GO ARTIFICIAL INTELLIGENCE

A SCALABLE EVOLUTIONARY APPROACH

By

Warren Duncan Fraser

[illegible]

Dr. Brian Hay
Advisory Committee Chair

[illegible]

Dr. Orion Lawlor
Advisory Committee

[illegible]

Dr. Jon Genetti
Advisory Committee
Chair, Department of Computer Science

April 25, 2016

Date

GO ARTIFICIAL INTELLIGENCE

A SCALABLE EVOLUTIONARY APPROACH

A

PROJECT

Presented to the faculty of the Department of Computer Science

of the University of Alaska Fairbanks

Submitted in partial satisfaction of

the requirements for the degree of

MASTER OF SCIENCE

By

Warren Duncan Fraser, B.S.

Fairbanks, AK

May 2016

Abstract

This report covers scaling neural networks for training Go artificial intelligence. The Go board is broken up into subsections, allowing for each subsection to be calculated independently, and then factored into an overall board evaluation. This modular approach allows for subsection networks to be translated to larger board evaluations, retaining knowledge gained. The methodology covered shows promise for significant reduction in training times required for unsupervised training of Go AI.

A brief history of artificial neural networks and an overview of Go and the specific rules that were used in this project are presented. Experiment design and results are presented, showing a promising proof of concept for reducing training time required for evolutionary Go AI.

The codebase for the project is Apache 2.0 licensed and is available on GitHub.

https://github.com/wduncanfraser/scalable_go/

Acknowledgements

I would like to thank Dr. Brian Hay for being both my undergraduate and graduate advisor, providing guidance throughout my education in the UAF Computer Science department.

I would like to thank Dr. Brandon Marken, who first introduced me to AI, and has been an invaluable source of advice throughout previous research into checkers AI and the design and analysis of this project.

I would also like to thank Dr. Orion Lawlor for teaching me about computer architecture and parallelization, which has been invaluable in both my academic and professional careers. He has also been an excellent source of advice and guidance.

Table of Contents

Signature Page i

Title Page ii

Abstract iv

Acknowledgements v

Table of Contents vi

List of Figures viii

Chapter 1: Introduction 1

1.1 Artificial Neural Networks 2

1.2 The Game of Go 4

1.3 Previous Work 7

1.4 Scaling Knowledge, Reducing Training Time 8

Chapter 2: Experiment Setup and Implementation 9

2.1 Library Implementation 9

2.1.1 NeuralNet 9

2.1.2 GoGame 11

2.1.3 GoGameNN 14

2.1.4 GoGameAB 16

2.2 Experiment Setup 17

2.2.1 Training Method 17

2.2.2 Scaling Training 18

2.2.3 Training Sets for Comparison 19

2.2.4 Comparing Methodologies 20

Chapter 3: Results 21

3.1 Training Results 21

3.2 Scaling to 19x19 22

Chapter 4: Conclusions and Future Work 23
4.1 Scaling Performance 23
4.2 Scaling to 19x19 23
4.3 Future Work and Lessons Learned 24
Bibliography 26
Appendix A – Experiment Set Details 28
A.1 Scaled Divergent Sets 28
A.2 Scaled Uniform Sets 28
A.3 Control Sets 29
Appendix B – Experiment Results 30
B.1 Scaled Divergent vs Control 30
B.2 Scaled Uniform vs Control 37
B.3 Scaled Uniform vs Scaled Divergent 45
Appendix C – Code Repository 54

List of Figures

Figure 1: McCulloch-Pitts Neuron 2

Figure 2: Perceptron Activation 3

Figure 3: Perceptron 3

Figure 4 - Feed Forward Network 4

Figure 5 - Sigmoid Activation 4

Figure 6 - Stone Liberties 5

Figure 7 - String Capture 5

Figure 8 - Eyes and Dead Strings 6

Figure 9 - Benchmark Specifications 9

Figure 10 - Neuralnet Constructor 9

Figure 11 - Neuralnet Initialization 9

Figure 12 - Integer Double Conversion 10

Figure 13 - Neuralnet File Output 10

Figure 14 - GoGame Make_Move 12

Figure 15 - generate_moves Pseudocode 12

Figure 16 - check_move Pseudocode 13

Figure 17 - construct_string Pseudocode 13

Figure 18 - calculate_scores Pseudocode 14

Figure 19 - GoGameNN Constructor 14

Figure 20 - Subsection Board 15

Figure 21 - Layer 1 Networks 15

Figure 22 - Layer 2 Network 15

Figure 23 - Alpha Beta Pruning 17

Figure 24 - Scaled Training Walkthrough 18

Figure 25 - Comparison Output 20

Figure 26 - Training Times 21

Figure 27- 19x19 Board Dimensions 22

Chapter 1: Introduction

Go has always been an exceptional game when examined mathematically, and has been a hotbed of artificial intelligence research for decades. Traditional approaches to classical game AI, such as those used in Checkers and Chess do not work well in Go, due to the large amount of possible moves and the complexity of the search tree. You can easily look 8-10 moves ahead with a checkers AI with a fairly basic heuristic such as piece count, but this is simply not feasible on a full sized 19x19 Go board. Each piece on a Go board has 3 possible states: black, white, or empty, which results in 3^{19} (or 1.74×10^{172}) possible board states. It is estimated that approximately 1.2% of board positions on a 19x19 board are legal, resulting in 2.08×10^{170} legal positions [1]. For comparison, the observable universe is estimated to contain approximately 10^{80} atoms [2].

The computationally worst case move of any Go game is the first move. On a 19x19 board, there are 361 possible moves to examine. The first layer of a search tree examines 360 moves for each of the initial 361, or 129,960 total board positions. Even on a smaller 9x9 Go board, a search tree of 2 ply examines 81x80x79 or 511,920 total board positions. Ply is a measure of how many moves one looks ahead from the current move. A 0 ply search tree would directly evaluate the possible moves for a given player, with no look ahead. A search tree of any significant depth, similar to the depth possible in a checkers AI, is simply not possible. To look ahead 8 moves on a 19x19 board for the first move, an AI must evaluate 9.42×10^{22} board positions. As one is not able to feasibly build a deep search tree, one must rely on a powerful heuristic in order to produce an AI of any competent play skill.

One such approach to a heuristic would be an Artificial Neural Network (ANN). There are many different types of ANNs, with both supervised and unsupervised learning. Supervised learning typically involves teaching a network desired behavior by giving it expert knowledge and reinforcing the desired traits. Unsupervised learning typically involves randomly generating networks and using a fitness algorithm to determine the effectiveness of the networks. The top performing networks are then selected and are put through further iterations of mutations, evolving towards the desired performance vector.

One of the primary issues with evolutionary ANNs is that the more complex the network becomes, the more nodes there are to mutate. This results in very long training time requirements as the network grows (such as what would be required for a full 19x19 Go game). If Go AI could be trained similarly to how human Go players learn, one could begin by training on computationally less expensive smaller boards and translate that knowledge to larger boards once the AI had reached a desired performance level. This approach would allow reduction of the total training time required by reducing the training time required on more complex networks.

This project uses unsupervised, evolutionary ANNs to create a Go artificial intelligence, focusing on the ability to scale knowledge from smaller Go boards. This project shows that training times can be reduced by scaling knowledge.

1.1 Artificial Neural Networks

Artificial Network Networks (ANNs) are modeled after biological neural networks, attempting to mimic the way that a brain functions. They are widely used throughout the computer science discipline. They are most commonly used in machine learning and artificial intelligence.

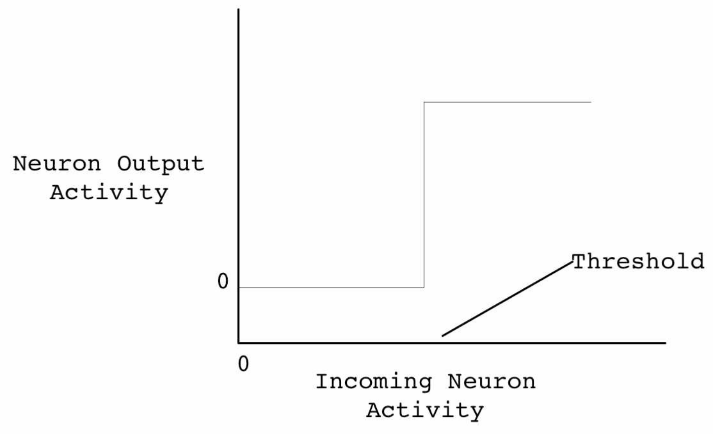


FIGURE 1: MCCULLOCH-PITTS NEURON

The history of ANNs dates back to 1943 when Warren McCulloch and Walter Pitts first published an article outlining the creation of an artificial neuron [3]. They understood that a real neuron “fires” when a stimulus excited it beyond a certain threshold. They created a highly idealized artificial neuron that took on one of two states: resting, where the stimulus level had not been reached and there was no output, and active, where the input activity exceeded the set limit and the neuron "fires." This is, however, a very limited model, and there is not much that can be modeled with a single Neuron.

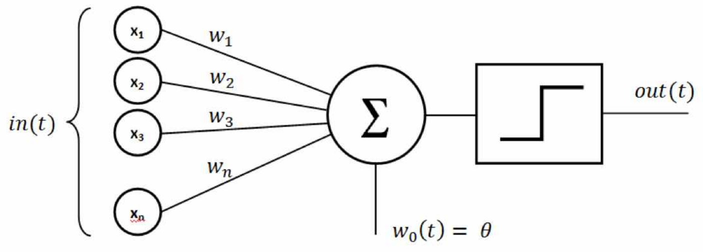


FIGURE 3: PERCEPTRON

SOURCE: HTTPS://GITHUB.COM/CDIPAOLO/GOML/TREE/MASTER/PERCEPTRON

The next major advance in ANNs was in 1958 when Frank Rosenblatt created the perceptron [4]. The perceptron is a single layer network based on neurons, originally intended to be trained for pattern recognition. The perceptron took an arbitrary amount of weighted inputs and produced a single binary output. The output was determined by summing the inputs multiplied by their corresponding weights. If the resulting value was greater than the set threshold of θ, then the perceptron "fires," returning 1. Otherwise, the perceptron returns 0. Rosenblatt was the first person to introduce the concept of weights, which allowed for the classification of inputs based on their importance to the output. The Perceptron initially showed a lot of promise and was able to map basic logic gates such as AND and OR.

$$output = \begin{cases} 0\ if \sum_j w_i x_j \leq \theta \\ 1\ if \sum_j w_i x_j > \theta \end{cases}$$

FIGURE 2: PERCEPTRON ACTIVATION

ANN research began to stagnate in 1969 when Marvin Minksy and Seymour Papert published a paper describing two keys issues with perceptrons [5]. The first was that perceptrons were incapable of modeling XOR logic gates. The second was that computers at the time did not have the processing power required to effectively compute large neural networks. These two findings contributed to what is known as the AI winter, where very little funding was put towards AI research for years to come.

In 1974 Paul Werbos developed the backpropagation algorithm which effectively solved the problem with ANNs not being able to properly model XOR logic gates [6]. Additionally, it provided a method for quickly performing supervised learning of multi-layer ANNs. The backpropagation algorithm was one of the key advances in ANNs that would eventually lead to interest and resurgence, and it is still a very popular method today for training large ANNs.

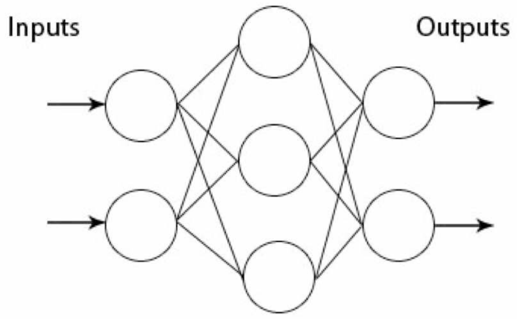


FIGURE 4 - FEED FORWARD NETWORK

Significant resurgence and renewed interest in ANNs began in the mid to late 2000s with the advent of deep learning. One of the most common types of ANNs today is a fully connected feedforward network. Typically, nodes in a feedforward network use a sigmoid activation function. The output of a node is determined by summing the weights multiplied by their respective inputs, and passing the summed value through the activation function. Between 2009 and 2012, the deep feedforward neural networks developed by a research group at the Swiss AI Lab IDSIA have won eight international competitions in pattern recognition and machine learning [7].

$$S(t) = \frac{1}{1 + e^{-t}}$$

FIGURE 5 - SIGMOID ACTIVATION

On the front of unsupervised learning, genetic algorithms can be used to solve complex problems when the answer is not known. David Fogel developed a checkers AI program called Blondie 24 that learned to play purely through a genetic algorithm, with no knowledge of how to play other than the rules themselves [8]. This was accomplished by randomly generating neural networks, and mutating them through generations using a fitness heuristic that measured the performance of the networks playing against each other. Blondie24 was able to play competitively against expert checkers players and against the best checkers AI at the time, Chinook. The genetic algorithm used by David Fogel is the foundation on which this project's training algorithm is based.

