Building a Chess AI with Deep Learning

Thomas Petrick
Georgia Tech
tpetrick3@gatech.edu

Kacy Lombard
Georgia Tech
klombard6@gatech.edu

Pooya Nayebi
Georgia Tech
pooyanayebi@gatech.edu

Luca Kurtz
Georgia Tech
lkurtz3@gatech.edu

Nicole Greene
Georgia Tech
nicolegreene@gatech.edu

Abstract

The problem we are addressing is that today's chess AI's are based on complicated evaluation . Current chess engines have been created to help find the best moves in any given position. The top chess AI's in the world can easily beat any human player, but these chess AI often play counter-intuitive "engine moves" which are based on logic that doesn't come naturally to humans playing the game. The effect of these moves can only be understood much later in the game. Previous work has been done to predict the move a human would make. Maia Chess is a deep-learning framework trained to predict the "human" move instead of the winning move. It was created to predict human-like plays or moves at any specific ELO rating. It was adapted from the AlphaZero/Leela Chess framework to specifically learn from human games at certain ELO ranges/ratings. To measure the success of the machine, the creators measured "move-matching accuracy", which is how often Maia Chess' predicted move is the same as the human move played in online games. Our somewhat unique approach is to create and train a chess AI on chess grandmaster games so that our engine will play more intuitive "human moves." Our goal is to create a chess engine that will give a "best" move output based on the current board state. Our expected results are that our evaluation function will perform at over 70% accuracy and the complete AI will be able to play chess at an intermediate level of around 1200 ELO. Our main goal is to maximize the accuracy of our supervised learning which will directly correlate to an increase in ELO rating.

Introduction

Our motivation for this project is that the members of the group are chess players and felt it would be interesting to explore the idea of creating a chess AI based on deep learning, a concept that we covered in this class. We have seen the emergence of current chess AI's and wanted to create our own from scratch that played more human-like moves rather than the perfect move that other chess AI's aim to play. The importance of creating a strong chess AI that will play moves similar to that of a human is that it would be vastly beneficial for people trying to learn chess, practice, and improve their skills. Current chess engines such as AlphaZero, Stockfish, and Leela Zero are far stronger than any human. These chess AI's also play very counter-intuitive moves and openings such as the Reti Opening and the Queen's Indian Defense. All chess engines rely on what is called an evaluation function, an algorithm to determine the strength of a given position. Almost all of these AI's use a very complicated linear combination of characteristics of a position to evaluate it's strength. The nature of these hard-coded evaluation functions makes the AI that is using them play counter intuitively. This makes them ineffective for training intermediate players. There have also been recent attempts to create a more intuitive AI with the introduction of the Maia Chess AI which is designed to specifically predict “human moves.” Our goal is to create a similar AI using deep learning. Essentially, instead of hard-coding this evaluation function, we want to use deep learning to evaluate the strength of a position. Our hope is that training our neural network with data about games played by humans will make the moves that it suggests more intuitive. Creating this evaluation function will be a matter of training our neural network. Then, we will use our trained network to predict the best possible move given a position.

Problem Definition

The portion of this project that involves machine learning is the creation of the evaluation function. We built this function using an artificial neural network. The goal of our evaluation function is to take two chess positions as inputs and return which of these two positions is better for white. For example, given one position where white is better and another positions where black is better, the evaluation function should return [1,0] or [0, 1], depending on which position is better for white. This evaluation function is at the core of every chess AI. We aim to build this evaluation function from scratch by training it with pairs of positions and the outcomes of their respective games. We juxtapose two positions and pass them in to the neural network and train it on the outcome of the games from which the position is extracted from. Our hope is that we can train the neural network to learn how to compare the positions. We will then use the trained neural net to evaluate the strength of positions that would be the result of a legal move given a current board state. Since this is a deep-learning model, assumptions are based on prior beliefs about the type of hypothesis the model should be learning. We assume that a randomly chosen position from a game that white wins is a stronger position for white.

