Neural Net Reports
CS 477/577

Willard L. Miranker
Yale/DCS/TR1227
April 2002

Table of Contents

Neural Network Investigation of Effects of Neurogenesis on Memory

Alison Austin and Lisa Malnick
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract

We found that **neural networks** vary in the number of iterations necessary for convergence when learning new data based on the construction of the network. Networks that are asked to **learn** new data perform identically to networks that have new neurons added, while networks that have lost neurons perform worse in a manner directly proportional to the number of neurons lost. We also demonstrated that recall improved with the addition of neurons, and worsened with their removal.

1. INTRODUCTION

Until recently, it was thought that humans were born with all of the neurons that they would ever have. However, it has recently been found that new neurons are born and differentiated in the hippocampus. Because of the involvement of the hippocampus in short term memory, retrieval of long term memory, emotion, stress and many mental illnesses, this finding may have a profound impact on the treatment of many forms of mental illness {Chambers, 2001}. The effects of neurogenesis and apoptosis are largely unknown. It is possible that the brain uses neurogenesis to speed up the learning process because it may be that it takes less energy to differentiate new neurons than to perform the restructuring which may be necessary when learning to behave under new circumstances.

In our project, we investigated the effects of neurogenesis and apoptosis on memory and learning. We first constructed a base neural network that used back-propagation to learn a series of images. The precision of the network was used as the base for all subsequent experiments. This base network was then tested with one image and a noisy version of that image, and the outputs of those simulations were noted. We also attempted to teach the base network a new set of images and determined the number of iterations necessary to learn those new images. Finally, we tested the difference between learning and recall in our base network and several networks that alter neuronal

content in various ways. We first added various numbers of new neurons to our base network and examined the effects of the additions. We then removed various numbers of neurons from the network and examined the learning and recall as compared to the original network. Finally, we removed some neurons and added new ones to determine the difference between differentiation of new neurons and restructuring old ones.

2. METHODS

2.1 Network Design

We created a feed-forward neural network that used gradient descent training with back-propagation, a constant learning rate and no momentum. There are 3 layers; the input layer has 225 neurons, the hidden layer has 50 neurons, and the output layer has 10 neurons. We used the hyperbolic tangent sigmoid transfer function as the gain function.

2.2 Training Data

For training data, we used a series of ten different letters, pixilated into a 15 x 15 grid. Pictures of this data can be found in Section 6.

2.3 New Data

To test the ability of the network to learn new data, we added another series of 5 letters, also pixilated into a 15 x 15 grid. This data can be found in Section 7.

3. EXPERIMENTS

3.1 Base Network

In this experiment, we constructed our base network as described in Section 2.1. We trained the network with the training set described in Section 2.2 and noted the performance of the network over 10,000 epochs. This network will be referred to as the singly trained base network. We then further trained the network with the complete training set of ten original and five new patterns described in Sections 2.2 and 2.3 and noted the number of iterations necessary for the network to learn the new data set. Finally, we tested the recall of both a clean and a noisy version of one of the initial inputs.

3.2 Base Network Plus Five Neurons

Next, we added five neurons to the hidden layer of our singly trained base network. The network is fully connected, and weights were randomly assigned. We then trained the network with all fifteen patterns from Sections 2.2 and 2.3 and noted the number of iterations necessary for this network to learn the new patterns. We also tested the recall of both one original pattern and a noisy version of that pattern. This is a simulation of neurogenesis with no accompanying apoptosis (cell death).

3.3 Base Network Minus Five Neurons

We then subtracted five neurons from the hidden layer of our singly trained base network, retrained the network with the complete set of fifteen patterns from Sections 2.2 and 2.3 and noted the number of iterations necessary for this network to learn the new patterns. We also tested the recall of one of the original patterns as well as a noisy version of that pattern. This experiment was designed to simulate apoptosis without any accompanying neurogenesis.

3.4 Reset Base Network

In addition, we reset the weights of our singly trained base network to random values and taught the network the entire set of training data plus the five new patterns and noted the performance of the network over 10,000 epochs. We also tested the recall of one pattern from the initial training data set and a noisy version of that pattern. This experiment simulated simultaneous neurogenesis and apoptosis to create a new network.

3.5 Base Network Plus Fifteen Neurons

We then added fifteen neurons to the hidden layer of our singly trained base network with random weights, showed the network the complete set of training data plus the five new patterns from Sections 2.2 and 2.3 and noted the number of iterations necessary for this network to learn the new patterns. We also tested the recall of one pattern from the initial training data set and a noisy version of that pattern. We used this experiment to determine if the number of neurons added to the hidden layer would affect the performance of the network.

3.6 Base Network Minus Fifteen Neurons

Finally, we subtracted fifteen neurons from the hidden layer of our singly trained base network, showed the network the complete set of training data plus the five new patterns from Sections 2.2 and 2.3 and noted the number of iterations necessary for this network

to learn the new patterns. Again, we tested the recall of one pattern from the initial training data set and a noisy version of that pattern. We used this experiment to determine if the number of neurons removed from the hidden layer would affect the performance of the network.

