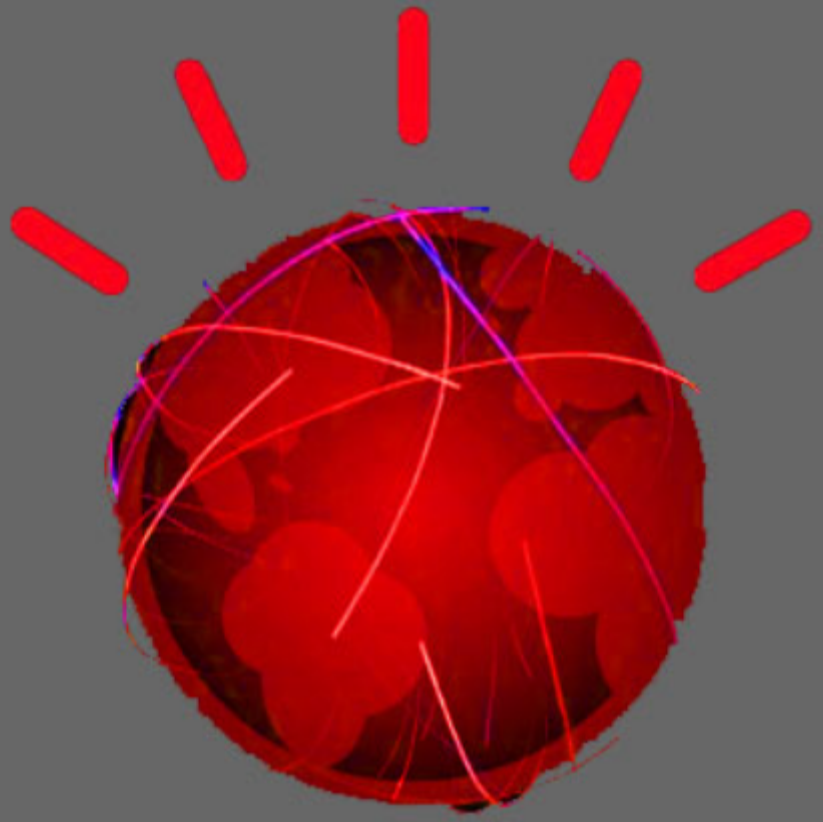


Lisa Fawcett, CS, Sophomore
Jonathan Donovan, ChemE, Junior
Victor Haocheng Shen, CS, Junior

DEEP RED

An Intelligent Approach to Chinese Checkers

Abstract

- Our ability to successfully train Neural Networks was limited by errors in the more fundamental classes, such as board and move generation

- Ultimately, once these errors were repaired, our alpha beta static evaluator performed quite well, and backpropogation began on it's static evaluation function

- We believe that backpropogation was an unrealistically lengthy process for the short time we had after finalizing the alpha beta search

- Shows promise

Introduction

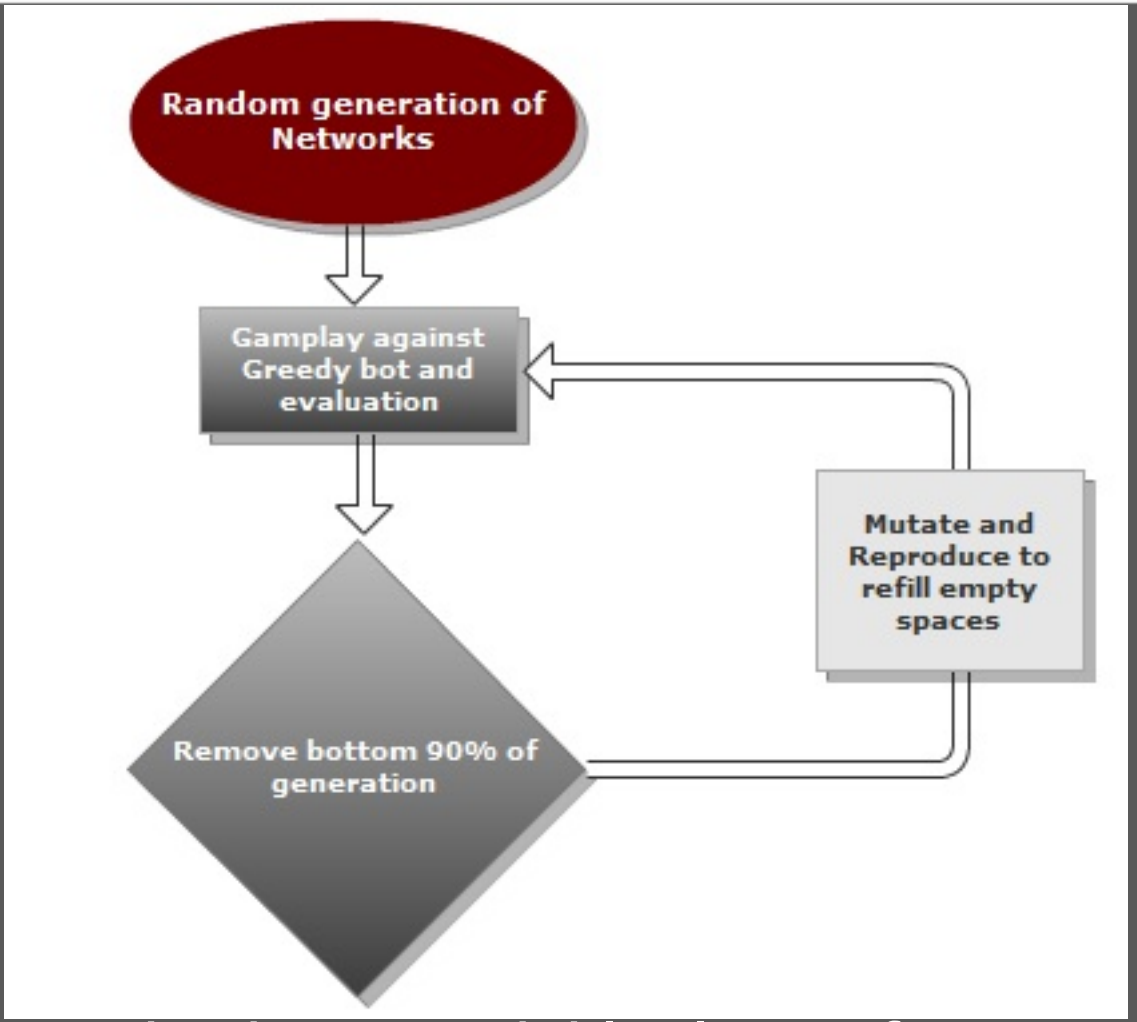
- Our initial effort, using an evolutionary algorithm to develop Neural Networks, did not succeed.
- Efforts to use backpropogation were promising while using a Neural Net that took all individual locations as inputs.
- Unfortunately, errors in board implementation and move generation forced us to begin training anew
- Additionally, input style was changed in order to speed training
- At this point we began using an implementation of our Alpha Beta searcher to train our boards

