**Evaluating Approaches to Evolving Neural Reversi Players**

*Anna Porter and Rob LeGrand, PhD*

**Abstract**

Reversi is a two-player, zero-sum, strategy board game whose complexity lies between that of checkers and chess. Our research seeks to create Reversi-playing agents by using a genetic algorithm to evolve weights of a neural network. Agents rely on their neural network to make their decisions for moves of a game. In this paper we compare different styles of evolution and various settings for the neural network.

\*\*\*\*\*

**Introduction**

*Reversi Introduction*

The game Reversi, also commonly known as Othello, is two-player, zero-sum game. It is played on an 8×8 board with 64 identical game pieces. The pieces are double-sided with a white side and black side. The game begins with the center four positions filled with white and black tiles in a diagonal fashion. The black player moves first. New game pieces may only be placed on an empty space.

A newly placed game piece will flip enemy tiles if there is an unbroken sequence of enemy tiles between the newly placed piece and an existing friendly piece. Tiles may be flipped in any of eight directions; up, down, left, right and diagonals. Tiles may also flip in more than one direction if there are unbroken chains in both directions. A player may not pick and choose which directions or which tiles to flip. At any point in the game, a move is only valid if it flips enemy tiles. This means that turns may be passed back to the opposing player if no moves are available. However, voluntary passing of turns is not allowed.

The game ends when either the board is full or both players have no more legal moves to make. The final score is calculated by counting the number of tiles controlled for each player. The player with the most tiles wins the game. Ties are a possibility if the game ends with both players controlling the same number of game pieces.

*Neural Networks*

Artificial Neural Networks (ANNs) are mathematical representations of the human brain. They consist of neurons and weights. Each neuron takes real-valued inputs from the previous layer and multiplies these inputs by given weights. Using those calculations, a neuron creates a summation of those multiplications and adds a bias weight. Next, the neuron applies an activation function to the final sum. If this calculation results from a hidden neuron, it will be passed to the next layer as a new input. If the output neuron performed the calculation, the result will be the final value calculated from the neural network. This structure is capable of approximating nonlinear, multivariate functions, making it useful for a variety of problems.

*Genetic Algorithm*

A genetic algorithm is a computational representation of evolution that occurs in nature. An initial population will be generated using a set of guidelines. Organisms are represented by collections of genes. Genes may correspond to weights of a neural network. Each organism in the population will be evaluated by a fitness function. The ability of an organism to accomplish an assigned task is directly proportional to its fitness value. Based on these fitness values, parents will be selected and crossover between their genes will occur. Neural networks with strong weight combinations will pass some of their weights to the next generation. The resulting children from the crossover will be used as the next generation and the process will repeat itself for a certain number of generations. Using the principle “survival of the fittest,” the skill of organisms should improve in subsequent generations if conditions are appropriate.

**Related Works**

The complexity of Reversi is higher than checkers, but lower than chess[name and year]. Due to this fact, Reversi is the topic of many research endeavors. We will be using an approach similar to the configuration used by Chong et al. Their most interesting variation was the addition of a spatial preprocessing layer. They were only able to achieve master level play using a spatial preprocessing layer. The agents lacking this aspect were much more difficult to train and did not achieve levels of play comparable to the spatial neural networks.[only year] Our approach chooses to exclude this type of layer, in order to determine how much spatial information can be learned by a basic neural network.

Our strategy is also similar to Tuponja and Šuković, however they used the popular puzzle game 2048. The primary difference in their approach was the evolution style. These networks were also allowed to change in shape during the evolutionary process.[] This approach would be an interesting technique to explore in future work.

Shahzad et al. compared the level of play achieved by different evaluation functions. They included a standard Weight Piece Counter (WPC), Multilayer Perceptron Networks (MLP), Temporal Difference Learning (TDL), and a Monte Carlo algorithm using Tournament Play Technique. Of the evaluation functions examined, MLP, the strategy used in our work, was found to achieve the highest level of play []. Their results were encouraging for the purposes of our project, as we will be applying MLPs within our agents.

Festa and Davino proved fairly strong play can be achieved by using a minimax algorithm with a strong evaluation function[]. Adding a minimax algorithm to our existing techniques could lead to further improvement of the skill of our organisms, but we have chosen to exclude it.