3.7 Hebbian Learning

We created a neural network with the same dimensions as our base network, but with Hebbian learning and a forgetting factor. Unfortunately, we were unable to get this network to converge over 10,000 epochs, so we were unable to include any of these results.

4. RESULTS

Table 1. Convergence Data

Network	Activity	Number of Iterations
Base Network	Learn Training Set	10000
Trained Base Network	Learn Entire Data Set	2220
Trained Base Network + 5 Neurons	Learn Entire Data Set	2220
Trained Base Network - 5 Neurons	Learn Entire Data Set	2588
Reset Base Network	Learn Entire Data Set	2220
Trained Base Network + 15 Neurons	Learn Entire Data Set	2220
Trained Base Network – 15 Neurons	Learn Entire Data Set	4138

Table 2. Recall Output

Base Network Trained With 10 Images (Trained Network)		Trained Network Re-Trained With All 15 Images		Base Network Trained With All 15 Images	
A	Noisy A	A	Noisy A	A	Noisy A
-0.0058	0.1626	0.1306	0.1616	0.0059	0.2025
-0.0163	0.4305	0.0063	0.5591	-0.0056	0.5347
-0.0067	0.1902	0.1933	0.4607	-0.0071	0.1976
-0.0028	0.2278	-0.0883	0.1513	-0.0050	0.2252
0.0095	0.4988	0.2842	0.6332	0.0210	0.4191
0.0063	0.6205	-0.3668	0.1529	0.0188	0.4456
0.0106	-0.1010	0.7528	0.6846	0.0157	0.0740
0.0042	-0.2447	0.5337	0.1438	0.0239	-0.3441
0.0036	0.1616	-0.4505	-0.2876	0.0036	0.1399
0.9353	0.9415	0.8285	0.7956	0.9219	0.9140

Trained Network + 5 Neurons		Trained Network + 15 Neurons		Trained Network – 5 Neurons		Trained Networks – 15 Neurons	
A	Noisy A	A	Noisy A	A	Noisy A	A	Noisy A
0.0059	0.2026	0.0059	0.2026	0.0198	0.3057	0.0018	0.2897
-0.0056	0.5347	-0.0056	0.5347	-0.0243	0.2320	0.0096	0.4826
-0.0071	0.1976	-0.0071	0.1976	0.0136	0.3645	-0.0220	0.2557
-0.0050	0.2252	-0.0050	0.2252	-0.0107	0.3101	-0.0367	0.1306
0.0210	0.4191	0.0210	0.4191	0.0097	0.1525	-0.0097	0.4036
0.0188	0.4456	0.0188	0.4456	0.0184	0.4770	0.0368	0.3259
0.0157	0.0740	0.0157	0.0740	0.0174	0.0387	0.0370	0.3353
0.0239	-0.3441	0.0239	-0.3441	0.0398	0.0604	0.0280	0.1171
0.0036	0.1399	0.0036	0.1399	-0.0024	0.1816	0.0264	0.1453
0.9219	0.9140	0.9219	0.9140	0.9111	0.8726	0.8790	0.8442

5. CONCLUSIONS

We found that neurogenesis does not have any affect on the learning rate of a neural network. Conversely, apoptosis significantly slows the learning rate proportional to the number of neurons lost. The affects of apoptosis and neurogenesis on recall of learned images were much more interesting. Our data indicated that a network that was trained on an initial set of data recalls that data to a higher degree of precision when new neurons are added before retraining the network then when the same network is presented with the new data. We also demonstrated that recall gets proportionally less precise as more neurons are removed from the network.

Through this project, we hoped to find interesting relationships between the learning and recall in a rebuilt network versus a newly constructed network. We learned that there is surprisingly little difference in learning rate between teaching an existing network new data and rebuilding the network with all new neurons. We also learned that recall of data is better when retraining the existing network then when rebuilding. We feel that this information, combined with further study, may be helpful in determining the clinical relevance of neurogenesis in treating mental illness as well as the biological reasons for neurogenesis, which are as of yet unknown.

