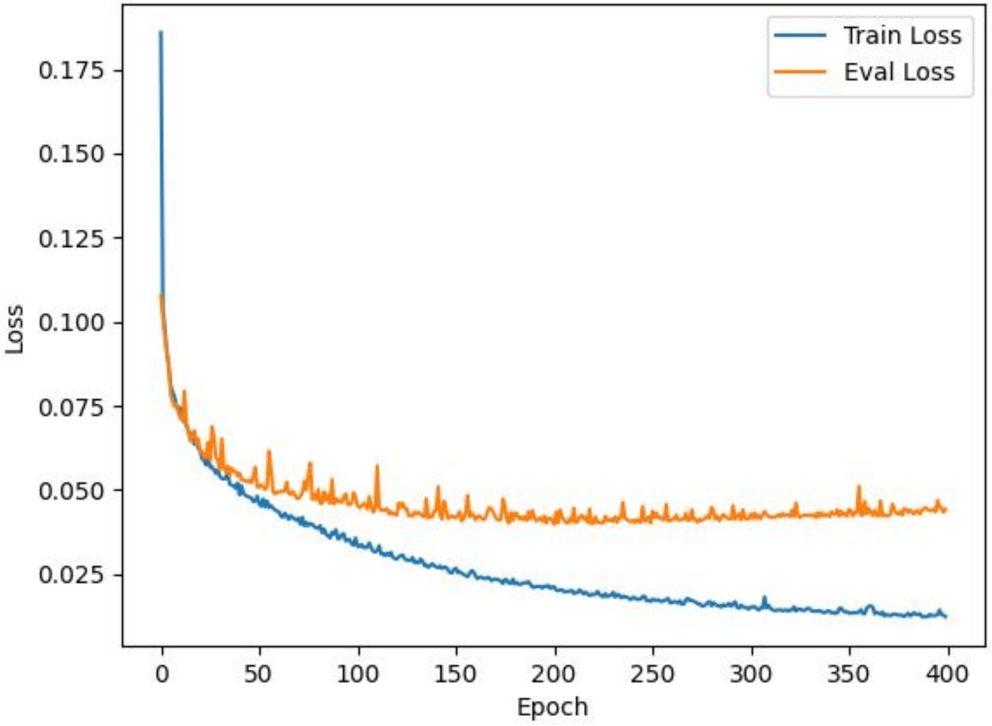






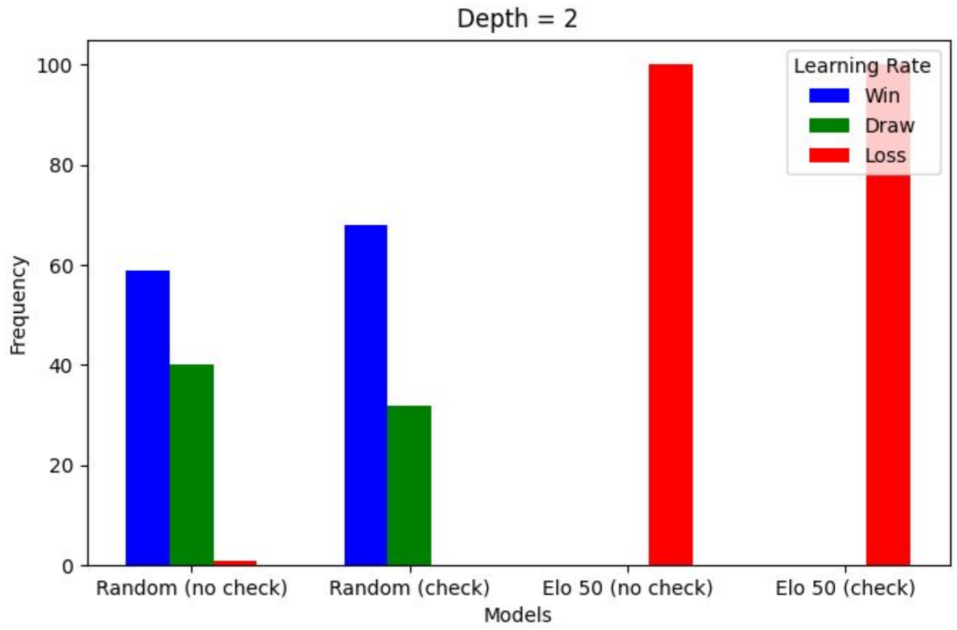
- **Simple feedforward neural network with multiple layers and RELU activation function**
- **To choose the right architecture we trained multiple models with K-Fold Cross-Validation (only on 10.000 positions)**
    - k = 3
    - learning rate = 0.001/0.01/0.1
    - batch size = 1000/100/10
- **Evaluation with MSE, MAE and R2**
- **Overall best is 3-Layer Neural Network (256,128,64) with learning rate = 0.01 and batch size = 1000**