Data Collection

Through extracting data with a python script, we actually have a unique data set that only exists with us. We used files in the pgn format that are used to represent chess games. Parsing these games with the python chess library, we extracted 200,000 random positions from over 20,000 grandmaster games from https://www.pgnmentor.com/files.html and mapped them to the corresponding game outcome. Then, we organized these board positions into pairs in which one position was eventually won by white while the other position was eventually won by black. Our expected outputs will be 10 or 01 depending on which game white won. To feed the data into our neural network, we encoded each board position into a binary bit string of length 774, storing information about the position, castling rights, and whose turn it is to play. Pairs of these bit strings are mapped to the outcome of the games and used as input/output pairs. Since we created a new data set by combining the PGN data with the python chess library, our data is novel. The data is already cleaned since we mapped random positions to their game outcomes. Because we are using pairs of positions as our input, we actually substantially increase the size of our data set. Thus, each epoch in our supervised training will be largely unique as we will shuffle the data set and pair the positions differently. This gives us an edge when training as we can generate a much larger training set with a limited amount of data. The games from which we extracted these positions come from a database of grand master games played throughout history. typically, a deep learning chess AI is trained on games played by high-level computers. However, given the goal of our project, which is to design an AI that plays more intuitively, our training set is based entirely upon games played by humans. We designed several methods that work together to generate the data. We have a method that parses each game and pushes a random amount of moves from the game onto a python board object from the python chess library. By doing this, we extract 10 unique and random positions from each game in our PNG format games. we store this large set of 200,000 positions in a numpy array containing all positions. We then label each of these positions with the corresponding results in a separate numpy array. We save these positions to our disk so that we can load them from a different python executable that trains our neural network with our data set. We picked this as our data set because the data is taken from real human games, so the machine will be trained on human moves, but since professional Grandmaster games are used, it will still play extremely well.

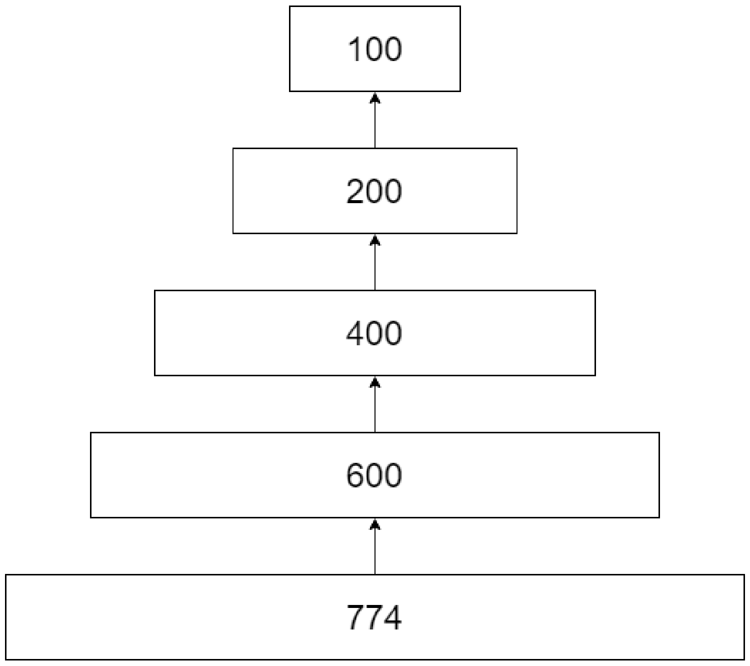

Figure 1: Deep Belief Network Architecture (fully connected)