1.2 The Game of Go

The game of Go is a perfect information, deterministic, zero-sum game of strategy between two players. In game theory, perfect information means that for any given move, the player has complete knowledge of every move that has happened prior, and that they have no less knowledge than they would at the end of the game [9]. A zero-sum game is one in which each player's gain or loss is exactly balanced by the losses or gains of the other participants [10].

The game of Go originated somewhere in China between 3,000 – 4,000 years ago. While the exact origin of the game is unknown, it spread throughout Asia developing varying rulesets over the years. The two most significant rule variations center around the scoring methods, and whether scoring is based on area occupied or territory surrounded. For this project, territory scoring rules were used. Since there are so many variations in the rules of Go, the following rule descriptions describe the exact rules that were chosen for this project implementation. These rules are largely based on traditional Japanese territory scoring [11].

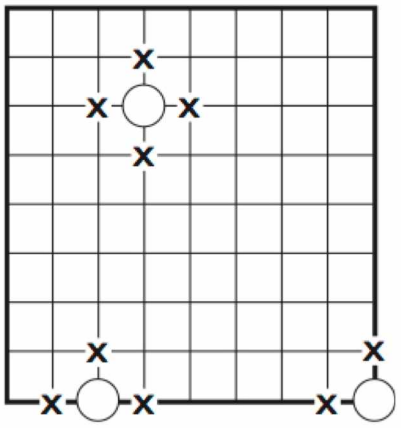


FIGURE 6 - STONE LIBERTIES

The game of Go starts with an empty board of a given size. Beginners typically play on 9x9 boards. Experienced players typically play short games on 13x13 boards, and full length games are played on a 19x19 board. Each player has a sufficient (in this case, unlimited) amount of stones to play the game to completion.

Black always takes the first move. Players take turns, placing one of their stones on a vacant intersection of the board's grid lines. The empty spaces adjacent to a stone are known as liberties. As seen in figure 6, the stone at the top has 4 liberties, while the stone in the bottom left has 3 liberties, and the stone in the bottom right has 2 liberties. If a stone ever loses all of its liberties, it is removed from the board and given to the other player as a prisoner.

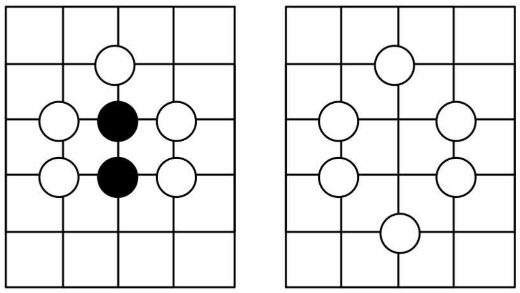


FIGURE 7 - STRING CAPTURE

In Go, adjacent stones of the same color form strings. Liberties for a string are calculated by summing all empty adjacent pieces to the string's member stones. Just like individual stones, a string is captured when its last liberty is removed. An example of this can be found in Figure 7. The black string contains two members and has a single liberty. When white occupies the last empty intersection, the black string is captured and the pieces given to white as prisoners.

There are some stipulations to the legality of any given move. No move can be made which recreates a prior board position. Additionally, no move may be made which leads to self-capture or suicide of stones.

A player may pass at any point during the game. If a player passes, they must give a single stone to the opponent as a prisoner. The game ends when both players pass in successive turns. Since black played first, the game must end with white passing.

In Go, strings have a concept of eyes. If a string has two eyes (two independent blank spaces within the string), it is impossible for the string to be captured. In Figure 8, if black were to place a stone on position "o," the resultant string would have two eyes and would be impossible to capture. Typically, hopeless or dead strings are removed from the board at the end of the game by consensus. A dead string is one in which it is impossible to form two eyes. Again in Figure 8, if black were to play a stone at position "p" or "q," it would be impossible for the string to form two eyes [12].

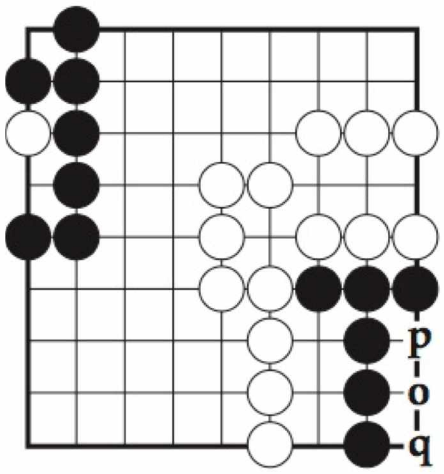


FIGURE 8 - EYES AND DEAD STRINGS

SOURCE: HTTP://WWW.BRITGO.ORG/INTRO/INTRO2.HTML

For the Go rules used in this project, there is no automatic removal of dead strings at the end of the game. For example, the white piece in the upper left of figure eight would typically be removed as a dead string as it has no hope of preventing capture. Typically, if there is a disagreement in the removal of strings, the game continues to resolve the disagreement. Since I am working with ANNs, all dead strings must be played out. If dead strings are left on the board they will affect the final score.

1.3 Previous Work

Go has been the subject of mathematical and artificial intelligence research for decades. Brute force methodologies and simple heuristics (such as piece counts) simply do not work well due to the scale and complexity of Go. Early approaches to Go AI usually focused on specific features or functions of the game, which resulted in very weak performing Go AI [13] [14]. Early Go AI could typically be defeated by intermediate or beginner level players, even when the Go AI was given bigger handicaps than any human player would ever accept.

Successful Go AI has largely been dependent on domain knowledge of Go, using expert move sets to train the computer Go players. Even then, until very recently, no Go AI has ever been able to beat an expert level player without handicap.

The most recent and significant example of Go AI is Google's AlphaGo. AlphaGo is the first Go AI to beat a master level player with no handicap [15]. AlphaGo uses a combination of machine learning and tree search techniques. AlphaGo uses a Monte Carlo tree search, and was initially trained using supervised learning with a massive database of around 30 million moves from recorded historical games. AlphaGo was then trained against itself with unsupervised learning, further evolving the AI. Early versions of AlphaGo used large computer clusters, with up to 64 search threads, 1,920 CPUs, and 280 GPUs [16]. AlphaGo varies greatly from my approach to Go AI. I am focusing on scaling knowledge using genetic algorithms, while AlphaGo was designed to become the best Go player possible, using high performance compute clusters and extensive human expert knowledge in supervised training.

In 2008, Lin Wu and Pierre Baldi published the paper "Learning to play Go using recursive neural networks," which covers attempting to write a Go artificial intelligence that can transfer knowledge from smaller board sizes to larger boards, such as from 9x9 to 19x19 [17]. They found encouraging results, however their research was largely dependent on human expert knowledge, using supervised training from move datasets.

In 2010, Jason Gauci and Kenneth O. Stanley published the paper "Indirect encoding of neural networks for scalable Go," which focuses on encoding Go boards indirectly [18]. They focused on allowing training to be conducted on smaller board sizes and scaled to larger boards, without relying on processing subsections of the board. This differs greatly from my approach, as I am focusing on transferring trained subsections of a board to larger network sizes, reducing the training time required for subsection weighting and evaluation. Additionally, they were focusing on using Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT). NEAT is different from the evolutionary approach that I took, as it mutates the structure and complexity of the network in addition to the weights and connections.

To reduce the computational time required for large Go board sizes, Gauci and Stanley chose to focus on building an action selector that evaluates current state and suggests where to move, rather than executing a board evaluation function against possible moves in a search tree. Their training consisted of training a 5x5 board for 500 generations against a fixed policy player. After training on a 5x5 Go board, the domain is switched to playing Go against the same policy on a 7x7 board. This use of outside knowledge differs from my approach, as I am using a purely genetic algorithm that plays randomly generated networks against themselves. Gauci and Stanley then compared the scaled networks against networks that had only played on a 7x7 board. They found that the scaled networks performed better, and concluded that scaling Go players is definitely a possibility.

1.4 Scaling Knowledge, Reducing Training Time

Go is a fairly uniform game, and concepts that apply to smaller boards usually hold true on larger boards. Additionally, human players typically look at subsections of the board, where specific territories are being formed. This is exemplified in the different sizes of Go boards used in training and short duration play.

It takes much longer to train neural networks for larger Go board sizes, so there would be significant benefit in being able to train on much less computationally expensive smaller boards, and transfer that knowledge to larger boards. While there has been some research done into scaling ANNs for Go, I wanted to focus on a purely genetic/evolutionary approach, with no domain knowledge of the game, to see if scaling boards using subsections could significantly reduce the overall training time.

Chapter 2: Experiment Setup and Implementation

2.1 Library Implementation

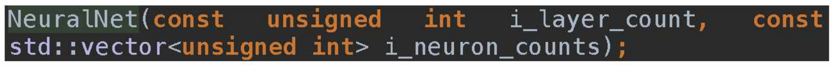
In order to properly carry out this experiment, four libraries were written that were used for training and testing the neural networks: NeuralNet, GoGame, GoGameNN, and GoGameAB. The entire codebase for this Go AI project was written in C++, to allow for low level performance gains and easy parallelization of training and gameplay via OpenMP and MPI. All benchmark values in the following sections are run on a 2015 MacBook Pro.

CPU: Intel Core i7, Quad Core 2.5ghz
RAM: 16GB
OS: OS X 10.11.4
Compiler: GCC 5.3

FIGURE 9 - BENCHMARK SPECIFICATIONS

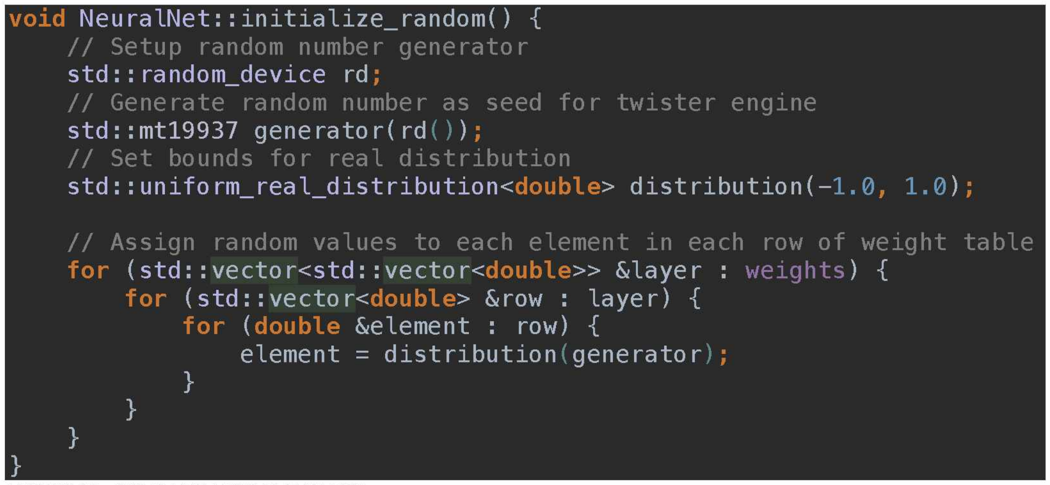
2.1.1 NeuralNet

For the first library, NeuralNet, I began by examining previous work I had done with checkers AI, and generalizing the neural network code into general a purpose neural network library. This neural network library is the foundation of the Go ANN heuristic, allowing the creation of a feed forward neural network of arbitrary size. A bias node is added for each layer except the output layer of the neural network. This network can either be loaded from file or initialized randomly.



```
NeuralNet(const unsigned int i_layer_count, const std::vector<unsigned int> i_neuron_counts);
```

FIGURE 10 - NEURALNET CONSTRUCTOR



```
void NeuralNet::initialize_random() {
    // Setup random number generator
    std::random_device rd;
    // Generate random number as seed for twister engine
    std::mt19937 generator(rd());
    // Set bounds for real distribution
    std::uniform_real_distribution<double> distribution(-1.0, 1.0);

    // Assign random values to each element in each row of weight table
    for (std::vector<std::vector<double>> &layer : weights) {
        for (std::vector<double> &row : layer) {
            for (double &element : row) {
                element = distribution(generator);
            }
        }
    }
}
```

FIGURE 11 - NEURALNET INITIALIZATION

The random generation of the weights is implemented using a uniform real distribution over the range of (-1.0, 1.0). Mutation is carried out in a very similar manner. A radius is specified as a parameter to the mutate function, and all weights in the NeuralNet instance are modified randomly over a uniform real distribution with a range of (-radius, radius).