A large source of inspiration for our research project was Blondie24 by David Fogel. In his book, he discusses using genetic algorithms to train a neural network with a spatial preprocessing layer for the game of checkers. Their agents also used a minimax algorithm to search for optimal board states. He was capable of achieving \_\_\_\_ levels of play.

**Methodology**

The goal of our experimentation is to create a strong Reversi player who utilizes a neural network to make its decisions throughout the game. The player will analyze each available move according to a heuristic value calculated by its feedforward neural network. The inputs for the initial layer of the neural network are generated from the Reversi game board. Inputs will be either −1, 0 or 1 representing an enemy tile, an empty tile, and a friendly tile respectively. The highest heuristic value corresponds to the best move to choose. Similarly, poor move choices will have low heuristic values. Neural networks differ by their assigned or evolved weights. Finding a good combination of weight values is critical to creating a good player.

In order to increase the skill of our players, we utilize a genetic algorithm to generate a population of organisms to evolve these weights. Organisms are assigned a fitness score based on the outcomes of games played against their neighbors. Based on the fitness score, parents are selected from the existing organisms to crossover their genes and create a new generation of players. In our case, we consider the weights of the neural network to be the genes. Every 1,000 generations, our generated players’ skill level is analyzed by playing every organism against every organism from another population. The opposing population is either the first generation or a subsequent increment of 1,000 generations. Pseudocode representing this process is shown below.

currentPopulation ← Generate initial population

For whichGeneration ← 1 to numGenerations:

For each organism in currentPopulation:

Play Reversi against 6 neighbors as Black

For each organism in currentPopulation:

Select parents using fitness score based on Reversi results

Crossover parent genes

Mutate genes

Create a new organism in newPopulation using genes

currentPopulation ← newPopulation

If whichGeneration mod 1000 = 0:

Save population for later evaluation

*Figure 1*

In these experiments, the initial generation is produced with no prior knowledge of the game. Established human strategies will not be given to the organisms in any way. Any appearance of intelligence they display is exclusively from the results of the genetic algorithm. A popular approach to increase skill level is to include a spatial preprocessing layer to provide organisms with more knowledge of the spatial relations of the board.[CITE THAT] However, our experiments will exclude this layer to see if our agents can learn spatial information on their own.

*Vanilla Configuration*

To begin our project, we settled on a basic set of parameters which came to be known as the vanilla configuration. Features of the vanilla configuration include gameplay on a Reversi board with size , with the center 4 tiles occupied by 2 black and 2 white tiles in a diagonal fashion. Weights are generated at the start of an evolution by using a normal distribution with a mean of 0 and standard deviation of 1. The activation function utilized by the neural network will be softplus. Vanilla configuration neural networks will have 64 inputs, a single hidden layer of 8 neurons, and a single output neuron.The final output neuron will not apply the activation function to its calculated sum. These networks will be exclusively feedforward ANNs, meaning there will be no cycles within the neural network. To generate the inputs given to our neural networks, an array is created with each entry corresponding to a piece on the board. If the space is occupied by a friendly tile, the value of the entry is 1. For enemy tiles, we use −1 and empty tiles are assigned 0.

The size of a population is arranged in a hexagonal pattern. Each organism plays 12 games, 6 as white and 6 as black, against its 6 neighbors. Organisms along the edge play organisms along the opposite edge. This creates a torus shaped population. Organisms will only be allowed to search one move ahead of the current board state, thus no minimax algorithm is implemented within their gameplay. In order to rate our organisms, each is assigned a fitness score during this stage. To calculate the value, we use a combination of wins, losses, ties, and the number of pieces controlled by a player at the end of a match. Wins are awarded 64 points, losses earn 0 points, ties are awarded 32 points, and the number of pieces controlled at the end of the game is added to this sum.

Each organism must choose another organism to breed with and produce a child organism. In the vanilla configuration, parents are selected from 6 surrounding neighbors using assigned probabilities based on their fitness scores. The lowest scoring neighbor is thrown out and his fitness score is subtracted from the remaining neighbors fitness scores. This new fitness value is directly proportional to the probability it will be selected as a parent.

New organisms are created by crossing over the genes of the two selected parents. For each gene, there is an equal probability it will come from the mother or the father. Mutation occurs for every gene and will add a random number generated using a normal distribution with a mean of 0 and a standard deviation of 1. These organisms will go on to play games, restarting the process and creating another new generation.