Introduction

- Generating random boards close to the end state, we hoped to overcome one of the steepest challenges to the success of our players.
- Random board generation had not trained them well enough to play to endgame, and we hoped to change this
- During this time we also improved our static evaluation function, and expanded the updated move generation to include limited marble usage
- We also ultimately added a decay factor to remove problems discriminating boards very near the end

Method

- Stage 1



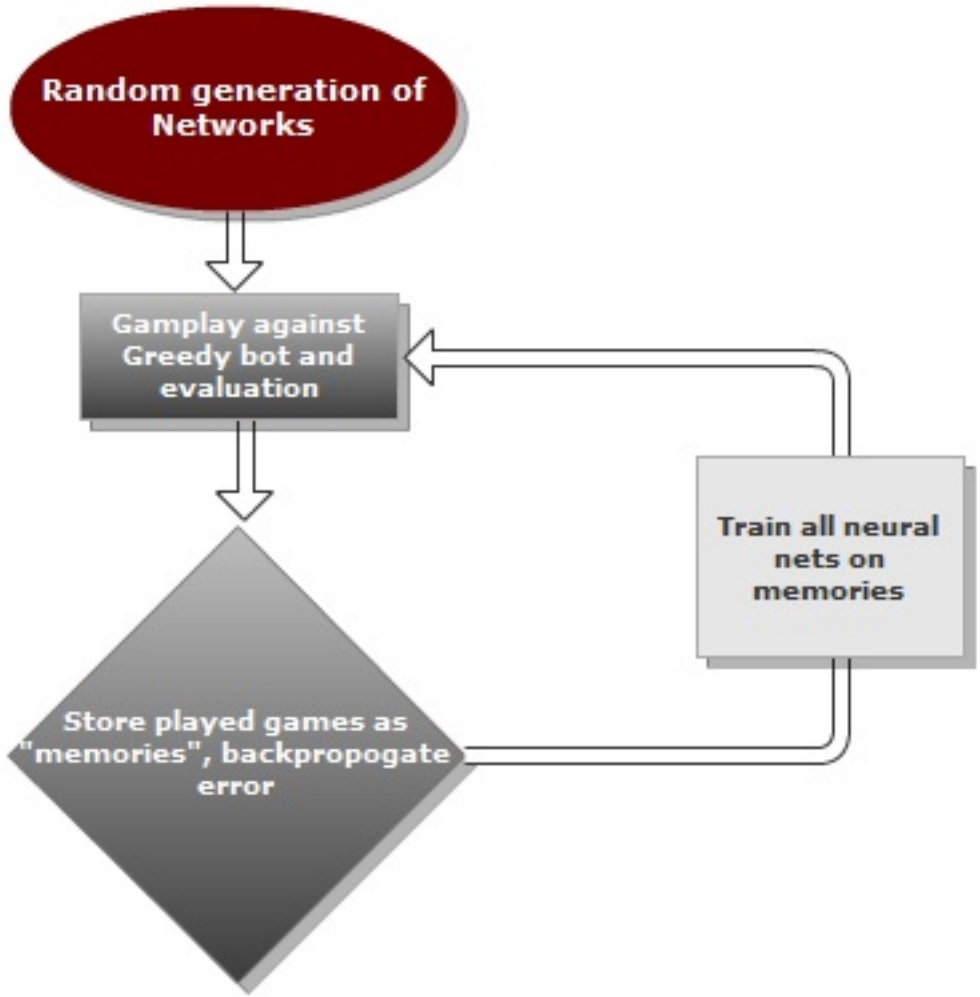
Benefits:
Large population
Quick runtime