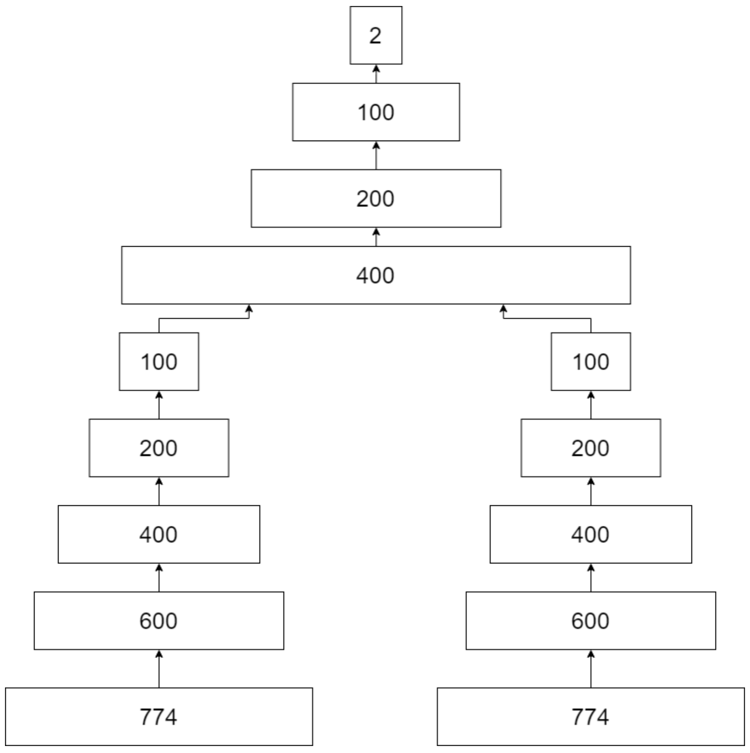

Figure 2: Supervised learning architecture (fully connected)

Methods

The baseline for this task is to manually evaluate a chess position based on a variety of factors and make a decision on which move to make based on material gain and foresight of a human. The core of our method is the creation of an evaluation function from scratch. In the history of chess AI, heuristic functions that were determined by a linear combination of factors such as king safety, pawn structure, and material advantage became the primary method for creating this evaluation. This manual creation of an evaluation function can also be seen as a baseline as it does not use data science techniques to determine a move, but rather a hard-coded heuristic function. For this project however, we will not be developing an algorithm for this evaluation function. Rather, we will be training a neural network through unsupervised and supervised training to evaluate the strength of a position. However, doing this is difficult given that the strength of a position has no definitive value. Instead, we trained our neural network to compare two positions using TensorFlow and Keras. The first idea we had to explore is how to pre-train part of our neural network to extract the high level features from a board position. From previous research, we found a method called greedy layer-wise training of deep belief networks. Essentially, because our input is a 774 element-long numpy array of bits, we can train a certain portion of the network to extract high-level features. The method we used is to create several networks with one hidden layer. The first is a fully connected network with the following structure: 774 - 600 - 774. We train this network to recreate the input using a 600 neuron hidden layer. We then use the values of this hidden layer to create another network with the following structure: 600 - 400 - 600. We repeat this process until we have abstracted our input to 100 neurons. This is the process known as greedy layer-wise training. For this unsupervised training, we used Relu as our activation function for all layers. Our error function was mean squared error and our optimization method was adam with a learning rate of 0.005. We trained each hidden layer for 200 epochs with a dataset of 200,000 positions. We saved the weights and biases of our pre-trained network onto our disk in a format called h5. After we trained our deep belief network, we fix the saved weights and biases for our supervised training. For this portion of our training, we juxtapose 2 of these deep belief networks as the first 5 layers of our network. The figure above illustrates this. We train this network through feeding pairs of games in which one comes from a victory for white and the other comes from a loss for white and train it with the results of the games. After the adjacent deep belief networks we added the following fully connected layers: 400 - 200 - 100 - 2. For our activation functions, we used Relu for all hidden layers and a softmax for our 2 node output. We trained this network for 1000 epochs, each with 200,000 unique pairs of positions. Our loss function was categorical cross-entropy as minimizing this will help us distinguish the two classes of positions. We created an adam optimization function with a learning rate that started at 0.005 and was multiplied by a factor of .98 with each epoch. Our batch size during this training was 256. After finishing the training, we saved the weights and biases of our supervised training network into a h5 file format. Given a chess position and this model, we are able to iterate through all legal moves from that position using an alpha-beta search, prune positions that will not be reached, and choose our best option given a depth.