*Variations on Vanilla*

Of these properties, several settings will be varied for each future configuration. Initially, we will only change a single feature of the vanilla configuration. Features with strong results from each category will be selected for further experimentation. Secondary configurations will feature two or more changed features compared to vanilla.

Activation Functions

1. Rectifier
2. Sigmoid
3. Softsign
4. Threshold

The activation function is a basic variable feature. Neural networks will utilize rectifier, softsign, sigmoid, or threshold activation functions. This results in a slight change to the output summations of each neuron. Different activation functions generate different levels of success and some activation functions are more appropriate for some problems than they are others. Notice that rectifier and softplus are similar shapes and sigmoid, softsign and threshold are similarly shaped.

Mutation Types

5. Uniform Shake

6. Cauchy Shake

7. Normal Shake and Replace

We will evaluate several types of mutation for the crossover stage of evolution. Variation 5 will add a random number, generated using a uniform distribution between and . These values were chosen to get a similar standard deviation to the other distributions. A “Cauchy mutation” also adds a random number, but it is generated using a Cauchy distribution with a mean of 0 and standard deviation of 1. A “replace mutation” may be applied as well. This mutation re-initializes a gene with a 1% probability, replacing it with a randomly generated number using the same uniform distribution mentioned previously.

Network Shape

8. 1 Neuron

9. 8 - 3 Neurons

10. 16 - 4 - 1 Neurons

Neural networks may vary by shape. The most basic shape is a single output neuron, which will be referred to as 1N. In this case, 64 weights are proportional to the value of owning the tile corresponding to the weight. The 65th weight is a bias weight. Recall the default shape is 8 neurons in a hidden layer, with one neuron at the output layer. This shape will be referred to as 8N. Once neural networks begin to have hidden layers, relationships between weights and the board are less intuitive since a single weight could be used for more than one calculation. It is also possible to have multiple hidden layers, and we have chosen 2 shapes from this category to test. 8 neurons in the first hidden layer, 3 neurons in the second hidden layer, with a single output node will be referred to as “8 - 3 - 1”. “16 - 4 - 1” similarly represents 16 neurons in the first layer, 4 neurons in the next layer, and a single output neuron. It is significant to note that larger neural networks will have more weights, be capable of learning more interesting features, but they will also take more generations to arrive at these conclusions.

Parent Selection

11. Mother 7 Fitness Probability

12. Both 7 Fitness Probability

13. Best Mother 6

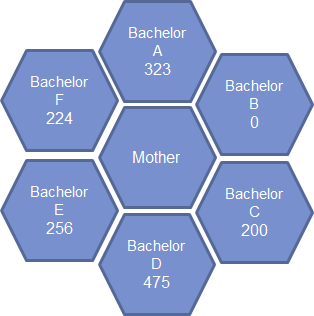
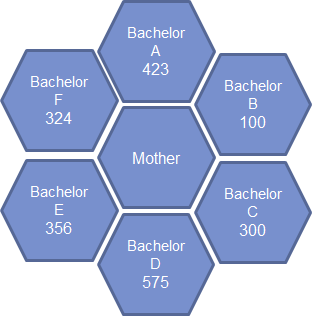
14. Best Mother 7

15. Best Both 7

Parent selection has also been chosen as a variable to consider. The figure from the left shows how the 6 neighbors are arranged around a center organism. The numbers at the bottom of each cell represent fitness scores.

Variation 11 will select the mother from the 7 possibilities using a probability directly proportional to their fitness scores. If the center organism is chosen as the parent, the organism will have two identical parents. Both 7 Fitness Probability will choose both of the parents according to a fitness probability. However, the parents chosen will not be allowed to be the same organism.

Variations 13, 14, and 15 are all deterministic parent selections. Best Mother 6 chooses the best mother, based on fitness score, from the 6 neighboring organisms, with the father in the center. Best Mother 7 chooses the best mother from all 7 options. Variation 15 takes the top two organisms from the 7 possibilities.