Issues: Local minima reached very quickly, loss of generalizability, little value in reproduction

Method

- Stage 2

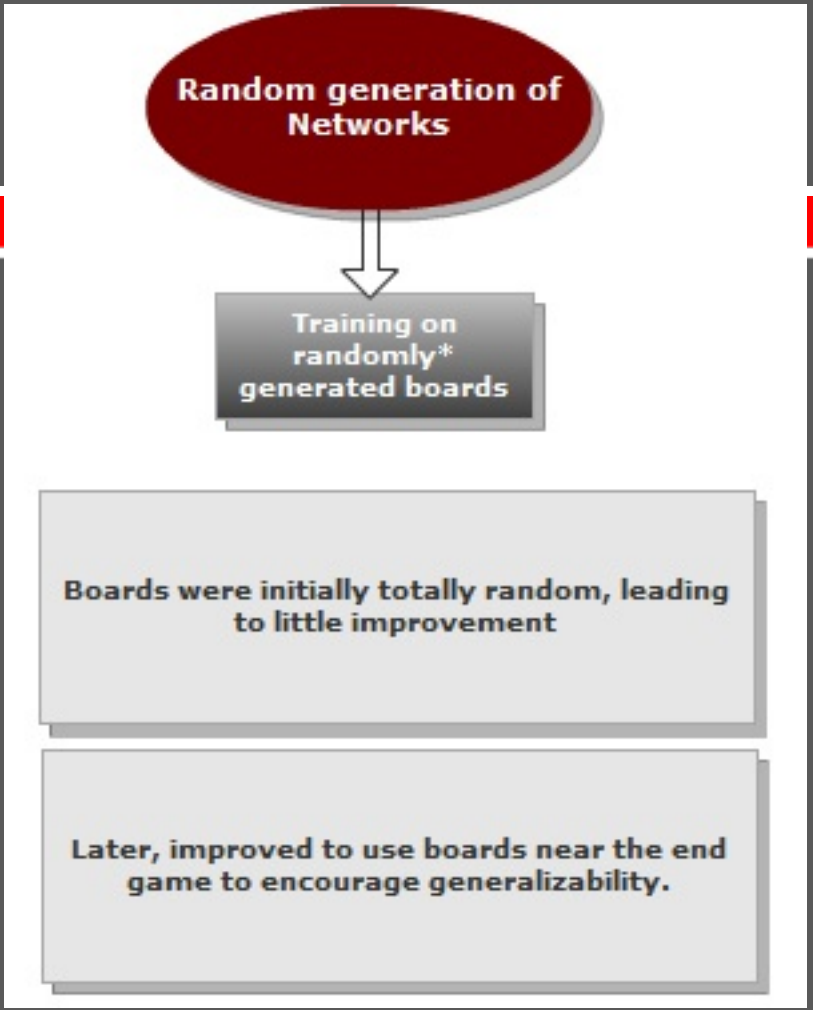
Benefits: few



Issues: Only negative examples led to abysmal performance, no improvement in scores, and quick convergence of populations.