Metrics

To assess the effectiveness of our neural network, we considered a variety of quantities. After each epoch, we print the accuracy of the neural network. This measures the percentage of the data set that the neural network predicts accurately. Because each epoch is largely unique due to the varying nature of our data set, the accuracy of the neural net on our training data is almost identical to its accuracy on testing data. The accuracy metric represents how often our network, given two positions, can correctly identify in which of the positions that white was victorious. Another metric we used was the output of our loss function during both supervised and unsupervised training. During the unsupervised training, we used a mean squared error loss function. The decrease of this loss function would be minimal during the last epochs. For our supervised training, we used categorical cross-entropy as our loss function. The output of this loss function would decrease with increasing epochs. These were our primary metrics for our neural network. After implementing the rest of the AI, namely the alpha-beta search algorithm that identifies a move by comparing positions with our trained neural net, we then assessed the accuracy of the moves that the AI would output. This metric was more subjective as we were not able to objectively judge the quality of a move. However, based on our personal knowledge of chess and chess analysis software, we were able to estimate the quality of our chess AI's moves. One final way of measuring the quality of our model was to actually have our AI play different opponents of varying strengths and assign an ELO rating to the engine. However, this metric is not favorable as it is subject to chance and requires a fully functioning chess AI that has the ability to play in all three stages of the game effectively.

Results

At the time of Touchpoint 2, we had completed supervised training with an experimental loss function and learning rate. When we finalized our initial unsupervised training, we achieved good results. Our mean squared error for each of our layers was about .003. The weights of this network are stored along with our self-generated data set. We had a script set up for the supervised training that was fully functional, but too slow for our personal computers. Because of this, we must rent a GPU instance and try to run our program there. However, we are pleased with the amount of training data we have and the initial results of our greedy layer-wise learning for our base layers. After we are able to train our supervised learning model, we will have a fully functional evaluation function to center our chess AI around. We initially trained our supervised model with very low epoch numbers. However, as expected our model did not perform up to standards. We expected this to change when we have access to more computing power. At this point, our accuracy was around 70 percent. This does not exceed the baseline we defined as members of our group that have experience playing chess can more accurately assess the strength of positions. However, we are pleased to achieve this level of accuracy with the limited computing power available to us. There is definitely room for improvement which we detail in the next section.

Results 2

After our first training session, we made considerable changes to our model. We doubled the data of the original training by adding more games and running our data extraction script again. We also designed a custom learning rate that exhibited a feature of exponential decay. Our epochs for unsupervised training were increased by a factor of 2 and the number of epoch for our supervised training was increased from 200 to 1000. We ran our new script with a much more powerful computer. With these changes, we were able to reach an accuracy of 77 percent, a 7 percent increase from our first training session. While this increase is non-trivial, it is not quite sufficient to play chess at a high level. We implemented an alpha-beta search algorithm that utilizes our trained neural network to output a move given a position. We also wrote a script that simulates a chess game that we can play against the AI. We played several games against the AI and determined it is not strong enough to beat the members of our team. However, we tested our AI's skills against players online on the website chess.com. The results of this experiment put our AI at about an ELO of 600. We believe there is significant room for improvement by changing several factors. First, we must increase the accuracy of our neural network by adding more training data and increasing the diversity of the games that it is trained on. We thought about including lower level games into our training data. Another possible improvement is to make modifications to the architecture of our unsupervised training structure. After running the unsupervised training, our model fixed the weights and biases associated with the deep belief network during the supervised training. If we left these weights open for modification, we might be able to exhibit better results. The quality of our data can also be improved. For example, we can filter out positions that happen adjacent to a capture, as this results in a transient advantage for one side. The biggest improvement in the game-playing performance can most likely be made in the actual search algorithm. We haven't included any of the rules of chess, including checkmate, into our algorithm. It is strictly running on the preferences of the neural net output.