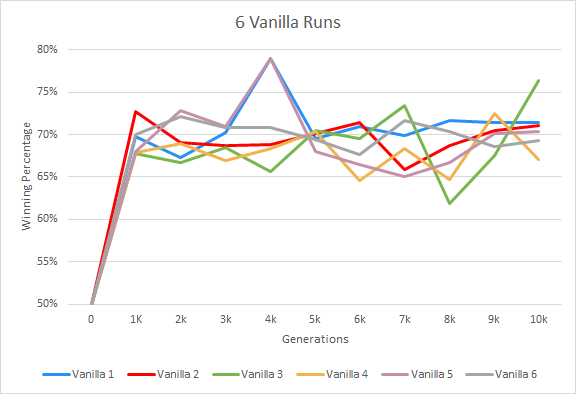
**Results**

Each variation was evolved for 10,000 generations and populations were saved and evaluated along the way. Every 1,000 generations, the each organism in the population plays each organism from the first generation. Winning percentages were calculated by adding the number of wins to half the number of ties and dividing by 20,000. These experiments were repeated from another randomly initialized population for another 10,000 generations. Within each categories of variation, the 10,000 generation of each variant from the first evolution were played against each other and their total wins from the outcomes of these games are displayed.

The evaluation approach previously described was found to be a bit unfair. The skill of one of our initial populations was especially poor and resulted in the Best Both 7 parent selection becoming an outlier for the first run. In order to achieve a more fair comparison, every 10,000th generation was played against a baseline population of size 50×50. Each organism within the baseline population has 8 neurons and uses the softplus activation function. Winning percentages were calculated using the same strategy described above.

*Vanilla 6 Times*

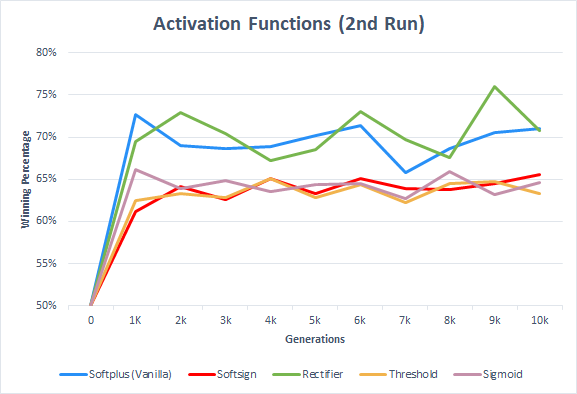
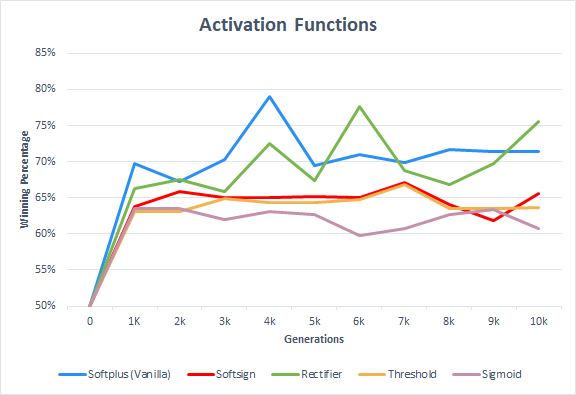
We chose to run the vanilla configuration 6 separate times. The initial populations are randomly generated, so each of these evolutions will begin at a different starting point. The mutation along the way is also random, so one thread of evolution will never be exactly the same as another. With the exclusion of outliers in generation 4,000, results are fairly closely related. At the final generation, 4 of the populations achieved winning percentages within an approximate range of 5%. NICE WAY TO SAY THIS SHOWS CONSISTENCY OR SMTHNG

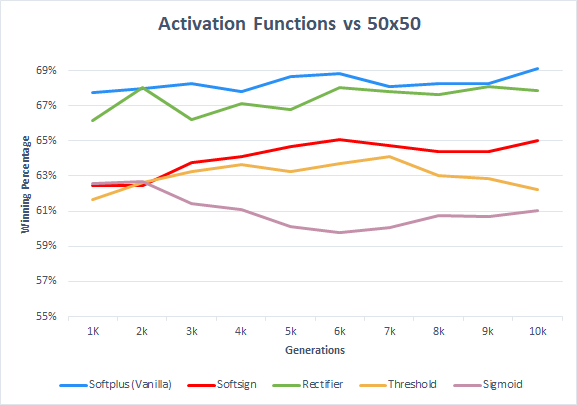


*Activation Functions*

Activation functions with similar shapes performed relatively similarly throughout their evolutions. It seems that activation functions which allow large numbers of values greater than one have done better than functions which restrict their output values to a small range. The grouping of similarly shaped activation functions holds true for all of the graphs and the direct comparison of activation functions. Sigmoid appears to perform the worst, as they had the lowest number of total wins. This was a surprising result, as sigmoid has been typically the standard activation function used with neural networks.