6. TRAINING DATA

Figure 1. A

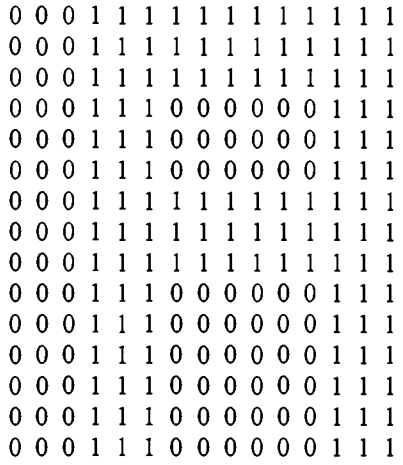


Figure 2. C

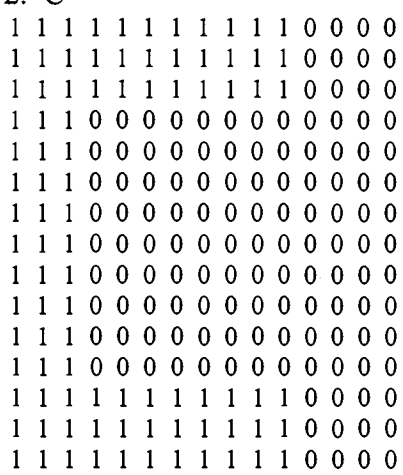


Figure 3. E

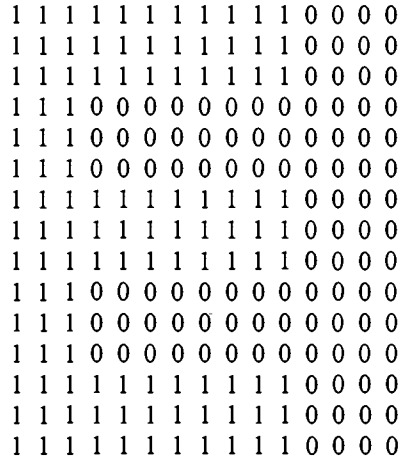


Figure 4. F

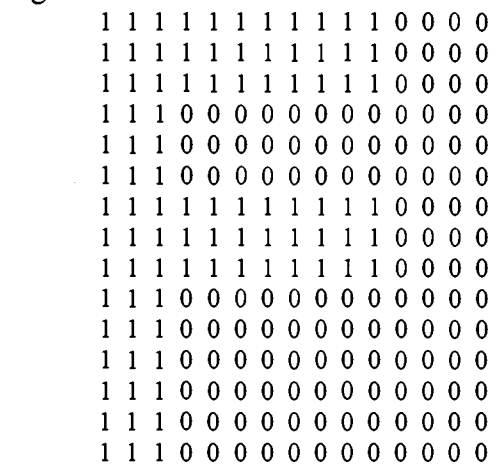


Figure 5. H

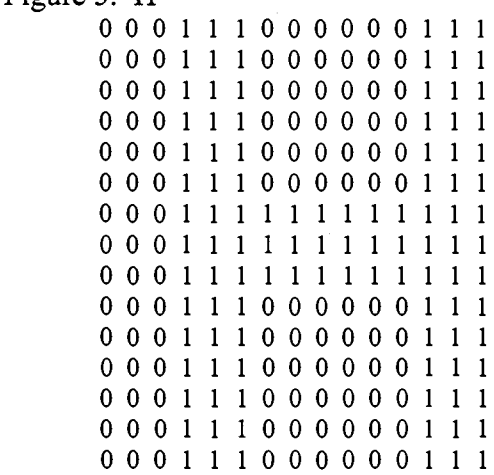


Figure 6. I

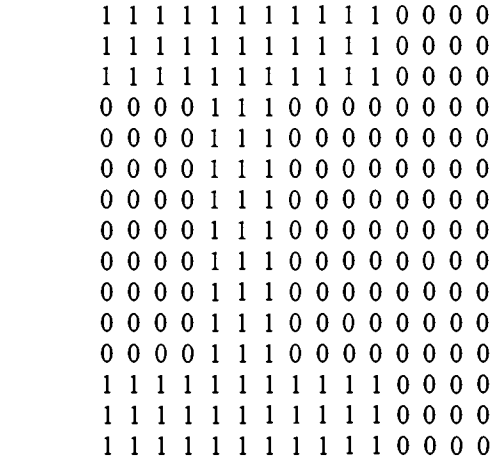


Figure 7. J

```
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
```

Figure 8. K

```
0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
0 1 1 0 0 0 0 1 1 0 0 0 0 0 0
0 1 1 0 0 0 1 1 1 0 0 0 0 0 0
0 1 1 0 0 1 1 1 0 0 0 0 0 0 0
0 1 1 0 1 1 1 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 1 0 0 0 0 0 0 0 0
0 1 1 0 0 1 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 1 1 1 0 0 0 0 0 0
0 1 1 0 0 0 0 1 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
```

Figure 9. L

```
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
```

Figure 10. M

```
1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 0 0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1 1 0 1 1 1
1 1 1 0 0 1 1 1 1 1 0 0 1 1 1
1 1 1 0 0 0 1 1 1 0 0 0 1 1 1
1 1 1 0 0 0 0 1 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
```

7. NEW IMAGES

Figure 11. O

```
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 0 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
```

Figure 12. S

```
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
```

Figure 13. U

```
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
```

Figure 14. X

```
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
0 1 1 1 1 1 0 0 0 1 1 1 1 1 0
0 0 1 1 1 1 1 0 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 1 1 1 1 1 0 0
0 1 1 1 1 1 0 0 0 1 1 1 1 1 0
1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
```

Figure 15. Y

0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 0 0

8. NOISY IMAGE

Figure 16. Noisy A

0 0 0 1 1 1 0 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 0 1 0 1 1 1 1
0 0 0 1 0 1 1 1 1 1 1 0 1 1 1
0 0 0 1 1 0 0 1 0 0 0 0 1 1 1
0 1 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0 0 0 0 0 1 1 1

REFERENCES

1. Chambers, Robert (2001). Neural Net Simulation of Hippocampal Neurogenesis.
2. Haykin, S. (1999). *Neural Networks a Comprehensive Foundation.* Prentice Hall.