Method

- Stage 3

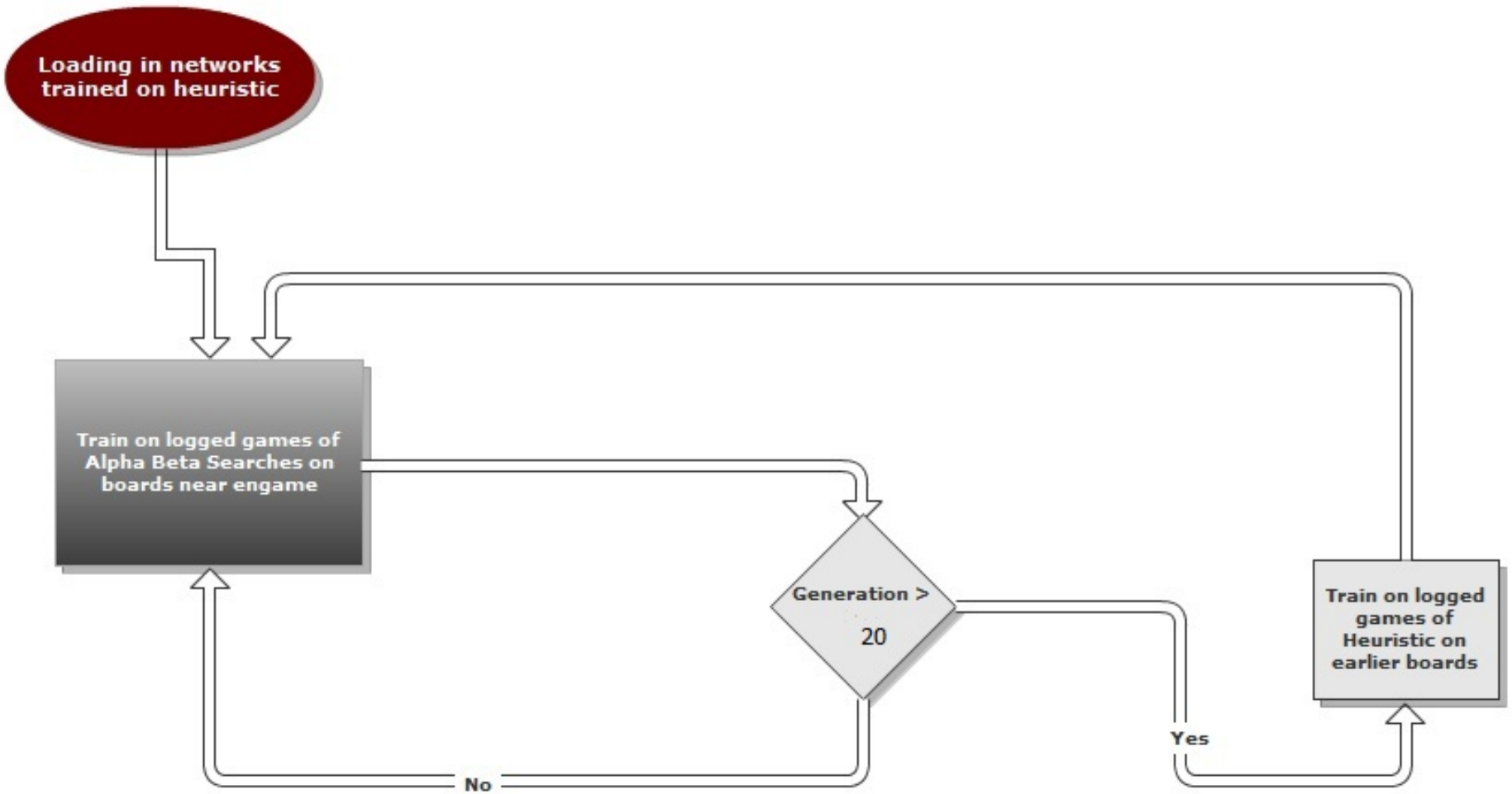


Benefits:
Most promising
method after modifications

Issues: Used during Game/Server switch, so some good data became unusable

Method

- Stage 4



Method

- Stage 4

Benefits:
Ideally, this would improve beyond abilities of heuristic searcher

Issues: Takes a long time to train, as only two values are used, 1/-1, and the function to calculate them is very complex.
Issue of potential intractability in input space

Related Work

Our initial effort was inspired by the following two papers on creating artificial neural network players

A Comparison of Neural Network Architectures in Reinforcement Learning in the Game of Othello
-by Dmitry Kamenetsky, Bsc.

Evolving an Expert Checkers Playing Program without Using Human Expertise
by Kumar Chellapilla, et.al

and then we found that unsupervised learning didn't work well...