At each generation, softplus had the highest or second highest winning percentage, even when compared to the baseline population. They also had the largest number of total wins with 139,723 wins. When each 10,000 generation of each activation function were played against each other, Softplus had the highest number of wins. Due to these properties, our decision to make softplus the vanilla setting was reaffirmed.





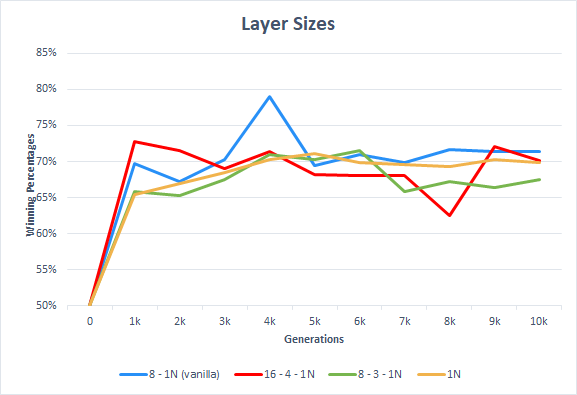
Should we insert the following graph this way, or take a screenshot of the way it appears in excel and insert that image.

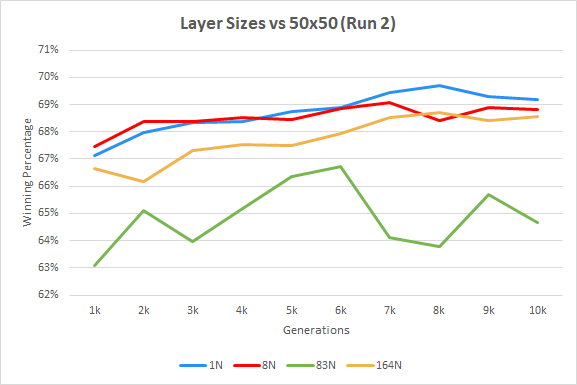
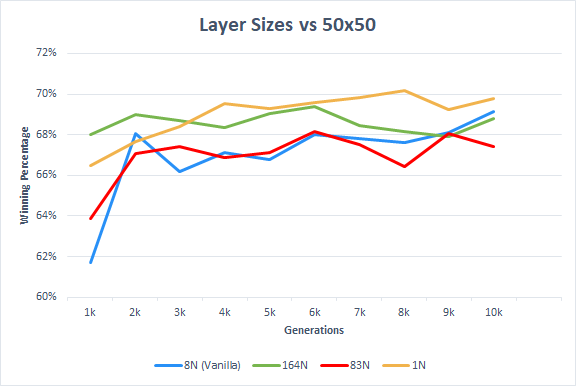
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Recitifier | Softplus | Softsign | Threshold | Sigmoid | Total Wins |
| Rectifier |  | 9347 | 9850 | 10849 | 12940 | 42986 |
| Softplus | 9913 |  | 9979 | 10790 | 13280 | 43962 |
| Softsign | 9416 | 9275 |  | 10368 | 9859 | 38918 |
| Threshold | 8445 | 8462 | 8890 |  | 9663 | 35460 |
| Sigmoid | 6528 | 6194 | 9459 | 9651 |  | 31832 |

*Neural Network Shapes*

Despite having varying numbers of layers and neurons, most of these variations have similar projections. When the 10,000 generations were played against each other, differences were more evident. 1 Neuron performed the best, and with the exception of variant 10, larger neural networks did poorer than small neural networks. This could mean large neural networks need more time to be trained, since they have more weights that need to be optimized. The relationship between weights is also more complex, another reason more generations may be required.