Predicting and Analyzing Voting Patterns In The US Senate

John Eure[1] and Jonathan Winer[2]
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract

Politics makes strange, but perhaps not unpredictable, bedfellows. The vote patterns of elected representatives are defined by the rough confluence of ideology, constituent opinion, special interest lobbing, back-door deal making, and the broadly defined art of politicking. As votes on particular pieces of legislation are binary expressions of these influences, the relative authority of these influences is far from transparent. This suppression hinders traditional predictive analysis. Here, we train an artificial neural network to predict the voting patterns of the 107th session of the US Senate. In so doing, we quantify the influence of various political lobbies (broadly defined) and thus yield an insightful secondary analysis.

Keywords

neural network, back-propagation, US Senate, voting analysis, campaign finance

1. Introduction

1.1. "Vote Counting"

There are endemic inaccuracies in the rough speculation that comprise status quo "vote counting." For example, currently when members of the US Senate consider bringing legislation to the floor, potential votes are estimated by their party's Whip's office. These tallies, which often define the legislative agenda, are constructed by intuition, informal conversations, and cursory analysis of previous vote patterns. Consider further, a President deciding whether to veto a piece of legislation susceptible to an override. Here important legislative decisions rely on congressional staffer gossip and innuendo, filtered through executive-branch staffer innuendo and gossip. In contrast to this glorified "tealeaf reading", we aim to develop an artificial neural network to more accurately and systematically predict voting patterns in the US Senate.

In predicting a vote on a particular piece of legislation, it is tempting, but not sufficient, to ask simply how a senator has voted previously on similar issues, or cynically to construe ideological alignments as determinative. While this may seem often to yield accurate results, these heuristics are accurate primarily in situations that are not our concern here. The much more difficult question on which we focus is predicting votes in situations where values and political pressures conflict and where anomalous results are the most important to predict. In understanding the relative weight of political influences in these situations, training artificial neural networks based upon training data derived from voting histories are particularly useful.

[1] John.eure@yale.edu
[2] Jonathan.winer@yale.edu

1.2. Formalizing the Legislative Process (Our Assumptions)

In order to consider the legislative process in a meaningfully quantitative way, we first need to develop a creative and valid interpretation of the voting arena. The problem here is to express a Senator's vote on a particular bill in terms of the influences that shape the vote. That is, to express legislation in terms of the political pressures it engenders. If we can do this, we can express future legislation in the same manner and predict a Senator's vote based upon the opinions of a finite, defined set of entities. As there is the potential for infinite complexity in amassing factors from political affiliation to the weather the day a vote was cast, a formal model is defined by choices we make in limiting the factors considered as influencing a vote.

Our assumptions in this regard are meant to demonstrate method, not necessarily maximize accuracy. We consider a piece of legislation to be defined by the positions of various political lobbies including think tanks, interest groups, and powerful lobbyists. To make our system more accurate we would undoubtedly need to expand our definition to account for other factors such as the position of the President, opinion polls of constituents, and the type of legislation being considered.[3] We choose this definition of legislation because it offers the best combination of intuitive appeal, pragmatic feasibility, and flexibility in including broad points of view. The opinions of such groups are precisely verifiable from published accounts, yet in choosing which organizations to consider as data points, we include varied types of influence (lobbyist, scholarly, etc) and broad and narrow ideological affiliations. It seems reasonable to expect that to a significant degree the entities not explicitly referenced in our model, are highly correlated to the positions of the organizations considered and thus implicitly referenced. Finally, it is not a coincidence that the organizations we isolate as determinative of voting patterns correspond to entities generally considered to exert great, and perhaps undue, influence on the political process.

In defining a particular Senator's vote on a piece of legislation, we only consider the Senator's vote on the "final passage" of a Senate resolution. This limitation is intended to normalize for the numerous procedural votes leading up to final passage. This assumption could be made more accurate by considering significant amendments as separate legislation, or in a more sophisticated manner, by weighing political posturing on amendments as scalar values multiplied against a Senator's final vote.