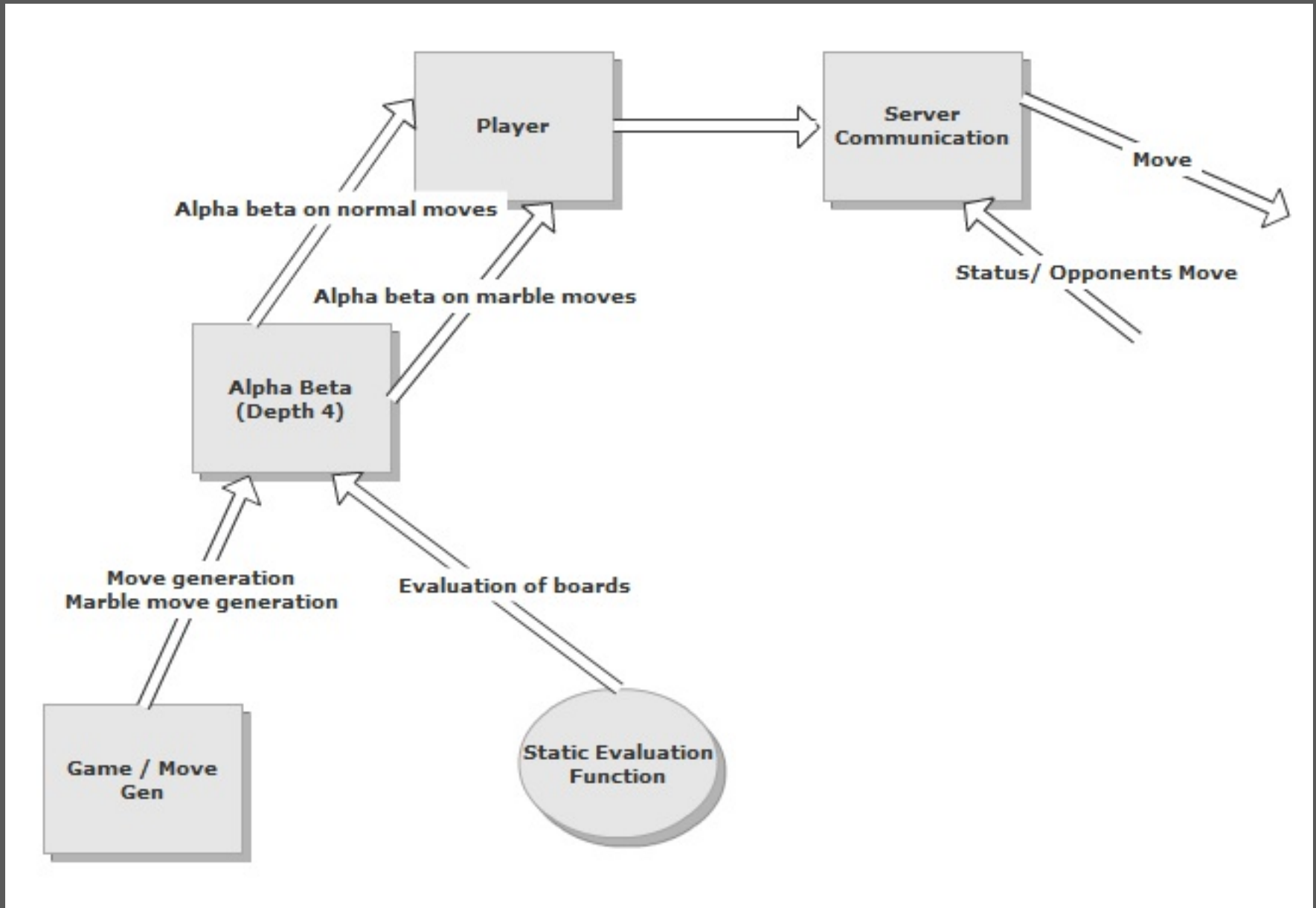
Related Work

...so then the Reinforcement Learning method is hinted by
Temporal Difference Learning and TD-Gammon
by Gerald Tesauro

Our working version is inspired by the ancient work done by Arthur Samuel's English checkers program. Basically, human expertise and analytics involved in making evaluation function.

The evolutionary algorithmic approach was also found in our development of a static evaluation function, as multiple functions were competed and the best selected through trial and error

System Architecture and Implementation



System Architecture and Implementation

- We were plagued by problems with bug testing until very shortly before break
- Most unfortunately, issues in input generation for Neural Networks, and move generation on our custom board required a rapid overhaul of the system
- This reduced the amount of time we were able to use our functioning backpropogation algorithm on the new board/function
- Our switch in focus to the basics paid off when our evaluation function alpha beta searched improved perfomance significantly when bug-free

System Architecture

- After working, we have improved our static evaluation function further using Evolutionary Algorithm techniques of survival of the fittest
- This allowed us to achieve a level of success on the server up to rank 3 (excluding Eigenbot)
- We have also added in a Marble using heuristic, where a marble is used if the alpha beta value of that move is > 0.1 of the other. This has allowed us to beat Eigenbot in our own simulations, and we have hope it will raise our ranking