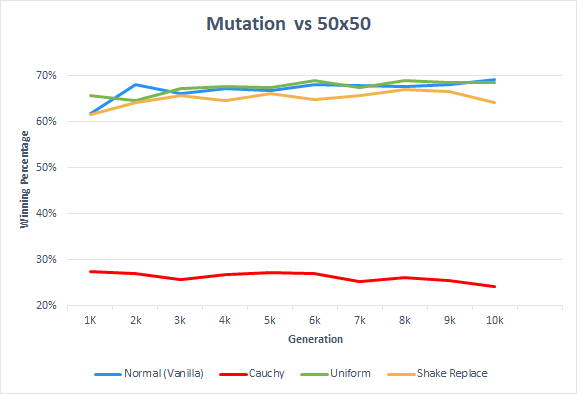
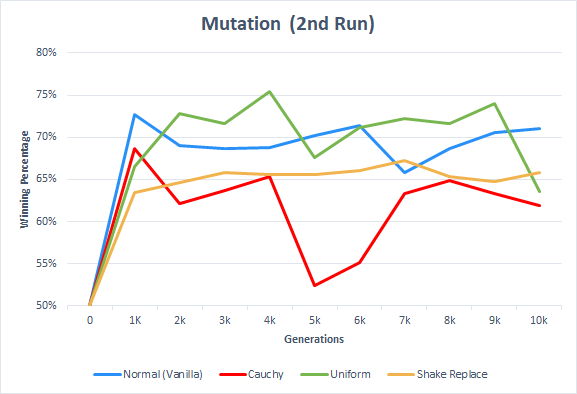
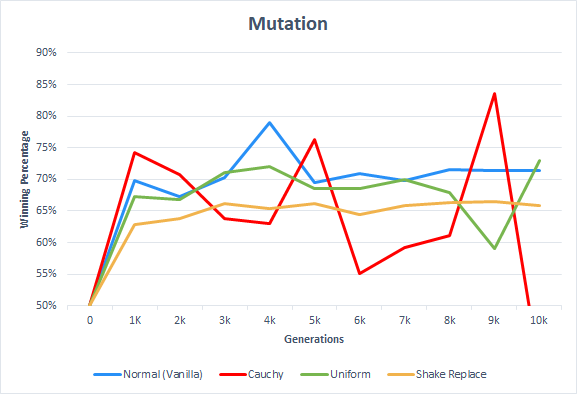
Within the baseline comparison for run 2, variant 9 did remarkably poorer than run 1. It could indicate that the initial population of the evolution was a poor starting position. Another explanation could be that it was unable to find any local maximum given the mutations that occurred along the way. Running these experiment parameters for additional runs could help to explain the sharp drop in performance.





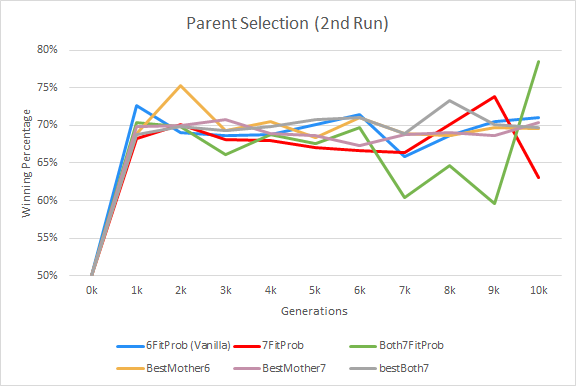
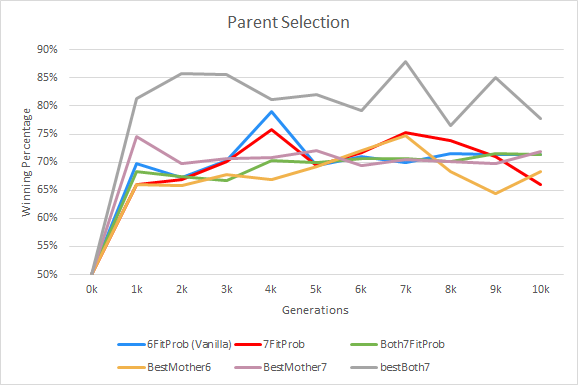
*Mutation Types*

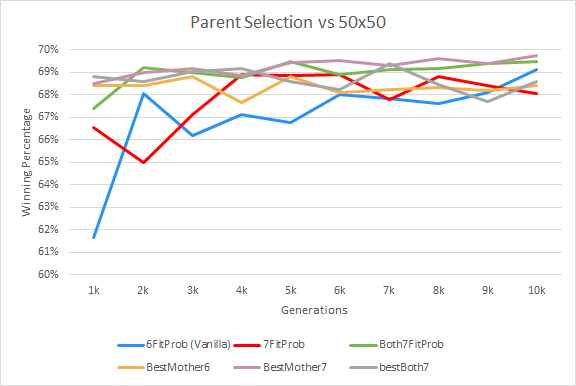
At first glance, it is clearly evident the Cauchy shake is the most volatile mutation type. This could mean that the distribution’s probability of producing exceptionally large or small values is too great. The only difference between variant 7 and the vanilla normal shake is the replacement mutation occuring with a 1% probability. It seems to make the generations perform more poorly in relation to their first generation. When the final generations were played against each other, Cauchy’s performance was remarkably poor. It did not come close to beating any other population and lost horribly against the baseline population. However, Cauchy’s final population was the worst generation across the board. There could be some interesting ways to harness Cauchy’s abilities. The 3 peaks in its performance might mean it is capable of reaching more local optima throughout its evolution.