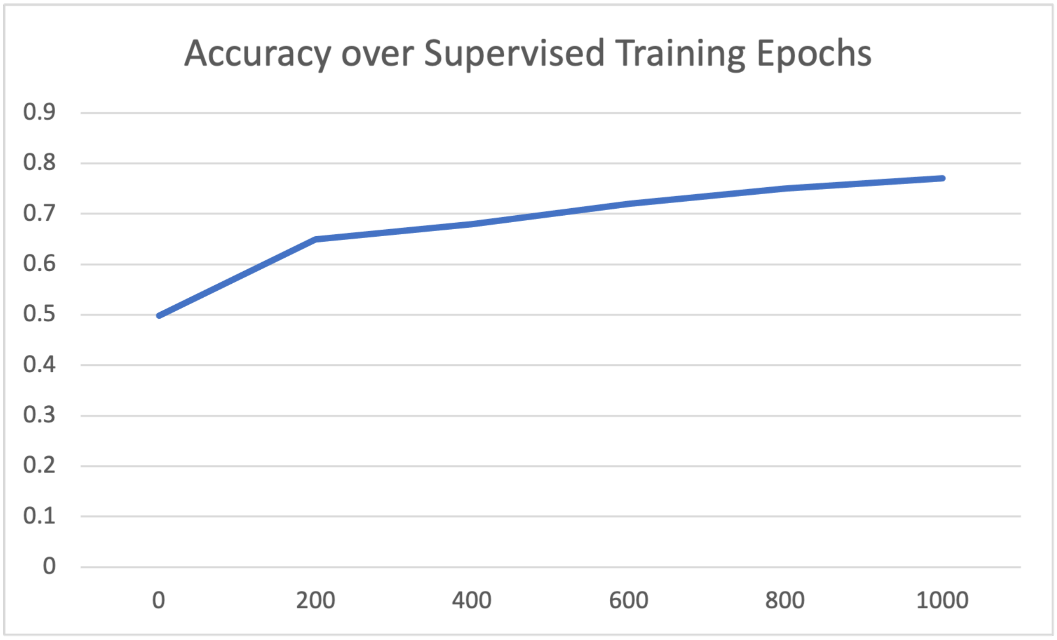

Figure 3: Accuracy of supervised network over training period

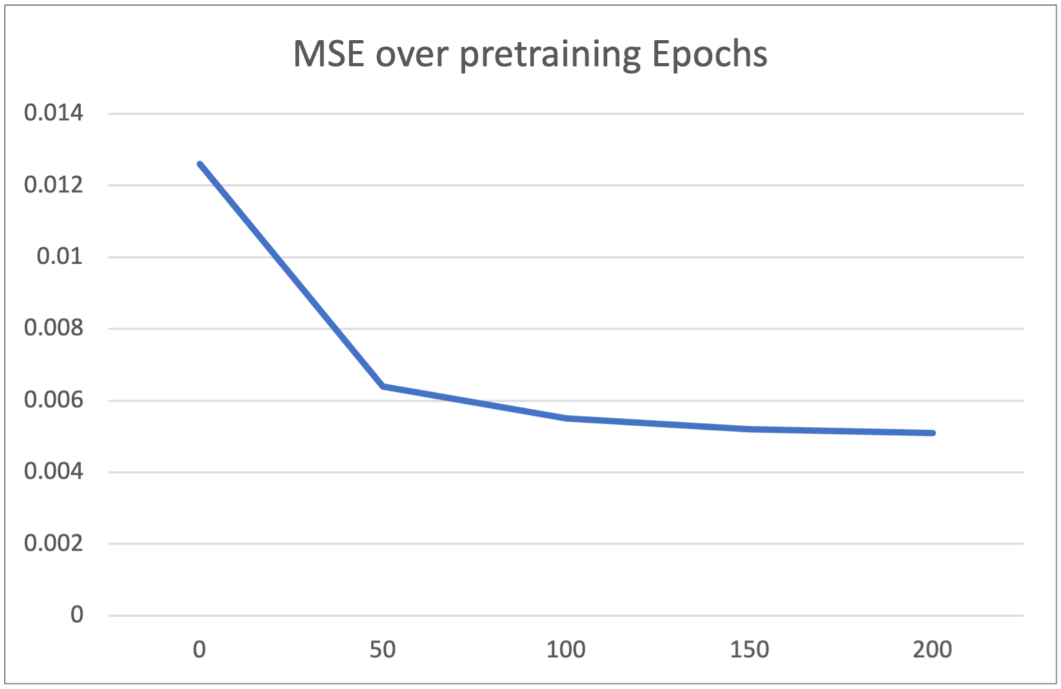

Figure 4: MSE of deep belief network over training period

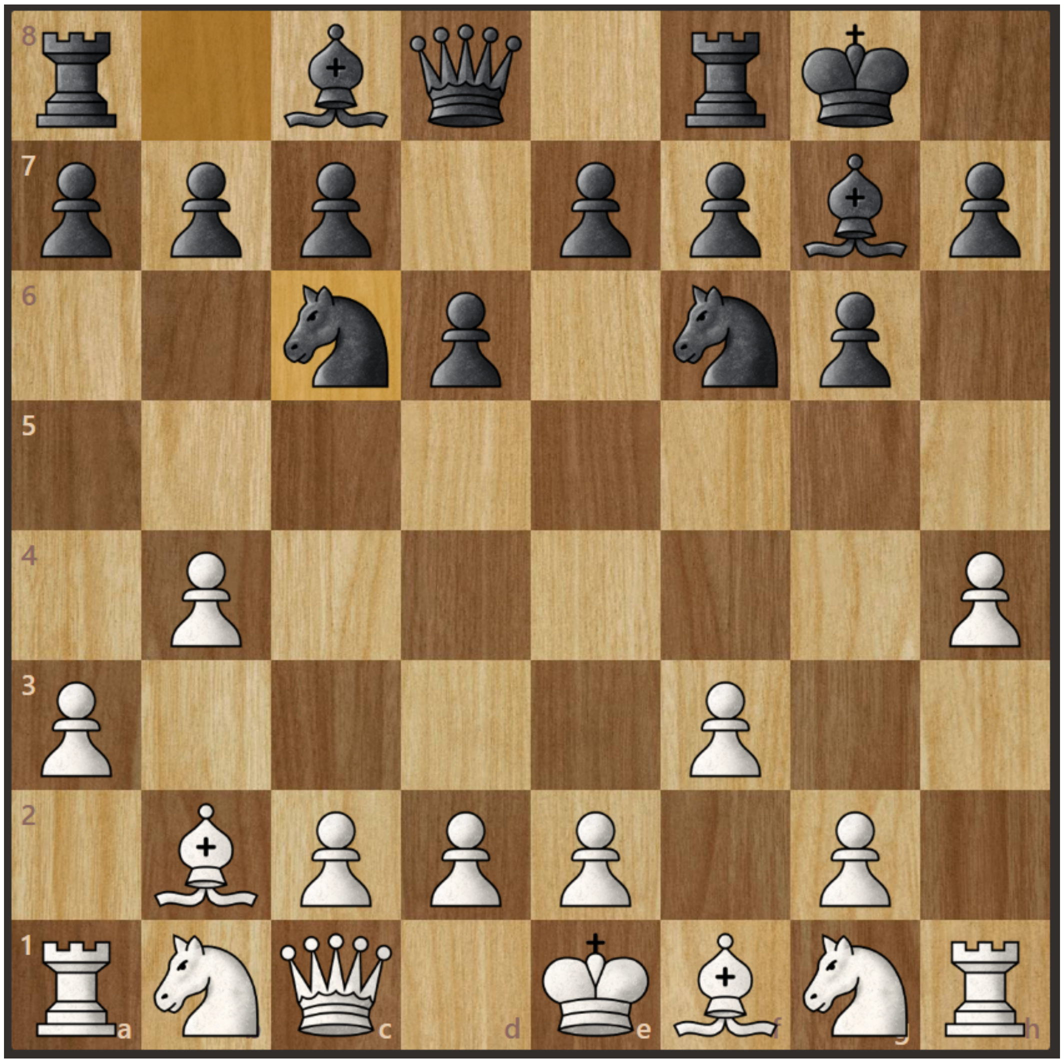

Figure 5: Chess Game Against Trained Network with Depth 4

Game PGN: 1. a3 Nf6 2. b4 g6 3. h4 d6 4. Bb2 Bg7 5. Qc1 O-O 6. f3 Nc6 *

In this game against the fully trained network (with a maximum accuracy of 77 percent and at a search depth of 4 in the alpha-beta search), the computer using the alpha-beta search and the trained network is playing white. We decided to play a setup based opening as black, known as the King's Indian Defense, since it can be played no matter what opening white plays. Since our neural network was not trained on openings whatsoever (only positions after 5 moves have been played), it doesn't fully understand opening concepts but rather only mid-game positional concepts. It is interesting to note that the network seems to very much favor "fianchetto'ing" its dark-square bishop. This is most likely due to it being a strong move and a common move at the grand master level. Additionally, it is interesting that against the King's Indian Defense, the network favored aggressively pushing two pawn's on the queen side, which is a common attacking idea and initiative against the King's Indian Defense. Additionally, in response to us playing the move "g6", the network favored immediately responding with "h4", which is a strong attacking idea. The idea is to break open black's king side and then somehow employ the dark square bishop on the long diagonal. This game was only played to a limited number of moves as a depth of 4 is a very time consuming calculation without better hardware. Overall, our network seems to understand some positional attacking ideas and complex moves but without a higher accuracy and greater search depth, lacks foresight in piece exchanges and lacks any understanding of opening principles.