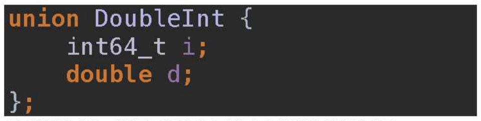
Currently, the NeuralNet library only supports feedforward operations. I chose not to implement any other functionality, as this project purely uses evolutionary neural networks. With the current implementation, it would not be possible to conduct supervised learning. Input layer values are passed as a parameter to the feedforward function. The feedforward function propagates through each layer calculating the value of each node by summing each input linked node multiplied by its associated weight. Once all input nodes have been summed into the node, the activation function is called against the node. The output layer can be retrieved by calling the *get_output* method.

Currently, the activation function $\frac{x}{1+|x|}$ is used for all neural networks. Additional activation functions could be added easily, with a constructor parameter determining what activation function to use. This was outside the scope of what was required for this experiment.

The NeuralNet library allows for outputting and reading from file using *ofstream* and *ifstream.* The first value output is the layer count, followed by a newline. The next line contains the neuron counts for each layer, separated by commas. The third line is all of the weights, output as comma separated values. The first two lines inform the network how many weights there are, and what format to expect them in when importing. Weights are stored in ASCII when output to file, and floats/doubles are truncated on output. To prevent data loss from truncation and rounding, a union of *double* and *int64_t* is used in order to output the exact value of the weight.

```
4
9,12,3,1,
4598683231414027791,…,4604503844013429472,
```

FIGURE 13 - NEURALNET FILE OUTPUT



```
union DoubleInt {
    int64_t i;
    double d;
};
```

FIGURE 12 - INTEGER DOUBLE CONVERSION

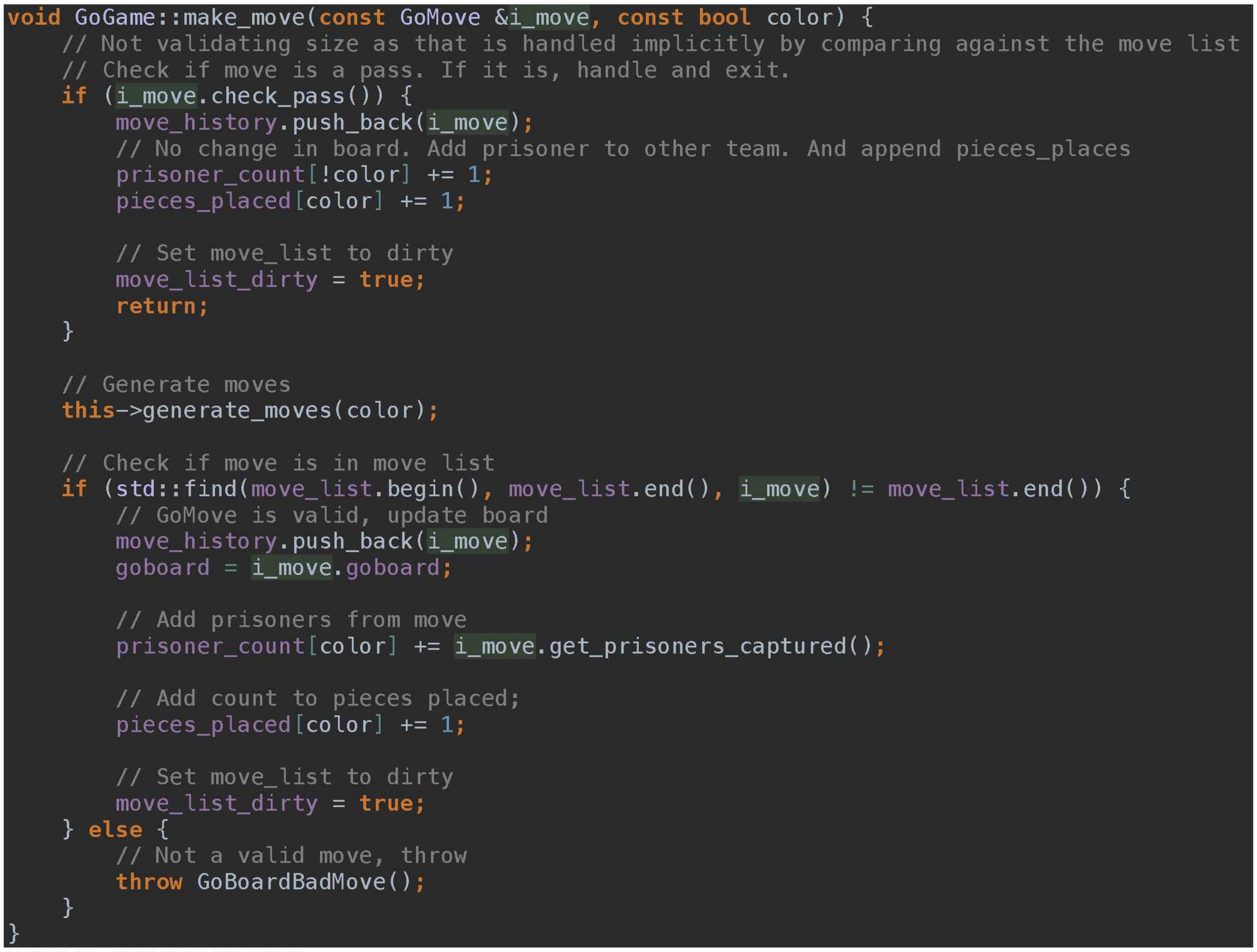
Unit testing is included with the project to ensure that there is no difference in networks that have been written to file and reimported, as different compilers and platforms treat unions of variable types differently.

Basic benchmarks of the NeuralNet class were performed against a model network from previous checkers AI work. The network had 4 layers with 32 input nodes, 40 nodes in the first hidden layer, 10 nodes in the second hidden layer, and a single output node. On my reference system, the network was able to perform approximately 2 million board operations per second.

2.1.2 GoGame

The GoGame library consists of multiple class definitions in order to provide a complete interface for playing a game of Go. The Go board is stored as a two dimensional vector of unsigned 8 bit integers, with the bottom left of the board designated as the origin (0,0), and board elements can be accessed via *board[y][x]*. Each element of the board has three primary states, and a fourth used for scoring. A blank space is represented as a 0, with 0 bits set. The first bit of any given element is used to determine if a piece is present. The second bit is used to determine the team (0 for black, 1 for white). As such, a black piece has a mask of 1 and a white piece has a mask of 3. The third bit is used when scoring to determine if a given space has been scored yet.

The GoGame instance keeps track of the current Go board, a list of all possible moves for the current board state (as well as Booleans for determining which team the moves were generated for, and whether the move list is dirty and needs to be regenerated), a history of all moves that have been made (used for preventing recreation of prior board states), prisoner counts, and pieces placed count. No direct access to the board is allowed. All moves are validated by generating a list of all possible moves, and verifying that the submitted move is a member of the move list.



```cpp
void GoGame::make_move(const GoMove &i_move, const bool color) {
    // Not validating size as that is handled implicitly by comparing against the move list
    // Check if move is a pass. If it is, handle and exit.
    if (i_move.check_pass()) {
        move_history.push_back(i_move);
        // No change in board. Add prisoner to other team. And append pieces_places
        prisoner_count[!color] += 1;
        pieces_placed[color] += 1;

        // Set move_list to dirty
        move_list_dirty = true;
        return;
    }

    // Generate moves
    this->generate_moves(color);

    // Check if move is in move list
    if (std::find(move_list.begin(), move_list.end(), i_move) != move_list.end()) {
        // GoMove is valid, update board
        move_history.push_back(i_move);
        goboard = i_move.goboard;

        // Add prisoners from move
        prisoner_count[color] += i_move.get_prisoners_captured();

        // Add count to pieces placed;
        pieces_placed[color] += 1;

        // Set move_list to dirty
        move_list_dirty = true;
    } else {
        // Not a valid move, throw
        throw GoBoardBadMove();
    }
}
```

FIGURE 14 - GOGAME MAKE_MOVE

The move generation for Go is fairly involved, as you must examine all empty spaces on the board and determine if it is legal to place a piece. For any given move to be valid, you must ensure that the move does not cause a suicide (create a string with a liberty of 0) and that you are not creating a previous board state. The move generation algorithm can be found in Figure 15.

- Retrieves the board size and loops through all grid elements.
- For each element, check if the space is occupied. If occupied, continue to next element, otherwise:
    - Create a potential GoMove instance with the placed piece.
    - Determine the impact of the move on the board by calling GoMove member function *check_move*.
    - Ensure the move is not a suicide.
    - Ensure the board state hasn't existed before.
    - If all criteria has been met, append to the move list.

FIGURE 15 - GENERATE_MOVES PSEUDOCODE

As part of the move generation algorithm, each potential empty space is passed to an instance of GoMove. GoMove tracks the

```
• Place the specified piece on the board.
• Treat all pieces as potential strings.
• Looks at adjacent pieces.
    o If adjacent piece is the same team, append it to the string.
    o If adjacent piece is unoccupied, add it as a liberty to the string.
    o If adjacent piece is an enemy, create a new enemy string for it.
• Check impact on adjacent enemy strings (call construct_string on each: if liberty = 0, remove string from board).
• Check impact on own string.
• Return liberty count of own string.
```

FIGURE 16 - CHECK_MOVE PSEUDOCODE

resultant game board, coordinates for the piece placed, as well as a count of the prisoners captured, and a Boolean to determine if the move is a pass. Once a potential GoMove instance is created, the *check_move* function is called to determine the impact of the placed piece on the board. The *check_move* function first checks the impact of the move on enemy strings. If any enemy strings' liberties are reduced to 0, they are removed from the board. Once the impact on the enemy player's strings has been calculated, the impact on the player's string is determined and the liberty of the player's string is returned. The *check_move* pseudocode can be found in Figure 16.

```
• Maintains a list of elements to check (elements already in string list).
• For each element in check list, looks at all adjacent pieces.
    o If friendly and not already in the string, add to string and check list.
    o If empty and not already in liberties, add to liberties.
    o Otherwise, skip.
• Once checked, remove element from check list.
• When no more elements are left to check, return completed string.
```

FIGURE 17 - CONSTRUCT_STRING PSEUDOCODE

Next, all the partial strings created by the *check_move* function must be completed to determine their impact. The *construct_string* method takes a partially completed string and finds all string members and liberties. The *construct_string* pseudocode can be found in Figure 17.

The combination of the *generate_moves* and *make_move* functions allow for any arbitrary Go game to be played on any size board. The GoGame implementation was made as modular and generic as possible to allow for the library to be reused for any future work with Go.

The final step for any Go game is to calculate the score. As I am using territory scoring with prisoner counts, each blank string on the board must be examined. If the blank string is bordered by a single color, the string piece count is added to that team's score. If the string is bordered by both teams, it is neutral territory and counts for neither team. After all territory has been calculated, prisoner counts are added to each team's score. Full pseudocode for the *calculate_scores* function can be found in Figure 18.

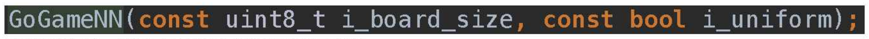
```
•  Retrieves the board size and loops through all grid
   elements.
•  For each element, check if it is a blank space, and if it
   hasn't been scored yet.
      o  Construct a string of all blank spaces
         adjacently connected to the initial space.
      o  Check if the string is bordered by one or both
         players.
      o  If bordered by 1 player, append string space
         count to that players score. If not, neutral
         territory.
      o  Mark the string as scored.
•  Once bored has been checked. Append prisoner
   counts to each player score.
```

FIGURE 18 - CALCULATE_SCORES PSEUDOCODE

The current iteration of the GoGame library was written to ensure correct operation, and there are many areas in which efficiency can be improved. However, this implementation is for a proof of concept, and none of the efficiencies to be gained would increase the performance of the AI by orders of magnitude.

2.1.3 GoGameNN

The GoGameNN library acts as a wrapper around the NeuralNet library, specializing it for Go AI with subsection boards for this specific project. The GoGameNN constructor takes two arguments: the board size and a Boolean specifying whether the neural network is using a uniform configuration. On creation, the GoGameNN constructor generates 2 layers of neural networks. The first layer is a vector of NeuralNet instances, one for each subsection of the Go board. Using 5x5 board as an example, there would be 9 3x3 subsection networks and 1 5x5 subsection network. If the Boolean for uniform is true, there is only one subsection network for each subsection size. The input node count



```
GoGameNN(const uint8_t i_board_size, const bool i_uniform);
```

FIGURE 19 - GOGAMENN CONSTRUCTOR

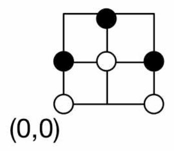


FIGURE 20 - SUBSECTION BOARD

equals the total board coordinates for any given subsection. The board coordinates for any given subsection are serialized and fed into the input nodes of any given subsection. The layer 1 network dimensions were based on my previous checkers AI, which in turn was based on David Fogel's Blondie24 [8]. With checkers, there are 32 input nodes: one for each playable position on the board. The first hidden layer was assigned 40 nodes, the second hidden layer 10 nodes, and a single output node. These node counts were chosen fairly arbitrarily as a proof of concept, and I have continued that trend. I generalized and rounded the counts for the layer 1 networks to the first hidden layer being 4/3 the count of the input nodes, rounded down. The second hidden layer is 1/4 the first hidden layer, rounded down. Similar to the checkers network, there is a single output node for each subsection network.