2. Network Design

2.1 Perceptron with Back-Propagation

We experimented with several network configurations during this project, using Matlab as our environment. Our input data consisted of 24 interest groups, and 15 different bills which they had opinions on, represented as spin variables with a -1 meaning "not in favor", a 0 meaning "no opinion", and a 1 meaning "in favor". The target results were each senator's vote on each of those bills, as spin variables with a -1 meaning a vote of "Nay", a 1 meaning a vote of "Yea", and a 0 meaning "Not Voting" or "Present". The basic net was a 2-layer feed-forward network with 24 inputs, 10 hidden

[3] For example, a libertarian Republican might favor a particular alignment of political pressures when considering economic legislation and a very different alignment in considering social issues.

nodes, and 100 output nodes. We used the hyperbolic tangent sigmoid function as our limiter for both layers (its range is from -1 to 1). After trying a number of different learning algorithms, we settled on "resilient back-propagation" as the most suitable for our net. It performed several orders of magnitude better (in speed and accuracy) than any other appropriate algorithm. In particular, it was at least 5 orders of magnitude faster than the standard gradient-descent algorithm.

Resilient back-propagation is an algorithm designed to eliminate some harmful effects of sigmoidal "squashing" functions on the gradient-descent algorithm. Instead of basing the amount of a weight or bias's change on the value of the gradient (which approaches 0 for large inputs into a sigmoidal function, regardless of accuracy), it instead changes each weight and bias by an individual value, and merely uses the gradient to calculate the sign of the change. These values are increased by a fixed delta whenever subsequent changes are in the same direction, and are reduced by a (potentially different) fixed delta whenever subsequent changes are in opposite directions (i.e., when the gradient oscillates).

We tried a variety of network structures, and found that increasing the hidden layer's size beyond 10 neurons produced little benefit, while reducing its size led to steadily-increasing errors in the final net. We also tried using no hidden layer at all, treating the network as 100 individual 1-layer perceptrons.

2.2 Training data

Appendix A shows the data used in training the neural network that forms the basis of our analysis. There were important constraints in compiling the data both with respect to the votes considered and the political entities included. To date, there have only been approximately 400 votes cast by the 107th session of the US Senate and of these only 22 were deemed significant and sufficiently controversial to warrant inclusion. The emphasis on confirmation votes reflects the role of nominations as an umbrella vote on policies. The vote sample could be expanded by including previous sessions and normalizing for more senior members voting histories.

In selecting which political lobbies to include in defining a piece of legislation, we used a number of criteria. The sample set must be balanced and broad in ideological disposition. Some entities are narrowly focused on specific issue areas while the majority of groups have a broad focus. Finally, the set ranges from purely political to more scholarly groups. Within these balancing constraints, we considered the groups generally considered to be the most powerful lobbies.

3. EXPERIMENT

3.1 Hypothesis

Our hypothesis is that it is possible to accurately represent the voting patterns of the United States Senate as a neural network, using the opinions of various interest groups as input.

3.2 Description of Experiment

We obtained information from 24 interest groups about 15 separate Senate bills, and obtained the results of the Senate votes on those bills from the Senate's web site,

using an automated Perl script. We then used Matlab to train a variety of neural networks using this data, varying the number of neurons in the hidden layer (0-15, and 100), and varying the training algorithm used. After picking the most successful design, we tested it 4 times by removing one data point, and seeing how well a net trained on that limited data would handle the missing data point. As a further test, we used it to predict the result of a vote that will soon be made.

3.3 Results

Increasing the number of neurons in the hidden layer led to an increase in accuracy. This increase failed to be significant after 10 neurons, and there was only a marginal gain in accuracy from having 100 hidden neurons.

Interestingly, the most accurate version of the network was the 1-layer perceptron, using no hidden layers at all. On multiple occasions, it achieved a margin of error of less than 0.000001, while the 2-layer nets never managed to achieve an error less than 0.001. This result is quite interesting, as it suggests that the voting patterns of individual senators are linearly separable in dimensions representing interest groups, and thus that they follow patterns and are not subject to random chance or whim. Unfortunately, the data we collected is too limited to draw strong conclusions, but some of our test results are very promising. In particular, the test wherein we predicted the result of the upcoming vote produces an answer extremely close to that predicted by political analysts.

3.3 Discussion of Difficulties

One of the chief problems in training the net was locating data. While the senatorial votes were fairly easy to locate, and were standardized in format (thus allowing a single Perl script to download arbitrary numbers of votes within seconds), data from the interest groups was more difficult to obtain. Not all groups have opinions on all issues, and not all of the issues they have opinions on translate directly into senatorial roll call votes. Furthermore, each of the interest groups have their own means of presenting information, thus necessitating a more individualized and complicated approach to data collection.

This led to the second main problem, which was a lack of data. Our neural net depends far too much on each of the individual pieces of data we've collected, as evidenced by the massive error we get when removing even one vote. Another difficulty, especially with the 2-layer nets, is that we used too many neurons for the amount of data, and thus have overfitting. This is one of the reasons we used the single-layer perceptron for our tests. There is, however, a strong indication that we may have just enough data to be accurate, since the 1-layer net's prediction of the upcoming vote lines up with that of actual political analysts.