Discussion

We have gained an idea for how game AI's work in general. We understand that an evaluation function is the core of a game AI. However, that evaluation function is usually built as a linear combination of factors affecting the game. We learned how to create such a heuristic function from scratch using deep learning. Not only did we utilize multi layered supervised learning with back-propagation, we also explored the realm of pre-training. We created a program that utilized greedy layer-wise unsupervised learning. This taught us about high level feature extraction and encoding higher level features in a smaller eventual output.We learned a lot about the pros and cons of different attributes of an artificial neural network. Examples of this include the loss function categorical cross entropy and the adam method of optimization. We also now understand that data collection and extraction is the most important and difficult part of building a deep learning network. Getting data in the correct format to optimize our training was the strong majority of the work that we did on this project. This was the first project for members of our group where computing power was a limiting factor. This taught us about the importance of enterprise computing and revealed the motivation of professionals to seek impressive computing power. There are many directions to improve the performance of our model. Further steps that we can take are modifying the architecture of our network and allowing for the pre-trained layer weights to be modified during our supervised training. Other improvements we can make are outlined in the results section above.

Ethics Statement

One socially positive outcome of our work is that people will be able to play and practice chess similar to a game against a real human, making it better for practice than playing against a machine that is impossible to beat. A big potential ethical issue is that someone can use the chess AI to cheat against other human opponents. Since the AI is better than the average human player, people could use it to get better moves than what they would normally play. A potential socially negative outcome of our work is, since playing against our machine will be similar to playing against a human, it may decrease the human interaction aspect of playing chess because people can just play against a machine in a similar way they would play against a human. If the machine is being created for the sole purpose of practice and improvement, making the user interface less "real" may prevent people from overusing it or playing the machine instead of playing with other people. However, if we didn't use human data it would defeat the purpose of creating the machine. A problem that many chess fanatics would raise is that machines are ruining chess by making it solely a game of preparation using an engine rather than a creative endeavour. This argument has been a common dilemma ever since the time that the world champion was beaten by a computer.

Bibliography

[1] Littman, M. L. (2007). *Markov games as a framework for multi-agent reinforcement learning.* Brown University / Bellcore, 94. https://www2.cs.duke.edu/courses/spring07/cps296.3/littman94markov.pdf

[2] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.* arXiv:1712.01815v1. https://arxiv.org/abs/1712.01815

[3] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., Silver, D. (2020). *Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model.* arXiv:1911.08265v2. https://arxiv.org/pdf/1911.08265.pdf

[4] McIlroy-Young, Reid, et al. *"Aligning Superhuman AI with Human Behavior."* Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining, 2020, doi:10.1145/3394486.3403219.

[5] J. Baxter, A. Tridgell, and L. Weaver. *Learning to play chess using temporal-differences.* Machine Learning, 40(3):243–263, 2000.

[6] Rodriguez, Jesus. *"Assumptions, Dimensionality and Some of the Original Motivations Behind Deep Learning."* Medium, 5 Oct. 2017, jrodthoughts.medium.com/assumptions-dimensionality-and-some-of-the-original-motivations-behind-deep-learning-a4df2f0f1689.

[7] Somers, James, and Louisa Thomas. *"How the Artificial Intelligence Program AlphaZero Mastered Its Games."* The New Yorker, www.newyorker.com/science/elements/how-the-artificial-intelligence-program-alphazero-mastered-its-games.

[8] David, Eli, et al. *"DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess."* ArXiv:1711.09667 [Cs, Stat], vol. 9887, 2016, pp. 88–96. arXiv.org, doi:10.1007/978-3-319-44781-$0_1 1$.