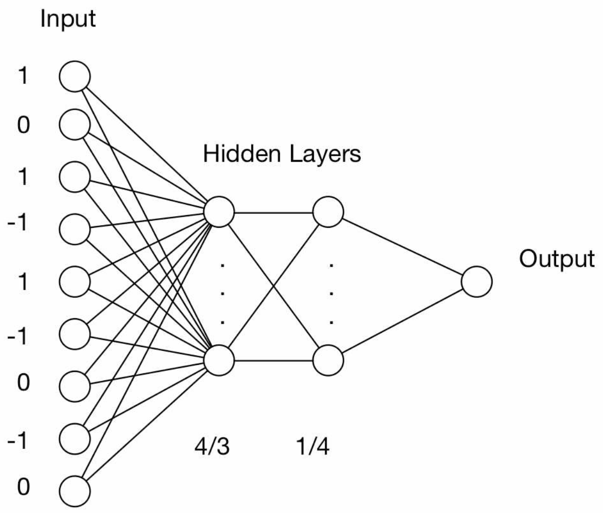


FIGURE 21 - LAYER 1 NETWORKS

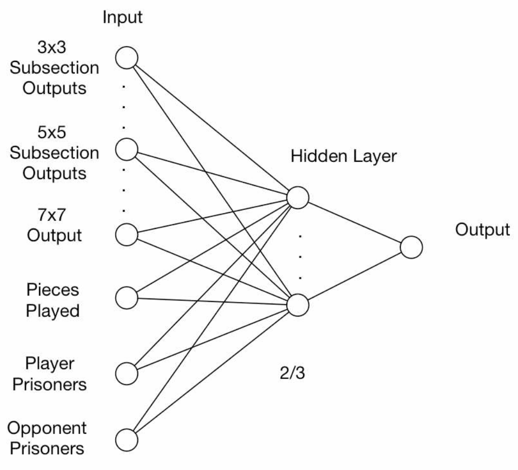


FIGURE 22 - LAYER 2 NETWORK

The layer 2 network takes the outputs from each of the layer 1 networks and 3 additional values: pieces played count, player prisoner count, and opponent prisoner count as inputs. The 3 additional inputs are normalized against $\frac{board\ size^2}{2}$, which equates to half the potential board positions. This normalization is done to keep all inputs between -1 and 1, preventing saturation of the node activation functions. The layer 2 network has a single hidden layer, and one output node. The hidden layer contains 2/3 the input nodes, rounded down. This value is based loosely on an ANN rule of thumb, that a good starting point for hidden layer node counts is $(input\ node\ count + output\ node\ count) \times \frac{2}{3}$. The single output node is used as a heuristic for determining the desirability of any given board state.

The GoGameNN class contains wrappers for the NeuralNet methods *initialize_random*, *mutate*, *feed_forward*, import and export from file, and *get_output*.

Additionally, the GoGameNN class has a method for scaling networks from a smaller board size to the next board size called *scale_network*. *Scale_network* takes a single parameter: another instance of GoGameNN. The specified network must be exactly 1 size smaller than the new GoGameNN. The layer 1 subsection networks are taken from the existing network and used to seed the subsections of the new network. A new network for the largest subsection is randomly generated. Additionally, a new layer 2 network is generated to accommodate the larger layer 1 network vector. A current limitation of the GoGameNN implementation is that when scaling up larger board sizes where there are multiple subsections of a given size in the existing network (for example, 3x3 subsections when scaling from a 5x5 board to a 7x7 board), the subsections are chosen at random to seed the new network. A side or corner subsection in Go is vastly different from center subsections, with different desired piece configurations. A future task would be to rewrite this logic to identify side and corner subsections, differentiating them from central subsections, and seeding the new network appropriately. This was largely out of scope for this specific implementation, as it does not affect the 3x3 to 5x5 scaling.

The GoGameNN library on my reference system is capable of calculating approximately 36,000 9x9 Go boards per second.

2.1.4 GoGameAB

The GoGameAB library implements a single function, *scalable_go_ab_prune*. *Scalable_go_ab_prune* implements an alpha beta pruning algorithm for use with GoGameNN and GoGame libraries. Alpha beta is used to look ahead multiple moves, and minimize potential enemy positions while maximizing your own position. Alpha Beta eliminates subtrees that are not promising, reducing total calculations compared to a naïve minmax tree. The alpha beta implementation is recursive, taking the GoGameNN and GoGame instances, depth remaining, alpha and beta values, and Booleans for determining current move color, whether the current level is for the max player, and the player color.

```
double scalable_go_ab_prune(GoGameNN &network, GoGame &i_gogame, const int
depth, double alpha, double beta, const bool move_color, const bool
max_player, const bool player_color);
```

FIGURE 23 - ALPHA BETA PRUNING

The alpha beta implementation uses the GoGameNN *feed_forward* output as the heuristic for the alpha beta search tree. *Scalable_go_ab_prune* is meant to be called against each potential move in a given move list. The move given the highest value is considered to be the best move.

2.2 Experiment Setup

The main purpose of this project is to show whether training times could be improved by scaling knowledge gained on smaller Go boards up to larger sizes. As a proof of concept, training times are compared scaling from a 3x3 board up to a 5x5 board. Additional training sets were performed at 7x7 to determine the training time required, with extrapolation showing the feasibility of scaling up to full 19x19 boards.

All networks were trained with 1 ply (looking 1 move ahead from the current move). This was chosen because it allowed for some look ahead, while avoiding prohibitively long training times.

2.2.1 Training Method

This project uses an evolutionary approach to training ANNs. A genetic algorithm was chosen in order to focus on a computer teaching itself how to play Go, with no prior knowledge of the game other than the rules. This allows for easy comparison of scaled networks that are trained with the exact same rulesets. Additionally, there is not much data for doing supervised learning on boards of size 3x3 and 5x5.

The training algorithm maintains a population of 30 networks, which are initially randomly seeded. Each network in the population plays every other team round robin, playing as both white and black. A single point is awarded to a network for a win. Zero points are awarded for a draw. A network loses 1 point for a loss. While playing as each color doubles the amount of games required, it also forces the networks to be proficient at playing both colors, and exposes them to board states they would otherwise not see (black always goes first, and that offset can significantly change of the game states that a given player would see).

After all games have been played, the top 1/3 of the population is kept to seed the next generation. The other 2/3 of the networks are discarded. One third of the new generation is mutated from the top 1/3 from the previous generation. The final third of the new generation is randomly seeded. The randomly seeded networks serve the purpose of preventing the initially seeded networks from mutating into a local optimum. Additionally, the top 1/3 are maintained completely unaltered as a control set. If the mutated networks perform worse than the top 1/3, then they can be discarded and replaced with new mutations.

This loop is then repeated until the desired generation count is reached.

2.2.2 Scaling Training

In order to scale the training, a population of networks is trained on a certain board size for a specified number of generations. Once the generations are complete, the top 1/3 of the last generation is taken to seed the networks for the next size board training. The imported networks are randomly selected to seed and scale up for the new board size. The same training loop described earlier is then repeated with the new networks with one exception: once a network has been scaled up, the randomly generated networks are no longer completely random. In order to retain the

1. Take a 3x3 Board.
   - 3x3 Board has a single 3x3 subsection.
   - Layer 1 would contain one 9 input network.
   - Layer 2 would take single output from 3x3 subsection, and then piece count and prisoner counts.
2. Train networks for X generations.
3. Take the top 1/3 of networks from the last generation.
4. Take a 5x5 Board.
   - 5x5 board has nine 3x3 subsections and one 5x5 subsection.
   - Seed all 3x3 subsections from a random imported network.
   - Generate random 5x5 subsection network.
   - Regenerate Layer 2 network to account for additional inputs.
5. Train networks for X generations.

FIGURE 24 - SCALED TRAINING WALKTHROUGH

knowledge gained from the previous training sets, all randomly generated networks are scaled up from the import set, just like all networks in the first generation.

2.2.3 Training Sets for Comparison

For this project, three different types of training sets were used. The first, the control set, is a network purely trained on a certain size. The next set is a population of networks trained on smaller boards and then scaled up. The subsection boards are divergent once scaled. The final training set is identical to the second, except that the subsection networks are kept uniform: only a single subsection network is maintained for each subsection size and is used to calculate all subsections of that size.

Specifically, the training sets are as follows:

- Control – Networks trained on a 5x5 board for 500 generations.
- Scaling with Divergent Subsections – Networks trained on a 3x3 board for 500 generations, scaled and then trained on a 5x5 board for 300 generations.
- Scaling with Uniform Subsection - Networks trained on a 3x3 board for 500 generations, scaled and then trained on a 5x5 board for 300 generations.
    - Only 1 subsection board for each size.

Four sets of each type were performed in order to take an average of the performance difference between the different methodologies.

The generation counts for each training method were chosen fairly arbitrarily, estimating how many generations would be required to get networks that are no longer playing randomly, while significantly reducing the total training time for the scaled sets. Further analysis could be done to determine the exact training time gains; however, the scope of this project is to present these methods as a proof of concept.

2.2.4 Comparing Methodologies

To compare the different training methods, the final top 1/3 networks from each training set are taken and compared against all other training sets of different types. For each comparison, all networks from one set play against all networks from another set as both black and white. Similarly to how training is scored, 1 point is awarded for a win, 1 point subtracted for a loss, and a draw results in no points awarded to either team. An example of the output from these comparisons can be found in Figure 25. The maximum score that any given network can obtain in a comparison is 20 (20 games played for each network), with a maximum average score difference of 40 between two sets.

```
Comparing set 1: size5set1 against set 2: size5set10
Final scores are as follows.
Set 1 Network 0: 20. Set 2 Network 0: -11.
Set 1 Network 1: -7. Set 2 Network 1: -19.
Set 1 Network 2: 20. Set 2 Network 2: -16.
Set 1 Network 3: 20. Set 2 Network 3: -7.
Set 1 Network 4: 20. Set 2 Network 4: -15.
Set 1 Network 5: 10. Set 2 Network 5: -12.
Set 1 Network 6: -9. Set 2 Network 6: -8.
Set 1 Network 7: 10. Set 2 Network 7: -9.
Set 1 Network 8: 20. Set 2 Network 8: -8.
Set 1 Network 9: 11. Set 2 Network 9: -10.
Set 1 Network Average: 11. Set 2 Network Average: -11.
```

FIGURE 25 - COMPARISON OUTPUT

All comparisons from one type to another can then be aggregated and averaged to determine which method came out ahead, as well as the average score difference.

Chapter 3: Results

3.1 Training Results

Four sets of each training method described in section 2.2.3 were completed and compared. The exact details of all comparisons can be found in Appendix B. Training was primarily performed on 2x Google Cloud Engine 16 core compute nodes.

Total training time for the scaled network methods was 154.2 CPU hours per set. Total training time for the control sets was 244.4 CPU hours per set.

