Group 8

Abstract:

The problem we are addressing is that there are no chess AI that play intuitively human moves. 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. 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.

Introduction/Background:

Current chess engines such as AlphaZero, Stockfish, and Leela Zero are far stronger than any human. These chess AI's also play very counterintuitive moves and openings such as the Reti Opening and the Queen's Indian Defense. These two factors paired together make training through playing these chess AI's very difficult. Creating a strong chess AI that will play moves similar to that of a human would be vastly beneficial for people trying to learn chess, practice, and improve their skills. The big problem is that these engines are looking so many moves in advance that a human would most not likely be able to understand why that move would be chosen. There have also been recent attempts at this with the introduction of the Maia Chess AI which is designed to specifically play "human moves." Essentially, we want to make an evaluation function from scratch using deep learning to analyze human games.

Problem Definition:

Our training data (our inputs) will be a large amount of grandmaster games in the "PGN" file format and our output will be who won the game. This is not a MDP as there are multiple agents and multiple states. It is best modeled by a Markov game. Our problem cannot be defined as a deterministic MDP as the opponents of our engine can make any legal move in any given position. If we were to model chess as an MDP and use reinforcement learning, we would end up with an engine similar to AlphaZero or Leela Chess. Our testing data inputs will be human made moves against our engine, and the outputs will be the top 2 or three moves in a position from our engine.

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 games 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.

Methods:

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. 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. 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. Eventually, we will have a trained model that is able to determine which of two positions is more beneficial to white. Given a chess position and this model, we will be 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.

The baseline we will be somewhat following is utilizing Markov games as a framework for reinforcement learning; however, we will not be training our engine against itself, but rather on training data consisting of a large amount of grandmaster games. The data we are using to train our model is free grandmaster data for top players in the form of PGN files. We use these files in conjunction with the python chess library. The PGN files for each player include hundreds to thousands of games with data for the event name, site location, players, elo ratings, result (who won and who lost), who played black vs white, and the series of moves for each player. A lot of data in the PGN's is not useful to us since we just want the board positions and moves, not other irrelevant information. To solve this, we are taking the data from the PGN's and using python's chess library to extract random board positions each game. Then, we convert these board positions to bitstring representations of size 774 and use pairs of board positions where white wins from one position and black wins from the other. One challenge that we have run into is arranging our data from PGN files to bitstring arrays. This task has proven to be quite challenging and has caused us to spend a large amount of time between touchpoint 1 and now getting our data prepared for our algorithm. Using these bitstring representations, we will train our network to compare the strength of white's position given 2 game states. Our training data will be novel, but the PGNs used were obtained online.

Metrics:

We think that a good way to evaluate our AI is to examine the accuracy of its evaluation function directly. We can set aside a portion of our training data as testing data and test its accuracy with that subset. Our first idea was to use k-fold cross validation where we test our algorithm on a part of our dataset that we didn't use for training. However, we believe that this will be semi-accurate because it is not guaranteed that the move that a grandmaster makes is the correct move that our algorithm should make. Therefore, there might be some problems with accuracy. Another potential metric would be using Elo and comparing the AI against other players until we determine an accurate Elo rating for our chess AI. This would be accurate because we would have a good idea of how our algorithm compares against human people in the world. All of the respected chess AI's are measured based on Elo and how they compare against professional chess players. Once we have a completely functional chess AI that makes use of this evaluation function, We can actually put it up against human players and determine its ELO rating.

Results:

We have finalized our unsupervised training and gotten 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 dataset. We have a script set up for the supervised training that is fully functional, but is 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 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 have trained our supervised model with very low epoch numbers. However, as expected our model did not perform up to standards. We expect this to change when we have access to more computing power. For now, our accuracy lies around 70%.

Conclusion:

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 pretraining. 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. Overall, we are excited to continue the project and go further with our training.

References:

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. 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_11.