4. Prediction and Analysis

4.1 Predictions for the 107th Session of the US Senate

To analyze the predictive accuracy of our neural net, we retrained the net excluding one vote from the training sample. We then presented the excluded vote and asked the net to predict output for each senator. Appendix B displays the results of this

experiment. The four votes here represent opposite ends of our net's spectrum of accuracy. The campaign finance vote was dictated by political posturing. Further precisely the advocacy groups that were in endangered by this bill define our sample set. While our net's accuracy in this case is in the 40 percentile, our errors are almost always due to predicting inaccurate no votes. This is consistent with a self-interested data-set. The Ashcroft vote was also highly political, owing as much to his personal relationships as a former senator as to policy debate. Accordingly, our net was significantly inaccurate. In votes largely defined by policy debates, such as the Anti-terrorism bill and the Patient's Bill of Rights, our net averaged 94% accuracy. This inconsistency points to the narrowness of political lobbies as a data set.

If refined, the predicative power of this net has broad applications. Indeed, currently a perfect test case presents itself in a vote anticipated today in the Senate. The current Economic Stimulus bill is highly policy based and inspires spirited opinions from varied political lobbies. Originally, Senate vote-counters anticipated a 51-49 majority opposing the Bill. In December, President Bush traveled to the Capital to lobby moderate Democrats. Three Senators have switched their allegiances, and the New York Times reports this morning that vote counters now anticipate a 52-48 majority in support of the Bill. This abrupt switch has lead majority leader Daschle to threaten a filibuster—effectively requiring 60 votes to pass the Bill. The Bush administration must now decided whether to attempt to lobby additional Senators at peril of being perceived as weak. Given a profile of the Economic Stimulus Bill, our net makes the following predication.

Profile

	Economic Stimulus
AARP	
ACLU	0
AFL-CIO	-1
Air Transport Association of America	0
American Association of Health Plans	-1
American Enterprise Institute	1
American Medical Association	0
American Public Power Association	0
CATO	1
Center for Constitutional Rights	0
Christian Coalition	0
Consumer Federation of America	0
Edison Electric Institute	0
Health Insurance Association of American	0
Independent Insurance Agencies of America	0
National Association of Manufaturers	1
National Consumer Bankruptcy Coalition	0
National Federation of Independent Business	1
NRA	0
Nuclear Energy Institute	0
Public Citizen	0
The Brookings Institute	-1
The Heritage Foundation	1
US Chamber of Commerce	1

Vote Prediction

Akaka, Daniel (D - HI)	-1	Corzine, Jon (D - NJ)	-1	Hollings, Ernest (D -	-1	Nelson, Ben (D - NE)	1
Allard, Wayne (R - CO)	1	Craig, Larry (R - ID)	1	Hutchinson, Tim (R -	1	Nickles, Don (R - OK)	1
Allen, George (R - VA)	1	Crapo, Mike (R - ID)	1	Hutchison, Kay Bailey	1	Reed, Jack (D - RI)	-1
Baucus, Max (D - MT)	1	Daschle, Thomas (D -	-1	Inhofe, James (R - OK)	1	Reid, Harry (D - NV)	-1
Bayh, Evan (D - IN)	-1	Dayton, Mark (D - MN)	-1	Inouye, Daniel (D - HI)	-1	Roberts, Pat (R - KS)	1
Bennett, Robert (R -	1	DeWine, Mike (R - OH)	1	Jeffords, James (I - VT)	-1	Rockefeller IV, John (D	-1
Biden Jr, Joseph (D -	-1	Dodd, Christopher (D -	-1	Johnson, Tim (D - SD)	1	Santorum, Rick (R -	1
Bingaman, Jeff (D -	-1	Domenici, Pete (R -	-1	Kennedy, Edward (D -	-1	Sarbanes, Paul (D -	-1
Bond, Christopher (R -	1	Dorgan, Byron (D - ND)	-1	Kerry, John (D - MA)	-1	Schumer, Charles (D -	-1
Boxer, Barbara (D -	-1	Durbin, Richard (D - IL)	-1	Kohl, Herb (D - WI)	1	Sessions, Jeff (R - AL)	1
Breaux, John (D - LA)	1	Edwards, John (D -	-1	Kyl, Jon (R - AZ)	1	Shelby, Richard (R -	1
Brownback, Sam (R -	1	Ensign, John (R - NV)	1	Landrieu, Mary (D - LA)	1	Smith, Bob (R - NH)	1
Bunning, Jim (R - KY)	1	Enzi, Mike (R - WY)	-1	Leahy, Patrick (D - VT)	-1	Smith, Gordon (R - OR)	1
Burns, Conrad (R - MT)	0.831	Feingold, Russell (D -	-1	Levin, Carl (D - MI)	-1	Snowe, Olympia (R -	1
Byrd, Robert (D - WV)	-1	Feinstein, Dianne (D -	1	Lieberman, Joseph (D -	-1	Specter, Arlen (R - PA)	1
Campbell, Ben	1	Fitzgerald, Peter (R -	1	Lincoln, Blanche (D -	1	Stabenow, Debbie (D -	-1
Cantwell, Maria (D -	-1	Frist, William (R - TN)	1	Lott, Trent (R - MS)	1	Stevens, Ted (R - AK)	-1
Carnahan, Jean (D -	1	Graham, Bob (D - FL)	-1	Lugar, Richard (R - IN)	-1	Thomas, Craig (R -	1
Carper, Thomas (D -	-1	Gramm, Phil (R - TX)	1	McCain, John (R - AZ)	1	Thompson, Fred (R -	-1
Chafee, Lincoln (R -	1	Grassley, Chuck (R -	1	McConnell, Mitch (R -	1	Thurmond, Strom (R -	1
Cleland, Max (D - GA)	1	Gregg, Judd (R - NH)	1	Mikulski, Barbara (D -	-1	Torricelli, Robert (D -	-1
Clinton, Hillary (D - NY)	-1	Hagel, Charles (R -	1	Miller, Zell (D - GA)	0.998	Voinovich, George (R -	1
Cochran, Thad (R -	1	Harkin, Tom (D - IA)	-1	Murkowski, Frank (R -	0.984	Warner, John (R - VA)	1
Collins, Susan (R - ME)	-0.44	Hatch, Orrin (R - UT)	1	Murray, Patty (D - WA)	-1	Wellstone, Paul (D -	-1
Conrad, Kent (D - ND)	-1	Helms, Jesse (R - NC)	1	Nelson, Bill (D - FL)	-1	Wyden, Ron (D - OR)	-1

