**Abstract**

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

**Introduction**

*Reversi Introduction*

The game Reversi, also commonly known as Othello, is two-player, zero-sum game. It is played on an 8x8 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. This structure is capable of approximating nonlinear, multivariate functions.

*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. 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. Using these fitness values, parents will be selected and crossover between their genes will occur. Genes may correspond to weights of a neural network. 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.

**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 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, 1 representing an enemy tile, an empty tile, and a friendly tile respectively. The highest heuristic value corresponds to the best move to chose. Similarly, poor move choices will have low heuristic values. Neural networks differ by their assigned or evolved weights.

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. 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 x generations

For each organism

play 6 neighbors

Select parents using fitness score

Crossover parent genes

Mutate genes

create a new organism using genes

currentPopulation = newPopulation

If (x % 1000)

Evaluate population

repeat.

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.

**Related Works**

The complexity of Reversi is higher than checkers, but lower than chess[]. Due to this fact, Reversi is the topic of many research endeavors. Chong et al. used a very similar approach in most respects. The 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.[] Our approach choses to exclude this type of layer, in order to determine how much spatial information can be learned by a basic neural network. Festa and Davino proved fairly strong play can be achieved by using a minimax algorithm with a strong heuristic function.[] STATE WHETHER MINIMAX WAS INCLUDED OR NOT. Boris and Goran applied a similar strategy to 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 this paper, was found to achieve the highest level of play. []

**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 8x8. 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. 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 10x10 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.

The activation function is a basic variable feature. Neural networks will utilize rectifier, softsign, softplus, 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.

We will evaluate several types of mutation for the crossover stage of evolution. A “uniform mutation” will add a random number, generated using a uniform distribution between and . A “Cauchy mutation” also adds a random number, but 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, replacing it with a randomly generated number using the same uniform distribution mentioned previously.

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, its 65 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 couldl 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 has also been chosen as a variable to consider. “Mother 7 Fit Prob” chooses the mother organism from 7 neighbors, rather than 6, based on probabilities directly proportional to their fitness scores. The father will be the center organism.

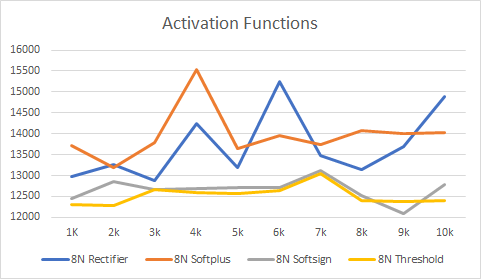
///leaving this old version of variations here incase it is better than the new one

Basic variations include using rectifier, softsign, sigmoid, or threshold for the activation function. The shape of neural networks will also vary. A single neuron, hidden layers of 8 and 3, and hidden layers of 16 and 4, will be tested and compared. Mutation variations include using a uniform or cauchy distribution, rather than the default normal distribution.

Parent selection has also been chosen as a variable. The mother organism may be chosen from 7 neighbors, rather than 6, with the father in the center. Another configuration features both parents being chosen from the 7 options, using a weighted fitness probability. The final three variations for parent selection are deterministic. One instance always selects the mother with the best fitness score from 6 neighbors, with the father in the center. Another slight variation is choosing the best mother from 7 neighbors, again with the father in the center. The last variation includes choosing the 2 best organisms from the 7 options.

**Activation Functions**

Populations using softplus, softsign, sigmoid, rectifier, and threshold were evolved for 10,000 generations using an 8 neuron network. Games were played between each agent in every 1,000th generation and the initial generation. Threshold appears to be the least successful as most of its generations had the lowest number of games won. Softsign’s trajectory was quite similar to threshold. The results of rectifier seem to be quite volatile, and fluctuate by large margins. If the activation functions were ranked by wins for each generation, softplus was consistently either first or second. It also had the highest number of total wins with 139,723 wins. Due to these properties, softplus was chosen as our vanilla configuration activation function.



**Neural Network Shapes**

Single Neuron

8 Neurons

8 - 3 Neurons

16 - 4 Neurons

**Mutation Types**

Uniform

Uniform Shake

Cauchy Shake

**Parent Selection**

>Mother 6 Fitness Probability

Mother 7 Fitness Probability

Both 7 fitness probability

Best mother 6

Best mother 7

Best both 7

Variations on vanilla

Activation functions

Mutations

Parent selection

Introduction

Gameplay

<http://ceur-ws.org/Vol-1107/paper2.pdf>

Minimax only, no neuralnet or evolution

Festa and Davino proved fairly strong play can be achieved by using a minimax algorithm with a strong heuristic function.[]

<https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=1438400>

Observing the Evolution of Neural Networks Learning to Play the Game of Othello

Minimax, neuralnet, even cites blondie24, **spatial preprocessing layer,** self adaptive gaussian mutation

Even got their players to master level

Chong et al used a very similar approach in most respects. The 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. []

“Evolving multilayer neural networks for othello”

<https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=7818911>

Neuroevolution

Applied a similar strategy to 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 evolutive process. []This approach would be an interesting technique to explore in future work.

<https://ieeexplore-ieee-org.easydb.angelo.edu/stamp/stamp.jsp?tp=&arnumber=6209067>

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 this paper, was found to achieve the highest level of play. []

Just first name last name. Not reverse

Maybe use acm on citation machine

Very brief overview of genetic algorithm, neural network, using weights as genes. Feedforward multilayer

A little pseudocode for the genetic algorithm maybe

Reasonably recent book about neural nets to cite a few details

>Neural net

>Gen alg no pseudo

>our project specific details

Related works

Our approach include gen alg pseudo

A collection of genes is an organism

In this research project, we will focus on feed forward ANN’s.