Evaluation Methodology

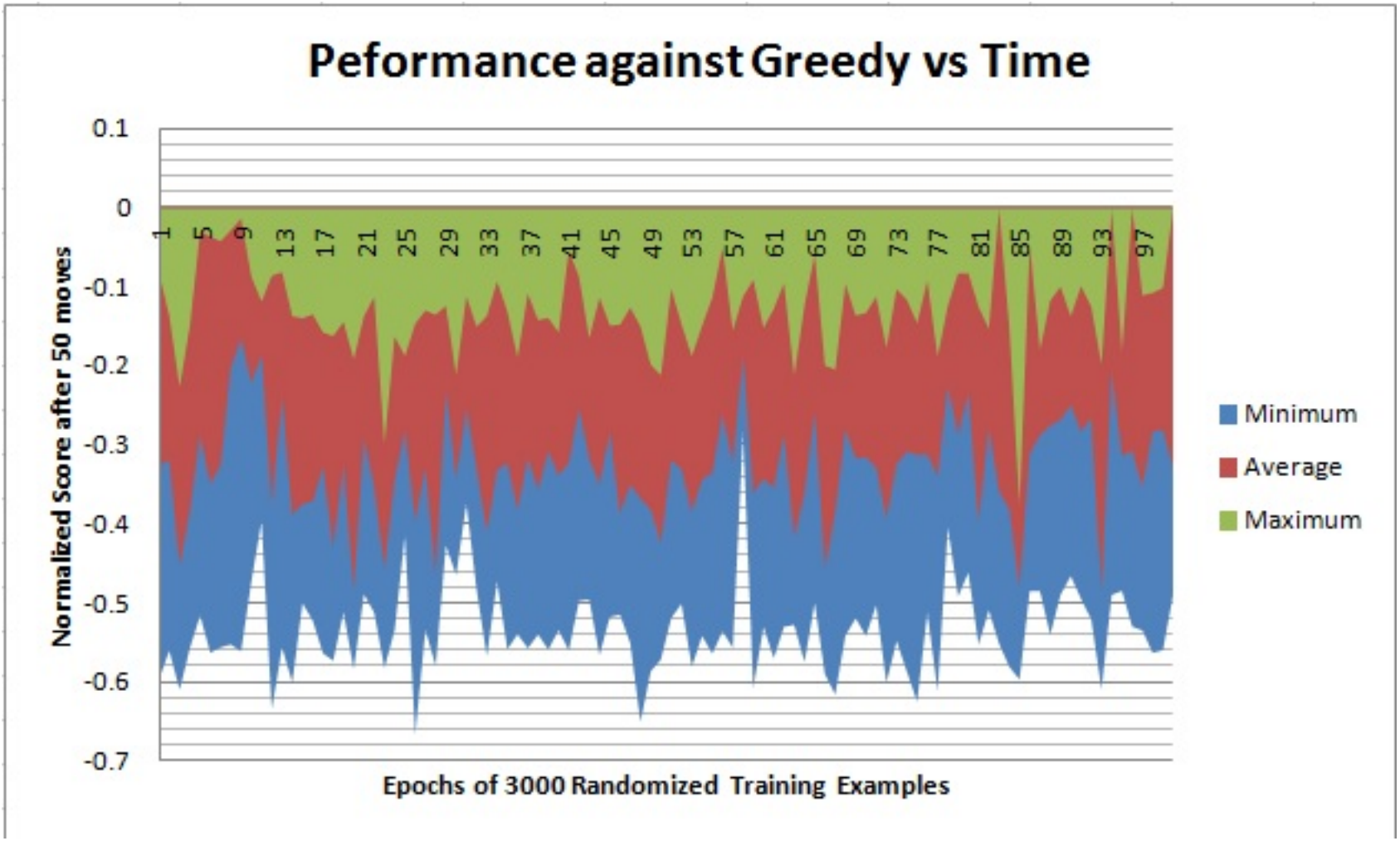
First method: Backpropogating randomly generated boards on static evaluation function

Second method: Train previously trained neural networks on the endvalue of games played by our alpha beta searcher

Unforunately, errors in other code allowed us to train Neural Nets for only a short time with functioning inputs / move generation for a short time

val(player) = ∑(player marble dist. from finish)^2
board_eval = (val(opponent) - val(us)) / (val(opponent) + val(us))
The board_eval will be -1 if our opponent has won and +1 if we have won.

Experimental Results



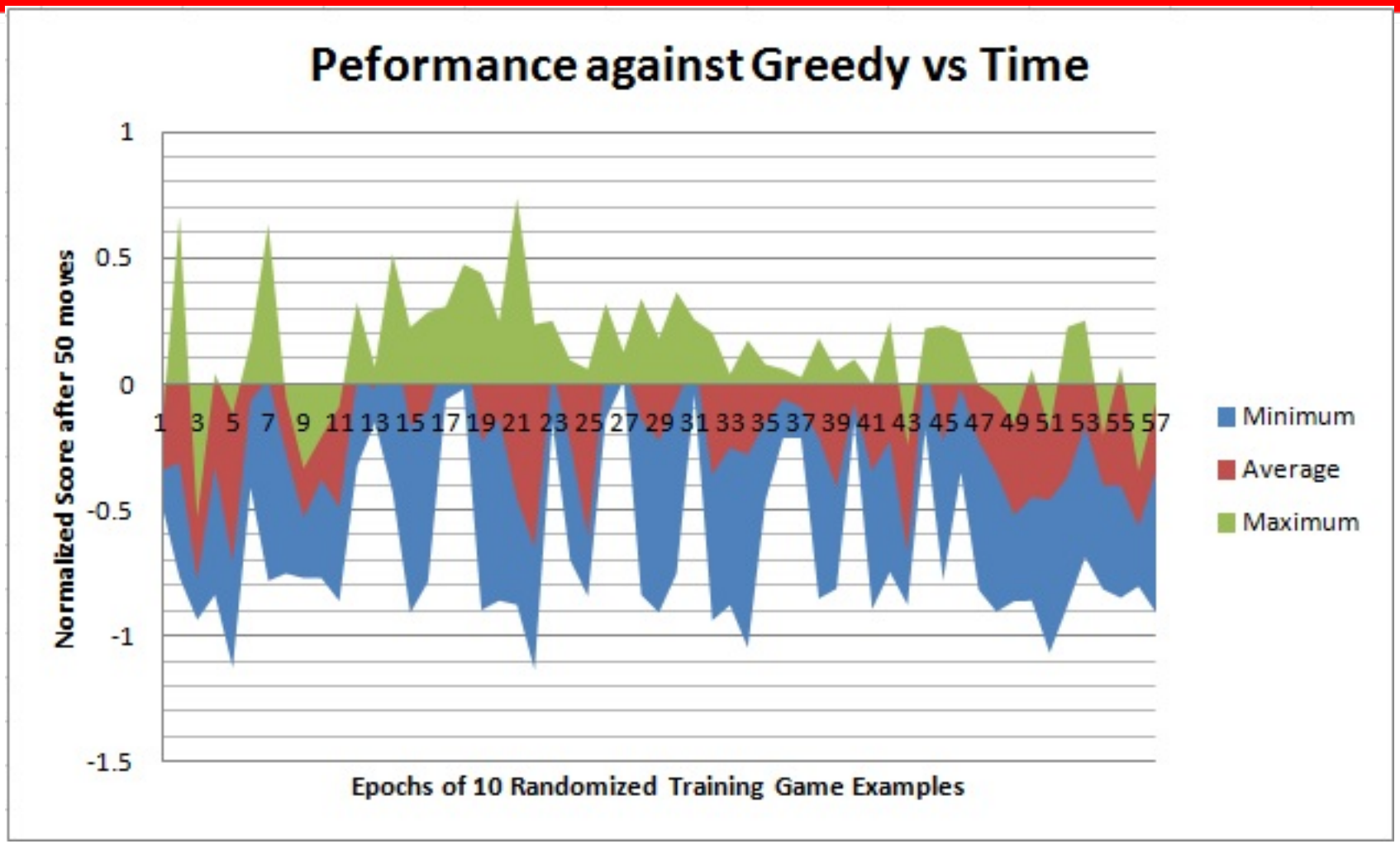
Poor performance of random sampling (Population 9)