Table #1: Vote Prediction for Economic Stimulus Bill

Our Net predicts a 54-46 majority in favor of the Bill—insufficient votes to override the filibuster. Significantly, our net accurately anticipates the last-minute switch by Democrats Breaux, Nelson, and Miller.

4.2 Analysis of Influence

Over our set of training data, the neural network described above converges to assign a synaptic weight to each element of an input vector. Each element of the input vector represents the policy position of a particular advocacy group. Thus, we conjecture that the weight assigned to each element of the input vector for each senator reflects the political influence of the given organization on that senator. To investigate this hypothesis, we conducted a secondary analysis of the same converging net used in the previous section to make predictions. We considered the voting pattern over our training set of three well-known, senior senators (a Republican, a Republican/Independent, and a Democrat). Below are the synaptic weights assigned to each element of the input vector and the percentage of times over the set of training data a vote corresponded to the element of the input vector when that input was not 0.

	Daschle	%Voted	Jeffords	%Voted	McCain	%Voted
AARP	-21.95	60%	15.88	50%	20.4	83%
ACLU	-13.01	16.00%	1.83	28.50%	-0.67	28.50%
AFL-CIO	0.86	62.50%	-3.02	55.50%	14.93	66.60%
Air Transport Association of America	3.3	66%	1.12	75%	11.03	66%
American Association of Health Plans	-16.67	0%	8.07	75%	-14.62	25%
American Enterprise Institute	2.16	54%	0.71	90%	0	80%
American Medical Association	16.07	100%	-8.02	50%	14.72	100%
American Public Power Association	14.24	71.40%	20.29	87.50%	18.03	75%
CATO	-12.03	44%	1.5	50%	0.12	30%
Center for Constitutional Rights	-4.43	60%	0	25%	-1.52	0%
Christian Coalition	3.1	44%	0.1	25%	0.1	0%
Consumer Federation of America	-5.55	25%	-34.45	25%	-2	0%
Edison Electric Institute	-30	0%	22.38	100%	10.76	100%
Health Insurance Association of American	-6.56	25%	-14.13	25%	-26.78	40%
Independent Insurance Agencies of America	2.87	50%	-1.19	50%	10.56	57%
National Association of Manufaturers	-7.02	25%	-1.12	50%	11.87	60%
National Consumer Bankruptcy Coalition	13.78	100%	14.41	100%	12.33	100%
National Federation of Independent Business	-5.28	33%	4.94	66%	13.95	66%
NRA	-2.09	40%	-0.01	75%	6.28	80%
Nuclear Energy Institute	-50.03	0%	0.96	100%	15.66	100%
Public Citizen	-9.3	40%	-18.9	20%	-0.13	25%
The Brookings Institute	-2.14	66%	-0.45	33%	0.18	41%
The Heritage Foundation	2.47	54%	0.28	90%	0.34	70%
US Chamber of Commerce	-39.6	0%	-42.96	0%	-24.04	0%
Correlation Coefficient		0.804083		0.699614		0.674126

Table #2: Analyzing influence of political lobbies

Finding a correlation here makes intuitive sense: we would expect that if a senator often voted in a way a group approved of, then the weight between them would higher. However, there are other factors that a simple correlation analysis does not take into account. One such factor is the trend that input vectors with less data often have higher weights. (That is, if a group has opinions on only a few bills, it tends to be ranked as more influential on that bill.) This is an artifact of the learning algorithm not being able to properly "judge" the true influence of the group from the limited data set.

