Performance of AI using only Image Recognition to judge Boardstates: Applied to Othello

Heinrich Wizard Kreuser

December 2020

Abstract

In this project, we test the feasibility of creating a model that learns to play Othello through associating psuedo-images of boardstates with the winrates of players choosing those boardstates. We measure the performance of models trained with this technique with different styles of image generation.

Introduction

The goal of this project is to create an AI that thinks similar to an amateur player: from the current board, predict all of the next possible boardstates and judge which are "good" or "bad", then choose the boardstate with the highest likelihood of leading to a win.

We start by designing a model to play the game of Othello, after which we explain how we can train the model. We then experiment with two different methods of gathering images to train the model with, after which we compare the performance of players trained using those images.

Designing the model

The model first calculates all of the possible moves for the current boardstate and all of the boardstates each move leads to. It adds these move and boardstate pairs to a table. The model then takes in each boardstate (let's call each one b) and spits out a value denoting how good the boardstate b was, let's call this value $p(b)$. We can interpret $p(b)$ as 'the probability that b would lead to a win'. The model thus has a table of each move it can make, each b that is a result of making that move and its value $p(b)$. The logical next step is to perform the move of which the $p(b)$ is the highest.

move n	move	resulting boardstate b_n	$p(b_n)$
0	34	b_0	0.332
1	43	b_1	0.645
2	56	b_2	0.226
3	65	b_3	0.611

In this example, there are 4 moves we can make. Move 1 - 43 - will lead to a boardstate with value 0.645, so we choose to play the move 43.

Inputing Boardstates to the Neural Network as Images

With our imitation of an amateur player, we theorize that players simply look at different boards and judge them. We want our Neural Network (NN) to recognize good boardstates from bad ones, so we will convert boardstates to images that we will then feed to the input layer of the NN.

To accomplish this goal, we lend from the idea of classical image recognition techniques - which is to take the red, green and blue bands of each pixel of an image as a vector and feed those vectors to the Input Layer of the NN. For the bands of our image (the boardstate), each pixel is a vector where each index's values are made up of 0s and 1s based on characteristics of that square on the board.

For instance, in chess, we would assign vectors as such (only keeping Kings and Pawns into account):

Emtpy	[0, 0, 0, 0]
White King	[1, 0, 1, 0]
Black King	[1, 1, 1, 0]
White Pawn	[1, 0, 0, 1]
Black Pawn	[1, 1, 0, 1]

Here, index 0 would represent the presence of a piece, index 1 the colour of the piece, index 2 and 3 whether the piece is a king or pawn respectively.

For Othello, each square is similarly represented by a vector. But instead of black or white, the terms 'neutral' (empty square), 'friend'(our piece) or 'foe'(the opponent's piece) are used to help generalizing the board to the perspective of the player judging the boardstate:

Neutral	[0, 0]
Friend	[1, 0]
Foe	[0, 1]

Now to insert this into the input layer of our NN, we would flatten all of the vectors to combine them into one long vector, meaning the input image of size $(64, 2)$ (denoting the number of squares and number of contents per square respectively) would be flattened into a vector with size 128.

Generating Training Images and Training the Model

We will now explain how we went about training the models.

First, we generate a population of n models. We then input the population into what we call a 'session'. A session is similar to a cost function, but instead assigns a score to each player relative to their performance against other players. The purpose of this is to measure the performance of a generation. We will explain these sessions in more detail later.

The session pins certain players against each other and eventually returns to us a table of each player, their score, as well as all of the boardstates they had chosen during all of their matches against other players during the session.

We then create the training imageset by labeling all of the boardstates that each player chose with their score. However, we only train the top k (sorted using score from the session) players in the population and the other $n - k$ players are replaced by newly generated ones.

This process of having the population go through a session, and then training and removing players is called one generation.

Sessions

The purpose of a session is to take a population and choose models to play against each other in order to assign a score to each player at the end of a session as well as to record matches played between players so that we may use them to generate training images, which we count in terms of imagesets.

An imageset is the recording of all boardstates chosen by a single player during a match - thus, a match amongst two players generate 2 imagesets. It is important to note that in Othello, the game cannot progress for more than 60 moves, meaning that imageset will have a maximum of 30 images.

We will now discuss the two styles of sessions we used.

Free-For-All Style Generation

In this technique, we have all n players play against all of the other $n-1$ players. Meaning that we will have $n(n-1)/2$ pairs of opponents.

With this technique, the number of imagesets generated is twice for each match pair, so $n(n-1)$ imagesets.

This technique is good at generating a large number of imagesets for a relatively small population size. The quality of these images may be questionable as a bad player winning against an even worse player will produce misleading imagesets.