Experimental Results

Method 4: Training on game results.

T > 20 the sharp decline is associated with increase in sample space as early boards are also used.

Experimental Results



Performance of game playout training with limited realistic training data (population 4)

Experimental Results

Method 4: Training on game results.

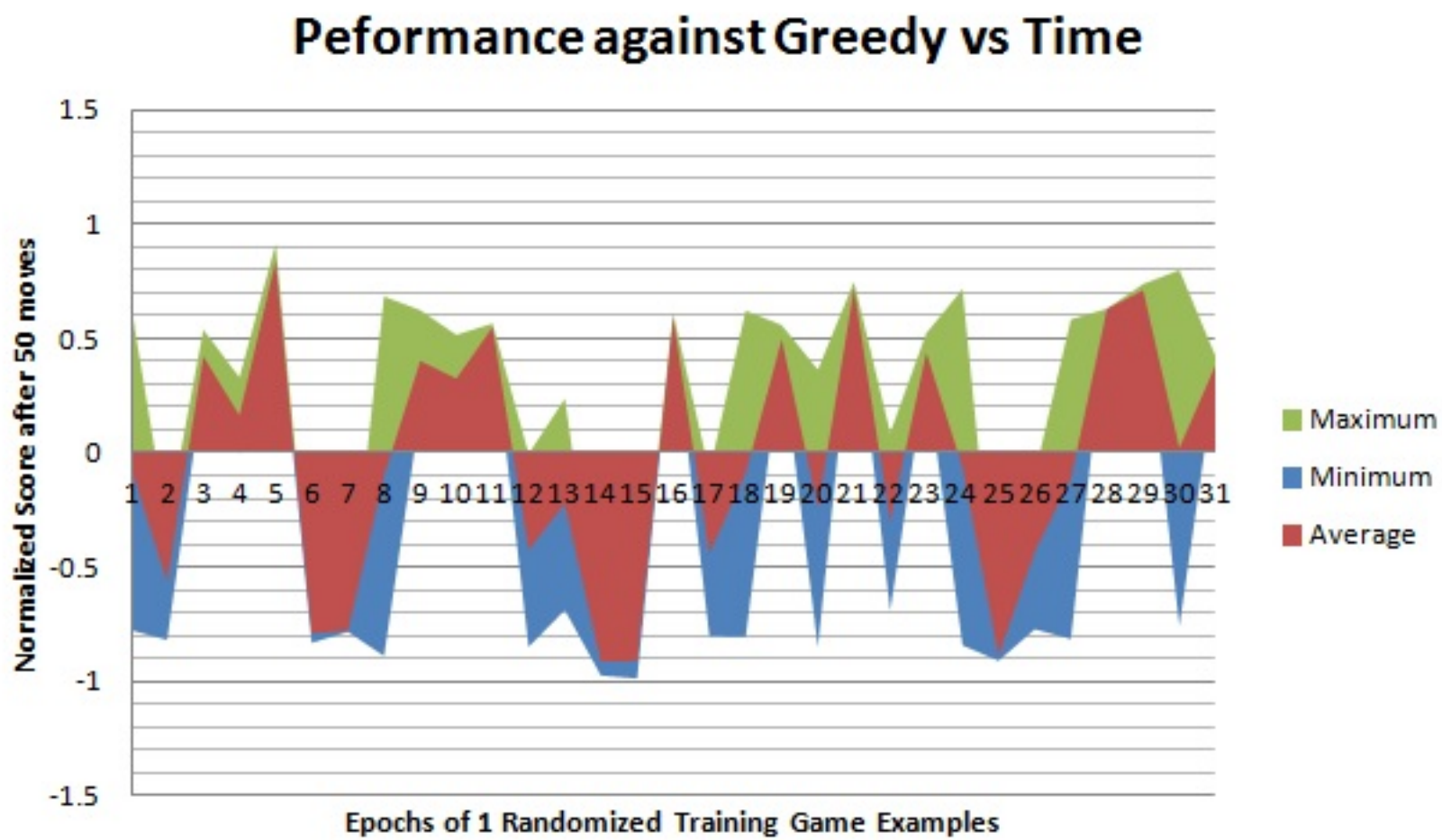
The results from before were continued by training only on near endgame states.