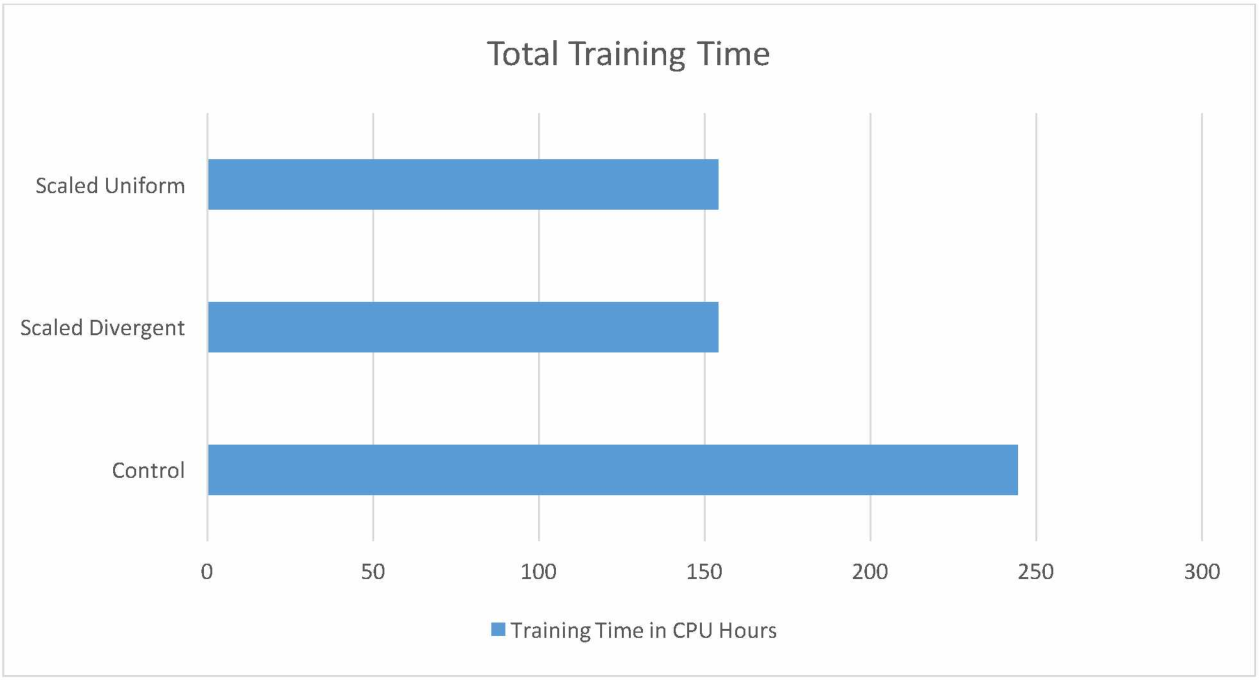


FIGURE 26 - TRAINING TIMES

Both scaled training methods outperformed the control training sets. On average, the divergent scaled networks scored 8.75 points against control, with an average score difference of 17.5. The uniform scaled networks scored 6.375 points on average against control, with an average score difference of 12.75.

The uniform scaled networks outperformed the divergent scaled networks with an average score of 2.0625 and an average score difference of 4.125.

To test training times, additional control sets were trained for 500 generations on a 7x7 board. The required training time increased to 3,795 CPU hours per set.

To perform basic real world analysis and determine if the trained networks had any reasonable ability to play Go, I wrote a client and played against it myself, and had it play against the Go AI available at http://www.cosumi.net/en/. I selected the top performing network from the divergent scaled sets. The networks were able to play competitively against me, and defeat me one game each (I am a beginner player). When playing against the Cosumi AI, the top networks tied twice and lost twice. I then played a network that had been trained for 1,000 generations at 3x3 and 1,000 generations at 5x5, and it beat the Cosumi AI by two points.

The AI demonstrated basic Go strategy, such as capturing corners, attempting to control territory, and blocking the opponent from capturing capturing territory.

3.2 Scaling to 19x19

In order to test the feasibility of a 19x19 board using this scaling method, I created a 19x19 instance of GoGameNN and GoGameAB to benchmark them. To give a sense of scale, the dimensions of a 19x19 GoGameNN can be found in Figure 26.

A GoGameNN instance at 19x19 can perform approximately 190 board evaluations per second on my reference system.

Taking the worst case move (first move of a game), the GoGameAB can evaluate moves with 0 ply (no look ahead) in approximately 2 seconds. Evaluating the first move with 1 ply takes approximately 10 minutes. Performing a 2 move look ahead or 2 ply evaluation would take approximately 3 days.

Total subsections: 968
Layer 1
- Input nodes: 52,616
- HL1 nodes: 70,154
- HL2 nodes: 17,538
- Output: 968

Layer 2
- Input: 971
- HL: 647
- Output: 1

FIGURE 27- 19X19 BOARD DIMENSIONS

Chapter 4: Conclusions and Future Work

4.1 Scaling Performance

Training comparison results show that there is definitely promise in scaling neural networks for Go AI. For this proof of concept, the average score differences show that the scaled training sets significantly outperformed the control sets, with a 37% reduction in training time required. Training times could likely be reduced even further, due to the significant performance advantage that the scaled networks had over the control sets.

Additionally, the uniform scaled networks slightly outperformed the divergent scaled networks. This is most likely due to the limited amount of generations performed on the 5x5 board, as the uniform networks have fewer nodes to mutate, allowing for a faster convergence. I would hypothesize that this would not hold true as the generation count increases, since the uniform networks would not be able to learn behaviors of the differing subsections, such as edges or corners versus positions closer to the middle of the board. I would like to further examine this behavior in future experiments.

This reduction in overall training times is very promising, considering that the resultant trained networks played competitively against myself and another Go AI. The networks showed basic awareness of the goal of Go with limited training time, looking ahead only a single move. If the training time were increased with deeper look ahead, the AI has potential to be even more competitive.

4.2 Scaling to 19x19

Benchmarks of the 19x19 GoGameAB search tree show that without significant parallelization, it would not be possible to look ahead any moves when playing on a 19x19 board with this subsection design. This loss of look ahead and the ability to build a search tree would be a significant handicap to the overall performance of the Go AI. However, if such a network had hardware backing similar to AlphaGo, it would be feasible to look ahead at least two moves. This would incrase performance beyond the 1 ply look ahead used in this project.

Additionally, the training time required for scaling from a 3x3 board up to 19x19 would be astronomical. Simply moving from a 5x5 board to 7x7 increased required CPU hours for 500 generations from 244.4 CPU hours to 3,795 CPU hours.

A basic estimate for 0 ply training on a 19x19 board would be that on average, it takes 1.5 seconds to evaluate potential moves for any given board position. We will assume that there are 200 moves made in an average game of Go. With a population of 30 networks, there are 900 games in a single generation. It would take approximately 1.5 x 200 x 900 seconds or 75 CPU hours to complete a single generation at 19x19, making training a 19x19 network very impractical without extensive parallelization.

4.3 Future Work and Lessons Learned

The biggest issue with this implementation is that it was built purely to conduct a statistical analysis of scaling Neural Networks. All initialization and mutation of the networks is conducted with completely random, unrecorded seeds. This makes it impossible to recreate the exact populations that were created during this experiment. If I had run into bugs during implementation, it would have been impossible to recreate the exact network configurations that caused the issue. I will be refactoring the NeuralNet library to seed the random devices with identifying values, which will allow for recreation of the training sets if required.

There are many optimizations to be had in the current implementation of this scaling Go AI. One of the most obvious areas would be examining the usefulness of having every single odd numbered board subsection from 3x3, 5x5, 7x7, up the 19x19 in a final 19x19 network. It would definitely speed up training and overall performance if certain subsections were found to be less useful and could be removed from the layer 1 neural networks. Additionally, there are a few cases where lists of moves are searched linearly during move generation and evaluation. I would like to refactor these sections of the code to use a standard map, or abstract the moves into another more efficient data structure. One idea for such a data structure would be to store the move list in a static two dimensional array that matches the board size. Each move would be stored at the coordinates of the piece placed. This would allow for constant time lookups of any given move.

The final item that I would like to implement would be to train a scaled 9x9 network with 2 ply and examine how it performs in real world situations, comparing it against experienced humans and other AI players. This experiment shows there is promise in scaling networks trained with the genetic algorithm used in this project with some basic real world analysis; however, there is no measure of how it performs on board sizes commonly used by real players.

Bibliography

[1] J. Tromp and G. Farnebäck, "Combinatorics of Go," in *Computers and Games, 5th Internation Conference*, Turin, Italy, 2006.

[2] WolframAlpha, "estimated number of atoms in the universe," [Online]. Available: http://www.wolframalpha.com/input/?i=atoms+universe. [Accessed 21 April 2016].

[3] W. McCulloh and W. Pitts, "A logicial calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics,* vol. 5, no. 4, pp. 115-133, 1943.

[4] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Pysochological review,* vol. 65, no. 6, p. 386, 1958.

[5] M. Minksy and S. Papert, Perceptron: an introduction to computational geometry, Cambridge, MA: The MIT Press, 1969.

[6] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," 1974.

[7] D. J. Schmidhuber, Interviewee, *How bio-inspired deep learning keeps winning competitions.* [Interview]. 28 November 2012.

[8] D. B. Fogel, Blondie24: Playing at the Edge of AI, San Francisco, CA: Morgan Kaufmann, 2002.

[9] Wikipedia, "Perfect Information," 29 February 2016. [Online]. Available: https://en.wikipedia.org/wiki/Perfect_information. [Accessed 16 April 2016].

[10] Wikipedia, "Zero-sum game," 11 April 2016. [Online]. Available: https://en.wikipedia.org/wiki/Zero-sum_game. [Accessed 16 April 2016].

[11] British Go Association, "A Brief History of Go," British Go Association, 7 March 2016. [Online]. Available: http://www.britgo.org/intro/history. [Accessed 17 April 2016].

[12] British Go Association, "How to Play," British Go Association, 21 March 2016. [Online]. Available: http://www.britgo.org/intro/intro2.html. [Accessed 17 April 2016].

[13] J. Burmeister and J. Wiles, "Technical Report 339," 1995. [Online]. Available: http://staff.itee.uq.edu.au/janetw/Computer%20Go/CS-TR-339.html. [Accessed 22 April 2016].

[14] N. Wedd, "Human-Computer Go Challenges," 23 March 2016. [Online]. Available: http://www.computer-go.info/h-c/index.html. [Accessed 22 April 2016].

[15] S. Byford, "Google's DeepMind defeats legendary Go player Lee Se-dol in historic victory," 9 March 2016. [Online]. Available: http://www.theverge.com/2016/3/9/11184362/google-alphago-go-deepmind-result. [Accessed 19 April 2016].

[16] D. Silver and A. Huang, "Mastering the game of Go with deep neural networks and tree search," *Nature,* vol. 529, p. 484–489, 28 January 2016.

[17] L. Wu and P. Baldi, "Learning to play Go using recursive neural networks," *Neural Networks,* vol. 21, no. 9, pp. 1392-1400, November 2008.

[18] J. Gauci and K. O. Stanley, "Indirect Encoding of Neural Networks for Scalable Go," in *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature*, New York, 2010.