Tournament Style Generation

As the name suggests, we have players play in tournament-style brackets. Players who score high in the tournament will play against other players whom have also been scoring high (and vice versa for low scoring players).

The tournament is made up of rounds. Each round, we sort all of the players based on their score and have every $2i$ and $2i+1$ players play against one another (the 1st and 2nd ranked players will be opponents as well as the 3rd and 4th etc.). After all of the matches in the round have completed, the top $n/2$ players progress to the next round while the rest are kicked out of the tournament (however, the score that they had accumulated to that point is kept). This halving continues until there is one player left, the winner of the tournament.

This technique produces a "survival of the fittest" environment where most of the imagesets produced are made up of players who consistently stayed in the top half every round. The number of imagesets generated with this technique is roughly equal to $2n$.

Performance

We will now compare the details of both methods, measure each style's overall performance and then compare them.

Imagesets and Population sizes

In order to compare the two image generation techniques, we must run both generation styles for a unique value of n for each to produce the same amount of imagesets.

For Free-For-All style generation, $n(n-1)$ images are produced. For large n, the amount of imagesets tends to n^2. However, for Tournament style generation, this number is roughly equal to $2n$. This means that we will have to have a considerably higher population size for Tournament style generation to yield the same amount of imagesets.

For the experiments, we chose $n = 30$ for Free-For-All and $n = 441$ for Tournament style. This yields 870 imagesets generated from both.

Performance Measure

To measure the performance of players, we created 5 models to play against the best performing agent of each generation.

The first 3 were varying levels of alpha-beta pruning players, whom would look 3, 5 and 7 moves ahead to determine what move to make. We label these players ab3, ab5 and ab7 respectively.

The next measure is having them play against a random bot which chooses a random moves. For consistency, we have random bot be played 10 times and then take the average score.

Figure 1: Free-For-All

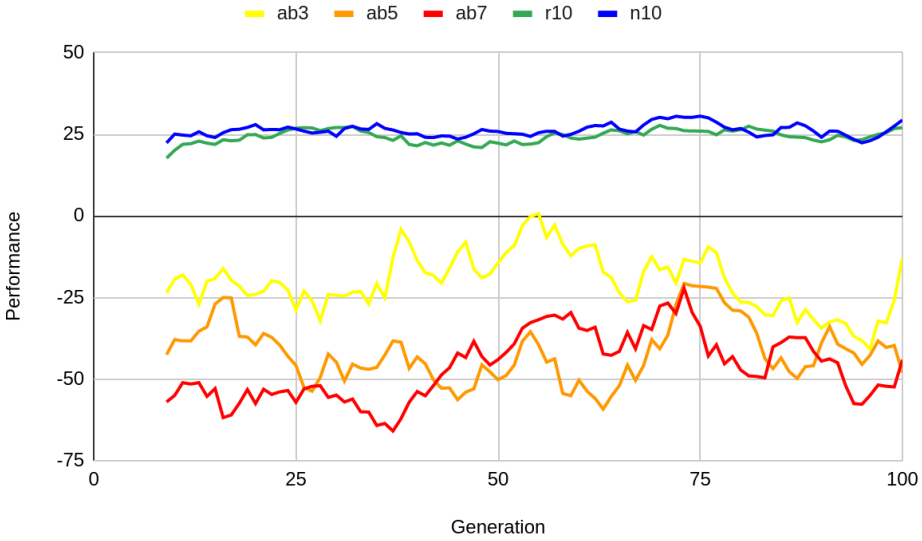
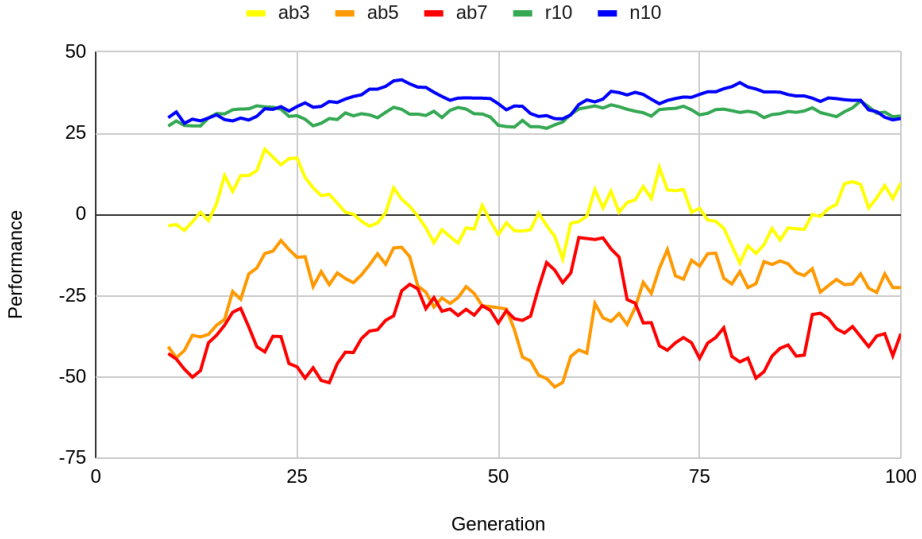