Training



- Split the Data into Train and Eval
  - 80% Train, 20% Eval
- MSE Loss function and Adam Optimizer
- Trained for 400 Epoch
  - learning rate = 0.01
  - batch size = 1000
- Reached little to no improvement after around 200 epochs on the eval dataset

Playing



- Plays all moves and evals the board after that move and plays the best move
- Uses minimax to look multiple moves ahead
- Possibility to check if position is checkmate and return custom value instead of eval value
  - model not trained to detect checkmate
- Results (depth=2):
  - wins or draws against random moves
  - loses against stockfish at elo 200

Reinforcement Learning

Reduced number of pieces

- Goal: Use Reinforcement Learning to teach a neural network to play chess
- Deep Q Network used
  - Map a state and action to a Q-Value which is the immediate reward and discounted future rewards
- Simplifications used:
  - Reduce number of pieces (see figure)
  - Assume every piece can theoretically move everywhere
  - Only distinction for reward if move was good or bad but not to what degree

UNIVERSITÀ DI SIENA 1240

Reinforcement Learning - Setup/Architecture

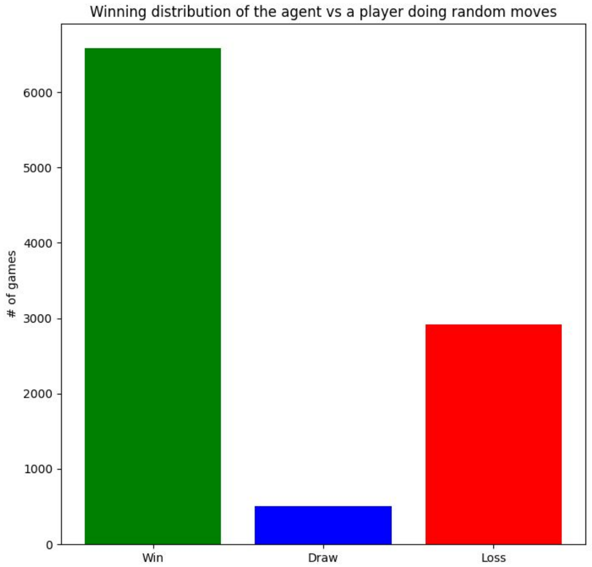
```
one_hot_mapping = {
    0: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],          # Empty
    1: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],          # White Pawn
    3: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],          # White Bishop
    4: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],          # White Knight
    5: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],          # White Rook
    1000: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],       # White King
    -1: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],         # Black Pawn
    -3: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],         # Black Bishop
    -4: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],         # Black Knight
    -5: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],         # Black Rook
    -1000: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]       # Black King
}
```

One-Hot-Mapping of chess board

- **Environment:**
  - **Chess board and its rules for legal moves**
  - **Enemy player (1000 elo)**
- **Agent is white player**
- **Reward: Evaluation by Stockfish if move increased probability to win, in general:**
  - **+1 if increased**
  - **-1 if decreased**
- **Input is a one-hot-representation of the board for each square on the board**
- **Output is Q-Value for each action that can theoretically be taken (even if illegal in current board)**

UNIVERSITÀ DI SIENA 1240

Data creation, training and result



Win distribution of playing 10.000 games of max. 20 turns versus an enemy doing random moves

- Play games with limited amount of turns (max. 20 each) to focus on early game
- Using epsilon-greedy algorithm to make mostly random moves
- For each move we save as training example
  - Current state (current board)
  - Action taken (move we did)
  - Reward for our action (change of Stockfish evaluation)
  - Next state (board after we and the enemy player moved)
- First 64 elements in output are actions of first pawn, then second pawn, ...

Problems and Future Work

- Dataset is to little and focuses more on early game
  - Create more data
- The models are still "small" for such a complex topic
  - use bigger models -> longer training
- Instead of using custom position train it on the standard chess position
  - Needs even more data and bigger models