Appendix A – Experiment Set Details

A.1 Scaled Divergent Sets

Size 3 Set 1 Specs: dc-nerv01, 500 Generations, Elapsed: 1771.33s

Size 5 Set 1 Specs: dc-nerv01, 300 Generations, Elapsed: 54546.5s

Size 3 Set 2 Specs: dc-nerv01, 500 Generations, Elapsed: 1648.72s

Size 5 Set 2 Specs: compute-1, 300 Generations, Elapsed: 32575.3s

Size 3 Set 3 Specs: dc-nerv01, 500 Generations, Elapsed: 1798.21s

Size 5 Set 3 Specs: compute-1, 300 Generations, Elapsed: 30196.2s

Size 3 Set 4 Specs: dc-nerv01, 500 Generations, Elapsed: 1601.6s

Size 5 Set 4 Specs: compute-1, 300 Generations, Elapsed: 33586.3s

A.2 Scaled Uniform Sets

Size 3 Set 5 Specs: duplicate of set 1, uniform

Size 5 Set 5 Specs: dc-nerv01, 300 generations, uniform, Elapsed: 58605.8s

Size 3 Set 6 Specs: duplicate of set 2, uniform

Size 5 Set 6 Specs: compute-2, 300 generations, uniform, Elapsed: 34590.6s

Size 3 Set 7 Specs: duplicate of set 3, uniform

Size 5 Set 7 Specs: compute-2, 300 generations, uniform, Elapsed: 31546.5s

Size 3 Set 8 Specs: duplicate of set 4, uniform

Size 5 Set 8 Specs: compute-2, 300 generations, uniform, Elapsed: 29684.2s

A.3 Control Sets

Size 5 Set 9 Specs: compute-1, 500 generations, Elapsed: 54940.1s

Size 5 Set 10 Specs: compute-2, 500 generations, Elapsed: 55321.9s

Size 5 Set 11 Specs: compute-1, 500 generations, Elapsed: 54568.8s

Size 5 Set 12 Specs: compute-2, 500 generations, Elapsed: 49466.7s

Appendix B – Experiment Results

B.1 Scaled Divergent vs Control

```
Comparing set 1: size5set1 against set 2: size5set9

Final scores are as follows.

Set 1 Network 0: 2. Set 2 Network 0: 12.

Set 1 Network 1: -6. Set 2 Network 1: 3.

Set 1 Network 2: -4. Set 2 Network 2: -14.

Set 1 Network 3: 2. Set 2 Network 3: -7.

Set 1 Network 4: 2. Set 2 Network 4: -7.

Set 1 Network 5: 4. Set 2 Network 5: -9.

Set 1 Network 6: -3. Set 2 Network 6: 0.

Set 1 Network 7: 2. Set 2 Network 7: 7.

Set 1 Network 8: 4. Set 2 Network 8: -3.

Set 1 Network 9: 11. Set 2 Network 9: 4.

Set 1 Network Average: 1. Set 2 Network Average: -1.


Comparing set 1: size5set1 against set 2: size5set10

Final scores are as follows.

Set 1 Network 0: 20. Set 2 Network 0: -11.

Set 1 Network 1: -7. Set 2 Network 1: -19.

Set 1 Network 2: 20. Set 2 Network 2: -16.

Set 1 Network 3: 20. Set 2 Network 3: -7.

Set 1 Network 4: 20. Set 2 Network 4: -15.

Set 1 Network 5: 10. Set 2 Network 5: -12.

Set 1 Network 6: -9. Set 2 Network 6: -8.

Set 1 Network 7: 10. Set 2 Network 7: -9.

Set 1 Network 8: 20. Set 2 Network 8: -8.

Set 1 Network 9: 11. Set 2 Network 9: -10.
Set 1 Network Average: 11. Set 2 Network Average: -11.


Comparing set 1: size5set1 against set 2: size5set11
Final scores are as follows.
Set 1 Network 0: 12. Set 2 Network 0: -16.
Set 1 Network 1: 6. Set 2 Network 1: -2.
Set 1 Network 2: 13. Set 2 Network 2: -18.
Set 1 Network 3: 12. Set 2 Network 3: -12.
Set 1 Network 4: 12. Set 2 Network 4: -17.
Set 1 Network 5: 12. Set 2 Network 5: -18.
Set 1 Network 6: 7. Set 2 Network 6: -15.
Set 1 Network 7: 11. Set 2 Network 7: -8.
Set 1 Network 8: 14. Set 2 Network 8: -6.
Set 1 Network 9: 15. Set 2 Network 9: -2.
Set 1 Network Average: 11. Set 2 Network Average: -11.


Comparing set 1: size5set1 against set 2: size5set12
Final scores are as follows.
Set 1 Network 0: 16. Set 2 Network 0: -14.
Set 1 Network 1: -8. Set 2 Network 1: 0.
Set 1 Network 2: 14. Set 2 Network 2: -12.
Set 1 Network 3: 18. Set 2 Network 3: -12.
Set 1 Network 4: 14. Set 2 Network 4: -14.
Set 1 Network 5: 18. Set 2 Network 5: -4.
Set 1 Network 6: -10. Set 2 Network 6: -12.
Set 1 Network 7: 16. Set 2 Network 7: -14.
Set 1 Network 8: 18. Set 2 Network 8: -14.
Set 1 Network 9: 12. Set 2 Network 9: -12.
Set 1 Network Average: 10. Set 2 Network Average: -10.
```

```
Comparing set 1: size5set2 against set 2: size5set9
Final scores are as follows.
Set 1 Network 0: 8. Set 2 Network 0: -10.
Set 1 Network 1: 20. Set 2 Network 1: -10.
Set 1 Network 2: 5. Set 2 Network 2: -14.
Set 1 Network 3: 6. Set 2 Network 3: -10.
Set 1 Network 4: -2. Set 2 Network 4: -2.
Set 1 Network 5: 8. Set 2 Network 5: -12.
Set 1 Network 6: 12. Set 2 Network 6: -12.
Set 1 Network 7: 6. Set 2 Network 7: -6.
Set 1 Network 8: 16. Set 2 Network 8: -18.
Set 1 Network 9: 20. Set 2 Network 9: -5.
Set 1 Network Average: 9. Set 2 Network Average: -9.

Comparing set 1: size5set2 against set 2: size5set10
Final scores are as follows.
Set 1 Network 0: 16. Set 2 Network 0: -12.
Set 1 Network 1: 12. Set 2 Network 1: -14.
Set 1 Network 2: 20. Set 2 Network 2: -10.
Set 1 Network 3: 14. Set 2 Network 3: -18.
Set 1 Network 4: 20. Set 2 Network 4: -10.
Set 1 Network 5: 20. Set 2 Network 5: -18.
Set 1 Network 6: 12. Set 2 Network 6: -18.
Set 1 Network 7: 8. Set 2 Network 7: -18.
Set 1 Network 8: 20. Set 2 Network 8: -18.
Set 1 Network 9: 12. Set 2 Network 9: -18.
Set 1 Network Average: 15. Set 2 Network Average: -15.
```

```
Comparing set 1: size5set2 against set 2: size5set11

Final scores are as follows.

Set 1 Network 0: 3. Set 2 Network 0: -18.

Set 1 Network 1: 16. Set 2 Network 1: -18.

Set 1 Network 2: 18. Set 2 Network 2: -18.

Set 1 Network 3: 13. Set 2 Network 3: -6.

Set 1 Network 4: 14. Set 2 Network 4: -14.

Set 1 Network 5: 12. Set 2 Network 5: -6.

Set 1 Network 6: 12. Set 2 Network 6: -6.

Set 1 Network 7: 11. Set 2 Network 7: -16.

Set 1 Network 8: 12. Set 2 Network 8: -17.

Set 1 Network 9: 16. Set 2 Network 9: -8.

Set 1 Network Average: 12. Set 2 Network Average: -12.


Comparing set 1: size5set2 against set 2: size5set12

Final scores are as follows.

Set 1 Network 0: 18. Set 2 Network 0: -20.

Set 1 Network 1: 18. Set 2 Network 1: -20.

Set 1 Network 2: 16. Set 2 Network 2: -20.

Set 1 Network 3: 18. Set 2 Network 3: -6.

Set 1 Network 4: 20. Set 2 Network 4: -20.

Set 1 Network 5: 18. Set 2 Network 5: -20.

Set 1 Network 6: 18. Set 2 Network 6: -18.

Set 1 Network 7: 20. Set 2 Network 7: -20.

Set 1 Network 8: 18. Set 2 Network 8: -18.

Set 1 Network 9: 18. Set 2 Network 9: -20.

Set 1 Network Average: 18. Set 2 Network Average: -18.


Comparing set 1: size5set3 against set 2: size5set9
```

```
Final scores are as follows.

Set 1 Network 0: 2. Set 2 Network 0: 4.

Set 1 Network 1: -14. Set 2 Network 1: 0.

Set 1 Network 2: -10. Set 2 Network 2: 6.

Set 1 Network 3: -6. Set 2 Network 3: -2.

Set 1 Network 4: 7. Set 2 Network 4: 0.

Set 1 Network 5: -6. Set 2 Network 5: 11.

Set 1 Network 6: 5. Set 2 Network 6: -4.

Set 1 Network 7: -2. Set 2 Network 7: 11.

Set 1 Network 8: -20. Set 2 Network 8: 4.

Set 1 Network 9: 0. Set 2 Network 9: 14.

Set 1 Network Average: -4. Set 2 Network Average: 4.


Comparing set 1: size5set3 against set 2: size5set10

Final scores are as follows.

Set 1 Network 0: 13. Set 2 Network 0: 0.

Set 1 Network 1: -9. Set 2 Network 1: 4.

Set 1 Network 2: -1. Set 2 Network 2: -1.

Set 1 Network 3: 10. Set 2 Network 3: -4.

Set 1 Network 4: -7. Set 2 Network 4: -2.

Set 1 Network 5: 15. Set 2 Network 5: -6.

Set 1 Network 6: -2. Set 2 Network 6: 2.

Set 1 Network 7: -2. Set 2 Network 7: 6.

Set 1 Network 8: -8. Set 2 Network 8: -3.

Set 1 Network 9: -6. Set 2 Network 9: 1.

Set 1 Network Average: 0. Set 2 Network Average: 0.


Comparing set 1: size5set3 against set 2: size5set11

Final scores are as follows.
```

```
Set 1 Network 0: 2. Set 2 Network 0: 15.

Set 1 Network 1: 7. Set 2 Network 1: -3.

Set 1 Network 2: -7. Set 2 Network 2: 6.

Set 1 Network 3: 2. Set 2 Network 3: 2.

Set 1 Network 4: -10. Set 2 Network 4: -6.

Set 1 Network 5: 2. Set 2 Network 5: 6.

Set 1 Network 6: -11. Set 2 Network 6: 2.

Set 1 Network 7: -13. Set 2 Network 7: 9.

Set 1 Network 8: 2. Set 2 Network 8: 0.

Set 1 Network 9: -8. Set 2 Network 9: 3.

Set 1 Network Average: -3. Set 2 Network Average: 3.


Comparing set 1: size5set3 against set 2: size5set12

Final scores are as follows.

Set 1 Network 0: -9. Set 2 Network 0: 5.

Set 1 Network 1: -10. Set 2 Network 1: 2.

Set 1 Network 2: 6. Set 2 Network 2: 0.

Set 1 Network 3: -9. Set 2 Network 3: 5.

Set 1 Network 4: 5. Set 2 Network 4: 6.

Set 1 Network 5: -9. Set 2 Network 5: 1.

Set 1 Network 6: 3. Set 2 Network 6: 8.

Set 1 Network 7: 5. Set 2 Network 7: -7.

Set 1 Network 8: -14. Set 2 Network 8: 10.

Set 1 Network 9: 5. Set 2 Network 9: -3.

Set 1 Network Average: -2. Set 2 Network Average: 2.


Comparing set 1: size5set4 against set 2: size5set9

Final scores are as follows.

Set 1 Network 0: 20. Set 2 Network 0: -20.
Set 1 Network 1: 20. Set 2 Network 1: -20.

Set 1 Network 2: 20. Set 2 Network 2: -20.

Set 1 Network 3: 20. Set 2 Network 3: -20.

Set 1 Network 4: 20. Set 2 Network 4: -20.

Set 1 Network 5: 20. Set 2 Network 5: -20.

Set 1 Network 6: 20. Set 2 Network 6: -20.

Set 1 Network 7: 20. Set 2 Network 7: -20.

Set 1 Network 8: 20. Set 2 Network 8: -20.

Set 1 Network 9: 20. Set 2 Network 9: -20.

Set 1 Network Average: 20. Set 2 Network Average: -20.


Comparing set 1: size5set4 against set 2: size5set10

Final scores are as follows.

Set 1 Network 0: 10. Set 2 Network 0: -20.

Set 1 Network 1: 10. Set 2 Network 1: -20.

Set 1 Network 2: 20. Set 2 Network 2: -20.

Set 1 Network 3: 10. Set 2 Network 3: -4.

Set 1 Network 4: 10. Set 2 Network 4: -20.

Set 1 Network 5: 10. Set 2 Network 5: -4.

Set 1 Network 6: 20. Set 2 Network 6: -4.

Set 1 Network 7: 10. Set 2 Network 7: -20.

Set 1 Network 8: 10. Set 2 Network 8: -4.

Set 1 Network 9: 10. Set 2 Network 9: -4.

Set 1 Network Average: 12. Set 2 Network Average: -12.


Comparing set 1: size5set4 against set 2: size5set11

Final scores are as follows.

Set 1 Network 0: 11. Set 2 Network 0: -18.

Set 1 Network 1: 16. Set 2 Network 1: -20.
Set 1 Network 2: 16. Set 2 Network 2: -14.

Set 1 Network 3: 14. Set 2 Network 3: -20.

Set 1 Network 4: 20. Set 2 Network 4: -6.

Set 1 Network 5: 15. Set 2 Network 5: -10.

Set 1 Network 6: 11. Set 2 Network 6: -10.

Set 1 Network 7: 13. Set 2 Network 7: -20.

Set 1 Network 8: 11. Set 2 Network 8: -12.

Set 1 Network 9: 17. Set 2 Network 9: -14.

Set 1 Network Average: 14. Set 2 Network Average: -14.


Comparing set 1: size5set4 against set 2: size5set12

Final scores are as follows.

Set 1 Network 0: 16. Set 2 Network 0: -18.

Set 1 Network 1: 14. Set 2 Network 1: -20.

Set 1 Network 2: 18. Set 2 Network 2: -17.

Set 1 Network 3: 18. Set 2 Network 3: -20.

Set 1 Network 4: 16. Set 2 Network 4: 0.

Set 1 Network 5: 18. Set 2 Network 5: -18.

Set 1 Network 6: 18. Set 2 Network 6: -14.

Set 1 Network 7: 18. Set 2 Network 7: -20.

Set 1 Network 8: 18. Set 2 Network 8: -18.

Set 1 Network 9: 8. Set 2 Network 9: -17.

Set 1 Network Average: 16. Set 2 Network Average: -16.
```