Figure 2: Tournament



For the final measure, we generated 10 models that use the same NN as the models that are trained, but these 10 players are generated at the start of the experiment and used throughout the entire experiment(s). We apply a similar consistency here by having the player play against each of the 10 players and

take the average score. The purpose of these players is to have players that perform random moves, but at a consistent level. We refer to these players as the Standard players.

Performance against these players can be seen in figures 1 and 2, which show the (running) average performance of the highest ranking model at the end of each generation (the running average is for the past 10 generations) against the 5 methods/models of measure over each generation.

After each generation, we train the top k generation using the images produced by the previous generation. The top performing model from the generation is then assessed by the above performance assessment and noted as the "performance of the generation". We ran the experiment for 100 generations for each style.

Performance Comparison

In the below figure 3, we compare the performance showed in figures 1 and 2.

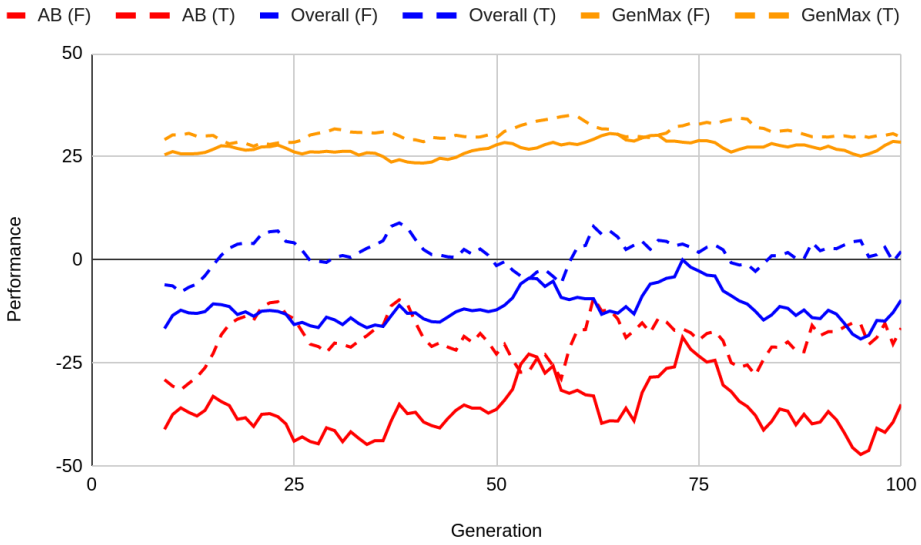


Figure 3: Comparison

The solid and dashed lines represent the performance of Free-For-All and tournament style generation separately.

The red lines denotes performance comparison against the alpha-beta players separate from the blue lines denoting overall performance including alpha-beta, standard and random players. Finally, the yellow line denotes the performance of the top agent in every generation against the rest of the generation.

Overall, the performance of Tournament style generation exceeds that of Free-For-All. The gap is especially visible during in the first half of the experiment against the alpha-beta players.

What is especially interesting to see is the performance comparison against random and standard players. From figures 1 and 2, Tournament style generation still outperforms Free-For-All style generation when it comes to standard and random players. This is interesting as performance against these players are often a good indicator of basic player skill over time. This means that the Tournament style's improvement over Free-For-All is twofold: generally the agents perform better (against random and standard players) as well as against alpha-beta players.

Conclusion

We measured the performance of agents whom learned to associate boardstates with the score of the players whom made them. We found that using different styles of generating these images leads to an overall better agent. However, the player is unable to keep up with players implementing alpha-beta pruning. It should be noted that the best player throughout the experiments could best alpha-beta players looking 3 moves ahead.

Future Work

One improvement could be another method of generating training images. The performance of agents rely heavily on the quality of the training imagesets and their associated player winrates. We associated each image in a match with the winrate of each player, but there could be boardstates throughout the match where the winner was losing and where most other players would lose. If one could solve this problem by assigning unique values to each boardstate somehow, it might increase the quality of the images.

Another improvement would abandon the method of training agents on images and instead use the tournament style as a form of filtering good players from the population. One would then use the top performing players in a Genetic Algorithm and combine the values of the Neural Networks of players. It would be interesting to see the performance of these types of players.