Unfortunately data collection overwrote older data, and we only have the data for the final point:

Min: 0.3699775310246579
Max: 0.5386018846062455
Mean: 0.4143652124407262 7

Boards play well till near endgame, then get stuck
Decay factor would likely increase training performance

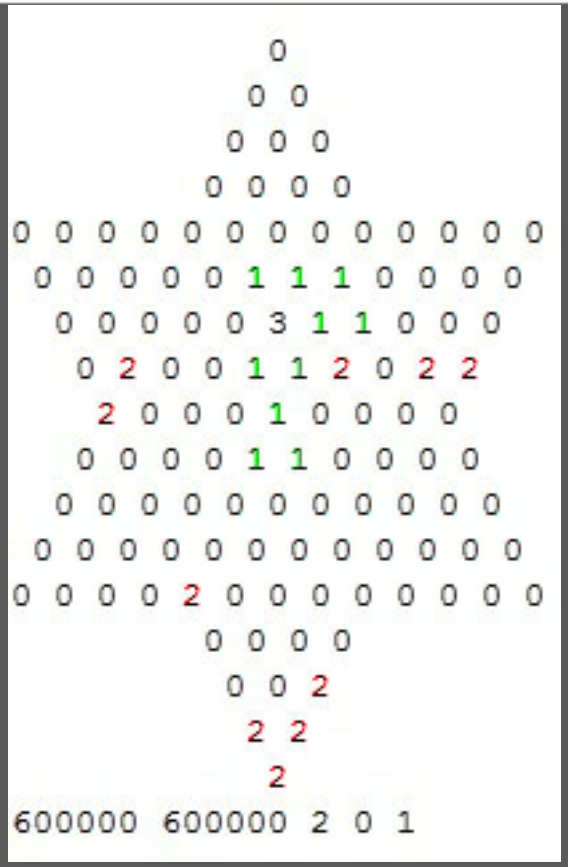
Experimental Results



Performance of game playout training with limited realistic training data (population 3)

Experimental Results

Method 4: Sample of trained gameplay



This board is the current peak of playing ability of our Neural Nets. Player 2 is a Greedy AI, and you can see how the net up until this point is competitive with the AI.

After this point, the NN makes illogical sideways moves, which we believe is caused by the uniform increase in values of the training, unless it can get the fine gradient of states near the goal state, it cannot play well at this point. A much more difficult problem

Experimental Discussion

- Ultimately we believe our NN training was unsuccessful not because of intractability of the problem, but due to issues with our own methodology of testing key components of the system
- Some experimental data, such as training on played games, showed massive improvments in NN performance over only hundreds of training examples on semi-random gameplay
- Data for endgame play was also promising
- However, the amount of time training took led us to be unable to finish after our code was verified

Future Work

Sub functions that can be modified to improve static eval:
- The negativity function
- Modifications to distance from goal loc function
- Modifications of threshold improvement to use a marble

Future modifications

Experiment with finding the optimal combination of the above functions

Continuation of training Neural Network, if only to demonstrate proof of concept

References

Evolving an Expert Checkers Playing Program without Using Human Expertise by Kumar Chellapilla et.al

A Comparison of Neural Network Architectures in Reinforcement Learning in the Game of Othello
by Dmitry Kamenetsky, Bsc.

Temporal Difference and TD-Gammon
by Gerald Tesauro

and.... the most important one...
Wikipedia

Acknowledgement

Special Thanks to:

Nikos!

Conclusion

- Bug testing of pivot components of code should be conducted ***thoroughly*** and ***often***, although JUnit tests were suggested and used for Neural Networks, they were not used at all in pivotal components that remained unworkable until the end of the project, very deleterious to our performance.
- Neural Networks serve as powerful representational tools, but require large amounts of time to train on complicated problems. Unsupervised search through the space of them is nearly futile.
- Learning factor and training data selection, as well as input representation, are all highly important, and performance hinges on the ability to successfully test many different methods.