B.2 Scaled Uniform vs Control

```
Comparing set 1: size5set5 against set 2: size5set9

Final scores are as follows.

Set 1 Network 0: 2. Set 2 Network 0: -4.
Set 1 Network 1: 20. Set 2 Network 1: -4.

Set 1 Network 2: 1. Set 2 Network 2: -4.

Set 1 Network 3: 0. Set 2 Network 3: -4.

Set 1 Network 4: 4. Set 2 Network 4: -4.

Set 1 Network 5: 2. Set 2 Network 5: -4.

Set 1 Network 6: 4. Set 2 Network 6: -4.

Set 1 Network 7: 2. Set 2 Network 7: -9.

Set 1 Network 8: 16. Set 2 Network 8: -17.

Set 1 Network 9: 4. Set 2 Network 9: -1.

Set 1 Network Average: 5. Set 2 Network Average: -5.


Comparing set 1: size5set5 against set 2: size5set10

Final scores are as follows.

Set 1 Network 0: -3. Set 2 Network 0: -13.

Set 1 Network 1: 12. Set 2 Network 1: -13.

Set 1 Network 2: 10. Set 2 Network 2: -20.

Set 1 Network 3: 7. Set 2 Network 3: 4.

Set 1 Network 4: 9. Set 2 Network 4: -13.

Set 1 Network 5: -1. Set 2 Network 5: 6.

Set 1 Network 6: 12. Set 2 Network 6: 2.

Set 1 Network 7: 6. Set 2 Network 7: -16.

Set 1 Network 8: 12. Set 2 Network 8: 4.

Set 1 Network 9: 13. Set 2 Network 9: -18.

Set 1 Network Average: 7. Set 2 Network Average: -7.


Comparing set 1: size5set5 against set 2: size5set11

Final scores are as follows.

Set 1 Network 0: 12. Set 2 Network 0: 0.

Set 1 Network 1: 12. Set 2 Network 1: -20.
Set 1 Network 2: 12. Set 2 Network 2: -20.

Set 1 Network 3: 10. Set 2 Network 3: 0.

Set 1 Network 4: 9. Set 2 Network 4: -20.

Set 1 Network 5: 12. Set 2 Network 5: 0.

Set 1 Network 6: 12. Set 2 Network 6: -20.

Set 1 Network 7: 12. Set 2 Network 7: 0.

Set 1 Network 8: 10. Set 2 Network 8: -13.

Set 1 Network 9: 12. Set 2 Network 9: -20.

Set 1 Network Average: 11. Set 2 Network Average: -11.


Comparing set 1: size5set5 against set 2: size5set12

Final scores are as follows.

Set 1 Network 0: 20. Set 2 Network 0: -18.

Set 1 Network 1: 20. Set 2 Network 1: -20.

Set 1 Network 2: 17. Set 2 Network 2: -20.

Set 1 Network 3: 20. Set 2 Network 3: -15.

Set 1 Network 4: 14. Set 2 Network 4: -20.

Set 1 Network 5: 20. Set 2 Network 5: -15.

Set 1 Network 6: 18. Set 2 Network 6: -20.

Set 1 Network 7: 20. Set 2 Network 7: -20.

Set 1 Network 8: 20. Set 2 Network 8: -20.

Set 1 Network 9: 14. Set 2 Network 9: -15.

Set 1 Network Average: 18. Set 2 Network Average: -18.


Comparing set 1: size5set6 against set 2: size5set9

Final scores are as follows.

Set 1 Network 0: 10. Set 2 Network 0: -12.

Set 1 Network 1: 11. Set 2 Network 1: -20.

Set 1 Network 2: 11. Set 2 Network 2: -14.
Set 1 Network 3: 12. Set 2 Network 3: -20.
Set 1 Network 4: 14. Set 2 Network 4: -20.
Set 1 Network 5: 14. Set 2 Network 5: -15.
Set 1 Network 6: 14. Set 2 Network 6: -20.
Set 1 Network 7: 11. Set 2 Network 7: -2.
Set 1 Network 8: 18. Set 2 Network 8: 10.
Set 1 Network 9: 18. Set 2 Network 9: -20.
Set 1 Network Average: 13. Set 2 Network Average: -13.

Comparing set 1: size5set6 against set 2: size5set10
Final scores are as follows.
Set 1 Network 0: 14. Set 2 Network 0: -20.
Set 1 Network 1: 6. Set 2 Network 1: -20.
Set 1 Network 2: 18. Set 2 Network 2: -20.
Set 1 Network 3: 18. Set 2 Network 3: -16.
Set 1 Network 4: 14. Set 2 Network 4: -20.
Set 1 Network 5: 18. Set 2 Network 5: -16.
Set 1 Network 6: 18. Set 2 Network 6: -12.
Set 1 Network 7: 10. Set 2 Network 7: 0.
Set 1 Network 8: 14. Set 2 Network 8: -16.
Set 1 Network 9: 20. Set 2 Network 9: -10.
Set 1 Network Average: 15. Set 2 Network Average: -15.

Comparing set 1: size5set6 against set 2: size5set11
Final scores are as follows.
Set 1 Network 0: 12. Set 2 Network 0: -14.
Set 1 Network 1: 15. Set 2 Network 1: -13.
Set 1 Network 2: 12. Set 2 Network 2: -8.
Set 1 Network 3: 13. Set 2 Network 3: -4.
Set 1 Network 4: 11. Set 2 Network 4: -20.

Set 1 Network 5: 13. Set 2 Network 5: -17.

Set 1 Network 6: 12. Set 2 Network 6: -18.

Set 1 Network 7: 14. Set 2 Network 7: -18.

Set 1 Network 8: 18. Set 2 Network 8: -20.

Set 1 Network 9: 18. Set 2 Network 9: -6.

Set 1 Network Average: 13. Set 2 Network Average: -13.


Comparing set 1: size5set6 against set 2: size5set12

Final scores are as follows.

Set 1 Network 0: 20. Set 2 Network 0: -20.

Set 1 Network 1: 20. Set 2 Network 1: -16.

Set 1 Network 2: 20. Set 2 Network 2: -20.

Set 1 Network 3: 20. Set 2 Network 3: -20.

Set 1 Network 4: 20. Set 2 Network 4: -20.

Set 1 Network 5: 20. Set 2 Network 5: -20.

Set 1 Network 6: 20. Set 2 Network 6: -20.

Set 1 Network 7: 20. Set 2 Network 7: -20.

Set 1 Network 8: 18. Set 2 Network 8: -20.

Set 1 Network 9: 18. Set 2 Network 9: -20.

Set 1 Network Average: 19. Set 2 Network Average: -19.


Comparing set 1: size5set7 against set 2: size5set9

Final scores are as follows.

Set 1 Network 0: -19. Set 2 Network 0: 8.

Set 1 Network 1: -14. Set 2 Network 1: 13.

Set 1 Network 2: -17. Set 2 Network 2: 16.

Set 1 Network 3: -14. Set 2 Network 3: 13.

Set 1 Network 4: 12. Set 2 Network 4: 13.
Set 1 Network 5: 6. Set 2 Network 5: 13.

Set 1 Network 6: -19. Set 2 Network 6: 13.

Set 1 Network 7: -19. Set 2 Network 7: 18.

Set 1 Network 8: -15. Set 2 Network 8: -7.

Set 1 Network 9: -17. Set 2 Network 9: 16.

Set 1 Network Average: -11. Set 2 Network Average: 11.


Comparing set 1: size5set7 against set 2: size5set10

Final scores are as follows.

Set 1 Network 0: 12. Set 2 Network 0: -11.

Set 1 Network 1: 5. Set 2 Network 1: -11.

Set 1 Network 2: 9. Set 2 Network 2: -11.

Set 1 Network 3: -13. Set 2 Network 3: 4.

Set 1 Network 4: -13. Set 2 Network 4: -5.

Set 1 Network 5: -6. Set 2 Network 5: 4.

Set 1 Network 6: 12. Set 2 Network 6: -5.

Set 1 Network 7: 12. Set 2 Network 7: -6.

Set 1 Network 8: 12. Set 2 Network 8: -1.

Set 1 Network 9: 12. Set 2 Network 9: 0.

Set 1 Network Average: 4. Set 2 Network Average: -4.


Comparing set 1: size5set7 against set 2: size5set11

Final scores are as follows.

Set 1 Network 0: -4. Set 2 Network 0: -5.

Set 1 Network 1: 8. Set 2 Network 1: 2.

Set 1 Network 2: -2. Set 2 Network 2: -2.

Set 1 Network 3: -4. Set 2 Network 3: -2.

Set 1 Network 4: 8. Set 2 Network 4: -3.

Set 1 Network 5: 6. Set 2 Network 5: 7.
Set 1 Network 6: -4. Set 2 Network 6: -4.
Set 1 Network 7: -5. Set 2 Network 7: -8.
Set 1 Network 8: 6. Set 2 Network 8: 0.
Set 1 Network 9: 5. Set 2 Network 9: 1.
Set 1 Network Average: 1. Set 2 Network Average: -1.

Comparing set 1: size5set7 against set 2: size5set12
Final scores are as follows.
Set 1 Network 0: -11. Set 2 Network 0: -6.
Set 1 Network 1: -9. Set 2 Network 1: 7.
Set 1 Network 2: -12. Set 2 Network 2: 8.
Set 1 Network 3: 7. Set 2 Network 3: -2.
Set 1 Network 4: 14. Set 2 Network 4: 8.
Set 1 Network 5: 14. Set 2 Network 5: 7.
Set 1 Network 6: -14. Set 2 Network 6: 5.
Set 1 Network 7: -13. Set 2 Network 7: -3.
Set 1 Network 8: -5. Set 2 Network 8: 8.
Set 1 Network 9: -8. Set 2 Network 9: 5.
Set 1 Network Average: -3. Set 2 Network Average: 3.

Comparing set 1: size5set8 against set 2: size5set9
Final scores are as follows.
Set 1 Network 0: -16. Set 2 Network 0: 13.
Set 1 Network 1: -13. Set 2 Network 1: 13.
Set 1 Network 2: -16. Set 2 Network 2: 10.
Set 1 Network 3: 3. Set 2 Network 3: 11.
Set 1 Network 4: -12. Set 2 Network 4: 13.
Set 1 Network 5: 7. Set 2 Network 5: 11.
Set 1 Network 6: -13. Set 2 Network 6: 12.
Set 1 Network 7: -13. Set 2 Network 7: 2.
Set 1 Network 8: -2. Set 2 Network 8: -4.
Set 1 Network 9: -13. Set 2 Network 9: 7.
Set 1 Network Average: -8. Set 2 Network Average: 8.

Comparing set 1: size5set8 against set 2: size5set10
Final scores are as follows.
Set 1 Network 0: 17. Set 2 Network 0: -7.
Set 1 Network 1: 6. Set 2 Network 1: -7.
Set 1 Network 2: 19. Set 2 Network 2: -8.
Set 1 Network 3: 16. Set 2 Network 3: -16.
Set 1 Network 4: 14. Set 2 Network 4: -4.
Set 1 Network 5: 17. Set 2 Network 5: -20.
Set 1 Network 6: 9. Set 2 Network 6: -18.
Set 1 Network 7: 11. Set 2 Network 7: -18.
Set 1 Network 8: 11. Set 2 Network 8: -14.
Set 1 Network 9: 7. Set 2 Network 9: -15.
Set 1 Network Average: 12. Set 2 Network Average: -12.

Comparing set 1: size5set8 against set 2: size5set11
Final scores are as follows.
Set 1 Network 0: 0. Set 2 Network 0: -12.
Set 1 Network 1: -8. Set 2 Network 1: 2.
Set 1 Network 2: 2. Set 2 Network 2: 2.
Set 1 Network 3: 4. Set 2 Network 3: 4.
Set 1 Network 4: 6. Set 2 Network 4: 11.
Set 1 Network 5: 2. Set 2 Network 5: -6.
Set 1 Network 6: -2. Set 2 Network 6: 11.
Set 1 Network 7: 0. Set 2 Network 7: -12.
Set 1 Network 8: 5. Set 2 Network 8: -11.
Set 1 Network 9: 0. Set 2 Network 9: 2.
Set 1 Network Average: 0. Set 2 Network Average: 0.

Comparing set 1: size5set8 against set 2: size5set12
Final scores are as follows.
Set 1 Network 0: 6. Set 2 Network 0: -2.
Set 1 Network 1: 2. Set 2 Network 1: -16.
Set 1 Network 2: 14. Set 2 Network 2: -6.
Set 1 Network 3: 20. Set 2 Network 3: -4.
Set 1 Network 4: 19. Set 2 Network 4: -6.
Set 1 Network 5: 8. Set 2 Network 5: -9.
Set 1 Network 6: -6. Set 2 Network 6: -6.
Set 1 Network 7: -2. Set 2 Network 7: -6.
Set 1 Network 8: 19. Set 2 Network 8: -5.
Set 1 Network 9: -14. Set 2 Network 9: -6.
Set 1 Network Average: 6. Set 2 Network Average: -6.
```