**Parent Selection**

Overall, most of the variations of parent selection seem to perform similarly, with the exception of variant 15. However, when the experiments were run a separate time from scratch, Best Both 7 was not an outlier. Using the baseline population also indicated the population was not as skilled as the first run graph suggests.





**Conclusions**

Given our style of input selection, the best activation function to use for this situation was softplus. According to our findings, additional layers do not increase performance in the first 10,000 generations, and it could indicate that additional generations are needed for larger networks to reach their optimum levels of play. It seems that selecting a distribution with skinnier tails rather than fat tails performs better. We also discovered that a replacement mutation hurts performance overall. Parent selection does not seem to be a major factor for performance within the evolutions.

**Future Work**

There are obviously many combinations of settings which were not analyzed by this research project and there are many other settings which could be varied. Any combination not analyzed by this project would produce interesting results. Settings which were not mentioned could include weight initialization for the first generation of organisms. Other styles of crossover could be produced, such as swapping sections of genes rather than individual genes. Another interesting type of mutation could be swapping two weights within a neural network. Perhaps another type of fitness function would be more successful. Possibilities are potentially effectively endless in this regard.

It is possible for the value of a tile to have a different importance for the black player or white player. In our current configuration, a neural network considers the value of a tile for black to be the same value, but negative, for white. It could be beneficial to allow 128 inputs rather than 64 inputs for our neural network.

One flaw of our fitness function is that it does not take the strength of the surrounding neighbors into account. An organism could have neighbors who play poorly and his fitness score could be quite high. However another organism could have lost many games due to the strength of his neighbors, but his fitness score would be quite low. We currently have no way to balance results based on the skill of the neighbors. In order to overcome this problem, varying subdivisions of the population could help. At each new generation, the subdivisions would change. Each organism would play each other organism within a population. If subdivisions from one generation to another were observed at the same time, the subdivisions would overlap. This could help equalize the playing field over time.

Currently, the organisms do not seem to be capable of learning large amounts of spatial information of the board. As Chong et al. discovered, it is quite difficult to achieve master levels of play without a spatial preprocessing layer. Adding this layer could strengthen the skill of our players by a large margin.

More combinations of the settings we have

Critters - subdividing the population into different subdivisions each generation. They all play each other, completely fair comparison per generation. Maybe mention the downside to an organism playing really strong neighbors vs organisms playing very weak players. Hard coding one neuron agent with symmetrical inputs. Spatial preprocessing layer.

**Works Cited**

Tuponja, Boris, Šuković, Goran. (2016, November 22-23) Evolving Neural Network to Play Game 2048. *24th Telecommunications Forum TELFOR 2016.* Retrieved July 2018, from [www.IEEExplore.ieee.org](http://www.ieeexplore.ieee.org): <https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=7818911>

Chong, Siang, Tan, Mei, & White, Jonathon. (2005, June). Observing the Evolution of Neural Networks Learning to Play the Game Othello. *IEEE Transactions on Evolutionary Computation, Volume 9* (Issue 3). Retrieved July 2018, from [www.IEEExplore.ieee.org](http://www.ieeexplore.ieee.org): <https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=1438400>

Festa, Jacopo, Davino, Stanislao. (2013) *Iago vs Othello. An Artificial Intelligence Agent Playing Reversi.* Retrieved July 2018, from www.ceur-ws.org: <http://ceur-ws.org/Vol-1107/paper2.pdf>

Fogel, David. (2001, October). Blondie24

Shahzad, Basit, Alssum, Lolowah, & Al-Ohali, Yousef. (2012). Selection of Suitable Evaluation Function Based on Win/Draw Parameter in Othello. *Ninth International Conference on Information Technology - New Generations.* Retrieved July 2018, from [www.IEEExplor.ieee.org](http://www.ieeexplor.ieee.org): <https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=6209067>

Reasonably recent book about neural nets to cite a few details

Conclusions maybe if we ran these longer we would get better results

Future work running things longer.