B.3 Scaled Uniform vs Scaled Divergent

```
Comparing set 1: size5set5 against set 2: size5set1
Final scores are as follows.
Set 1 Network 0: 2. Set 2 Network 0: -2.
Set 1 Network 1: 5. Set 2 Network 1: -20.
Set 1 Network 2: 2. Set 2 Network 2: 0.
Set 1 Network 3: 10. Set 2 Network 3: 2.
Set 1 Network 4: 12. Set 2 Network 4: -4.
Set 1 Network 5: 4. Set 2 Network 5: -6.
Set 1 Network 6: 6. Set 2 Network 6: -20.
Set 1 Network 7: 8. Set 2 Network 7: -8.

Set 1 Network 8: 6. Set 2 Network 8: 4.

Set 1 Network 9: 12. Set 2 Network 9: -13.

Set 1 Network Average: 6. Set 2 Network Average: -6.


Comparing set 1: size5set5 against set 2: size5set2

Final scores are as follows.

Set 1 Network 0: 4. Set 2 Network 0: -14.

Set 1 Network 1: -4. Set 2 Network 1: -12.

Set 1 Network 2: 4. Set 2 Network 2: 0.

Set 1 Network 3: -2. Set 2 Network 3: 14.

Set 1 Network 4: -4. Set 2 Network 4: 14.

Set 1 Network 5: -4. Set 2 Network 5: -2.

Set 1 Network 6: -4. Set 2 Network 6: 10.

Set 1 Network 7: -6. Set 2 Network 7: 12.

Set 1 Network 8: 0. Set 2 Network 8: 14.

Set 1 Network 9: -4. Set 2 Network 9: -16.

Set 1 Network Average: -2. Set 2 Network Average: 2.


Comparing set 1: size5set5 against set 2: size5set3

Final scores are as follows.

Set 1 Network 0: 18. Set 2 Network 0: -18.

Set 1 Network 1: 16. Set 2 Network 1: -20.

Set 1 Network 2: 20. Set 2 Network 2: -20.

Set 1 Network 3: 20. Set 2 Network 3: -20.

Set 1 Network 4: 16. Set 2 Network 4: -20.

Set 1 Network 5: 20. Set 2 Network 5: -18.

Set 1 Network 6: 20. Set 2 Network 6: -20.

Set 1 Network 7: 20. Set 2 Network 7: -18.
Set 1 Network 8: 20. Set 2 Network 8: -20.
Set 1 Network 9: 20. Set 2 Network 9: -16.
Set 1 Network Average: 19. Set 2 Network Average: -19.


Comparing set 1: size5set5 against set 2: size5set4
Final scores are as follows.
Set 1 Network 0: 20. Set 2 Network 0: -2.
Set 1 Network 1: 20. Set 2 Network 1: -8.
Set 1 Network 2: 20. Set 2 Network 2: -10.
Set 1 Network 3: 6. Set 2 Network 3: -2.
Set 1 Network 4: -13. Set 2 Network 4: -10.
Set 1 Network 5: -8. Set 2 Network 5: -4.
Set 1 Network 6: 0. Set 2 Network 6: -6.
Set 1 Network 7: 8. Set 2 Network 7: 0.
Set 1 Network 8: 0. Set 2 Network 8: -8.
Set 1 Network 9: 0. Set 2 Network 9: -3.
Set 1 Network Average: 5. Set 2 Network Average: -5.


Comparing set 1: size5set6 against set 2: size5set1
Final scores are as follows.
Set 1 Network 0: 10. Set 2 Network 0: -16.
Set 1 Network 1: 10. Set 2 Network 1: -12.
Set 1 Network 2: 18. Set 2 Network 2: -12.
Set 1 Network 3: 18. Set 2 Network 3: -16.
Set 1 Network 4: 4. Set 2 Network 4: -10.
Set 1 Network 5: 4. Set 2 Network 5: -10.
Set 1 Network 6: 12. Set 2 Network 6: -8.
Set 1 Network 7: 20. Set 2 Network 7: -10.
Set 1 Network 8: 14. Set 2 Network 8: -14.
Set 1 Network 9: 14. Set 2 Network 9: -16.

Set 1 Network Average: 12. Set 2 Network Average: -12.


Comparing set 1: size5set6 against set 2: size5set2

Final scores are as follows.

Set 1 Network 0: 5. Set 2 Network 0: -12.

Set 1 Network 1: 2. Set 2 Network 1: 15.

Set 1 Network 2: 4. Set 2 Network 2: 2.

Set 1 Network 3: 6. Set 2 Network 3: -1.

Set 1 Network 4: -3. Set 2 Network 4: -12.

Set 1 Network 5: 10. Set 2 Network 5: -12.

Set 1 Network 6: -3. Set 2 Network 6: -1.

Set 1 Network 7: 4. Set 2 Network 7: -14.

Set 1 Network 8: 1. Set 2 Network 8: 1.

Set 1 Network 9: 1. Set 2 Network 9: 7.

Set 1 Network Average: 2. Set 2 Network Average: -2.


Comparing set 1: size5set6 against set 2: size5set3

Final scores are as follows.

Set 1 Network 0: 18. Set 2 Network 0: -18.

Set 1 Network 1: 19. Set 2 Network 1: -16.

Set 1 Network 2: 20. Set 2 Network 2: -20.

Set 1 Network 3: 18. Set 2 Network 3: -20.

Set 1 Network 4: 12. Set 2 Network 4: -16.

Set 1 Network 5: 18. Set 2 Network 5: -20.

Set 1 Network 6: 12. Set 2 Network 6: -18.

Set 1 Network 7: 20. Set 2 Network 7: -16.

Set 1 Network 8: 20. Set 2 Network 8: -17.

Set 1 Network 9: 20. Set 2 Network 9: -16.
Set 1 Network Average: 17. Set 2 Network Average: -17.


Comparing set 1: size5set6 against set 2: size5set4

Final scores are as follows.

Set 1 Network 0: 0. Set 2 Network 0: -2.

Set 1 Network 1: 0. Set 2 Network 1: -2.

Set 1 Network 2: 12. Set 2 Network 2: 4.

Set 1 Network 3: -4. Set 2 Network 3: 2.

Set 1 Network 4: 0. Set 2 Network 4: -2.

Set 1 Network 5: -4. Set 2 Network 5: 0.

Set 1 Network 6: 0. Set 2 Network 6: 2.

Set 1 Network 7: -4. Set 2 Network 7: -2.

Set 1 Network 8: 0. Set 2 Network 8: -2.

Set 1 Network 9: 0. Set 2 Network 9: 2.

Set 1 Network Average: 0. Set 2 Network Average: 0.


Comparing set 1: size5set7 against set 2: size5set1

Final scores are as follows.

Set 1 Network 0: -7. Set 2 Network 0: 2.

Set 1 Network 1: -6. Set 2 Network 1: 4.

Set 1 Network 2: -2. Set 2 Network 2: 10.

Set 1 Network 3: 8. Set 2 Network 3: 6.

Set 1 Network 4: -10. Set 2 Network 4: 4.

Set 1 Network 5: -10. Set 2 Network 5: 8.

Set 1 Network 6: 1. Set 2 Network 6: 13.

Set 1 Network 7: 2. Set 2 Network 7: 8.

Set 1 Network 8: -20. Set 2 Network 8: 2.

Set 1 Network 9: -7. Set 2 Network 9: -6.

Set 1 Network Average: -5. Set 2 Network Average: 5.
```

```
Comparing set 1: size5set7 against set 2: size5set2

Final scores are as follows.

Set 1 Network 0: -17. Set 2 Network 0: 2.

Set 1 Network 1: -1. Set 2 Network 1: 12.

Set 1 Network 2: -20. Set 2 Network 2: 5.

Set 1 Network 3: -19. Set 2 Network 3: 15.

Set 1 Network 4: -15. Set 2 Network 4: 19.

Set 1 Network 5: -13. Set 2 Network 5: 15.

Set 1 Network 6: -18. Set 2 Network 6: 11.

Set 1 Network 7: -19. Set 2 Network 7: 14.

Set 1 Network 8: -5. Set 2 Network 8: 19.

Set 1 Network 9: 4. Set 2 Network 9: 11.

Set 1 Network Average: -12. Set 2 Network Average: 12.


Comparing set 1: size5set7 against set 2: size5set3

Final scores are as follows.

Set 1 Network 0: 12. Set 2 Network 0: -4.

Set 1 Network 1: 15. Set 2 Network 1: -14.

Set 1 Network 2: 2. Set 2 Network 2: 4.

Set 1 Network 3: 2. Set 2 Network 3: -10.

Set 1 Network 4: 1. Set 2 Network 4: -11.

Set 1 Network 5: -4. Set 2 Network 5: -8.

Set 1 Network 6: 10. Set 2 Network 6: -12.

Set 1 Network 7: 8. Set 2 Network 7: -9.

Set 1 Network 8: 14. Set 2 Network 8: 6.

Set 1 Network 9: 12. Set 2 Network 9: -14.

Set 1 Network Average: 7. Set 2 Network Average: -7.
```

```
Comparing set 1: size5set7 against set 2: size5set4
Final scores are as follows.
Set 1 Network 0: -8. Set 2 Network 0: 10.
Set 1 Network 1: -8. Set 2 Network 1: 6.
Set 1 Network 2: -18. Set 2 Network 2: 10.
Set 1 Network 3: -18. Set 2 Network 3: 14.
Set 1 Network 4: -18. Set 2 Network 4: 10.
Set 1 Network 5: -20. Set 2 Network 5: 6.
Set 1 Network 6: 6. Set 2 Network 6: 12.
Set 1 Network 7: -2. Set 2 Network 7: 12.
Set 1 Network 8: -8. Set 2 Network 8: 10.
Set 1 Network 9: -2. Set 2 Network 9: 6.
Set 1 Network Average: -9. Set 2 Network Average: 9.

Comparing set 1: size5set8 against set 2: size5set1
Final scores are as follows.
Set 1 Network 0: 2. Set 2 Network 0: 5.
Set 1 Network 1: -6. Set 2 Network 1: -8.
Set 1 Network 2: 5. Set 2 Network 2: 19.
Set 1 Network 3: -3. Set 2 Network 3: 8.
Set 1 Network 4: 0. Set 2 Network 4: 2.
Set 1 Network 5: 5. Set 2 Network 5: 16.
Set 1 Network 6: -18. Set 2 Network 6: -2.
Set 1 Network 7: -16. Set 2 Network 7: 2.
Set 1 Network 8: -12. Set 2 Network 8: 7.
Set 1 Network 9: -4. Set 2 Network 9: -2.
Set 1 Network Average: -4. Set 2 Network Average: 4.

Comparing set 1: size5set8 against set 2: size5set2
```

```
Final scores are as follows.
Set 1 Network 0: 0. Set 2 Network 0: -1.
Set 1 Network 1: -4. Set 2 Network 1: 2.
Set 1 Network 2: -2. Set 2 Network 2: 8.
Set 1 Network 3: 2. Set 2 Network 3: 0.
Set 1 Network 4: -2. Set 2 Network 4: 8.
Set 1 Network 5: -4. Set 2 Network 5: 2.
Set 1 Network 6: -13. Set 2 Network 6: 0.
Set 1 Network 7: -4. Set 2 Network 7: 12.
Set 1 Network 8: -2. Set 2 Network 8: 0.
Set 1 Network 9: -4. Set 2 Network 9: 2.
Set 1 Network Average: -3. Set 2 Network Average: 3.

Comparing set 1: size5set8 against set 2: size5set3
Final scores are as follows.
Set 1 Network 0: 20. Set 2 Network 0: -10.
Set 1 Network 1: 4. Set 2 Network 1: -13.
Set 1 Network 2: 20. Set 2 Network 2: -8.
Set 1 Network 3: 19. Set 2 Network 3: -7.
Set 1 Network 4: 20. Set 2 Network 4: -16.
Set 1 Network 5: 14. Set 2 Network 5: -12.
Set 1 Network 6: -2. Set 2 Network 6: -12.
Set 1 Network 7: -5. Set 2 Network 7: -16.
Set 1 Network 8: 20. Set 2 Network 8: -7.
Set 1 Network 9: 6. Set 2 Network 9: -15.
Set 1 Network Average: 11. Set 2 Network Average: -11.

Comparing set 1: size5set8 against set 2: size5set4
Final scores are as follows.
```

```
Set 1 Network 0: -19. Set 2 Network 0: 14.

Set 1 Network 1: -12. Set 2 Network 1: 12.

Set 1 Network 2: -20. Set 2 Network 2: 12.

Set 1 Network 3: 0. Set 2 Network 3: 12.

Set 1 Network 4: -20. Set 2 Network 4: 10.

Set 1 Network 5: -20. Set 2 Network 5: 10.

Set 1 Network 6: -4. Set 2 Network 6: 13.

Set 1 Network 7: -12. Set 2 Network 7: 10.

Set 1 Network 8: -12. Set 2 Network 8: 12.

Set 1 Network 9: 2. Set 2 Network 9: 12.

Set 1 Network Average: -11. Set 2 Network Average: 11.
```

Appendix C – Code Repository

All code for this project at the time of completion can be found at the following permalink: https://github.com/wduncanfraser/scalable_go/tree/MP. While this codebase will continue to evolve as I continue to develop Go AI, this tag will remain as a static record for this project documentation.