Conclusion

The powerful applications of predicting voting patterns through neural networks are demonstrated by our net's prediction of today's Economic Stimulus bill. Nevertheless, we repeatedly encountered errors and difficulties that suggest the need for a more accurate definition of the voting arena. That is, a definition of a bill broader than the positions of various political advocacy groups. Our study suggests such exploration could yield highly accurate predictions.

APPENDIX A: SUMMARY OF TRAINING DATA

	S. 1052 Patient Protection Act	Gramm Amendment to	Campaign Finance Vt. 64	Bankrupcy Reform Vt. 36 (S.	Farm Security Act of 2001 H.R. 2646	Death Tax Senate 59-39	Ashcroft Nomination
AARP	1	1	1	0	0	1	
ACLU	0	0	-1	0	0	0	
AFL-CIO	-1	-1	1	1	0	0	
Air Transport Association of America	0	0	-1	0	0	0	
American Association of Health Plans	-1	1	0	0	0	0	
American Enterprise Institute	-1	-1	-1	1	1	1	
American Medical Association	1	-1	0	0	0	0	
American Public Power Association	0	0	-1	0	0	0	
CATO	-1	-1	-1	0	-1	1	
Center for Constitutional Rights	0	0	0	0	0	0	
Christian Coalition	1	1	-1	0	1	1	
Consumer Federation of America	1	1	0	-1	0	0	
Edison Electric Institute	0	0	0	0	0	0	
Health Insurance Association of American	-1	1	-1	0	0	0	
Independent Insurance Agencies of America	-1	1	-1	0	0	0	
National Association of Manufaturers	-1	-1	-1	0	0	0	
National Consumer Bankruptcy Coalition	0	0	0	1	0	0	
National Federation of Independent Business	-1	-1	0	0	0	0	
NRA	0	0	-1	0	0	0	
Nuclear Energy Institute	0	0	0	0	0	0	
Public Citizen	0	0	1	-1	0	1	
The Brookings Institute	1	1	1	0	1	-1	
The Heritage Foundation	-1	-1	-1	0	-1	1	
US Chamber of Commerce	0	0	-1	0	0	0	

	Passangers' Bill of Rights	Tax Relief Act Vt. 165	Olson Vt. 167	Nomination Health Vt. 220	Insurance Supplemental Appropriations 228	Gregory Nomination	Mueller Nomination	254 N. Truck R
AARP	0	1	0	1	0	0	0	
ACLU	0	0	0	-1	0	0	0	
AFL-CIO	0	0	0	1	0	0	0	
Air Transport Association of America	-1	0	0	0	0	0	0	
American Association of Health Plans	0	0	0	-1	0	0	0	
American Enterprise Institute	-1	1	1	0	0	1	1	
American Medical Association	0	0	0	1	0	0	0	
American Public Power Association	0	0	0	0	1	0	0	
CATO	1	1	-1	0	0	0	0	
Center for Constitutional Rights	0	0	-1	0	0	1	-1	
Christian Coalition	0	1	1	0	0	1	1	
Consumer Federation of America	0	0	0	0	0	0	0	
Edison Electric Institute	0	0	0	0	0	0	0	
Health Insurance Association of American	0	0	0	1	0	0	0	
Independent Insurance Agencies of America	0	0	0	1	0	0	0	
National Association of Manufaturers	0	0	0	0	0	0	0	
National Consumer Bankruptcy Coalition	0	0	0	0	0	0	0	
National Federation of Independent Business	0	0	0	0	0	0	0	
NRA	0	0	1	0	0	0	0	
Nuclear Energy Institute	0	0	1	0	0	0	0	
Public Citizen	1	0	0	0	0	-1	0	
The Brookings Institute	1	-1	-1	1	0	-1	-1	
The Heritage Foundation	0	1	1	0	0	1	1	
US Chamber of Commerce	0	0	0	0	0	0	0	

APPENDIX B: VOTE PREDICTION

	Patient Protection A	*Prediction*	Campaign Finance	*Prediction*	Ashcroft Nomiation	*Prediction*	Anti-Terrorism	*Prediction*
Akaka, Daniel (D - HI)	1	*1*	1	*-1*	-1	*1*	1	*1*
Allard, Wayne (R - CO)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Allen, George (R - VA)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Baucus, Max (D - MT)	1	*1*	1	*-1*	-1	*1*	1	*1*
Bayh, Evan (D - IN)	1	*1*	1	*-1*	-1	*1*	1	*1*
Bennett, Robert (R - UT)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Biden Jr, Joseph (D - DE)	1	*1*	1	*-1*	-1	*1*	1	*1*
Bingaman, Jeff (D - NM)	1	*1*	1	*-1*	-1	*1*	1	*1*
Bond, Christopher (R - MO)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Boxer, Barbara (D - CA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Breaux, John (D - LA)	1	*1*	-1	*-1*	1	*-0.5511*	1	*1*
Brownback, Sam (R - KS)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Bunning, Jim (R - KY)	-1	*-0.9726*	-1	*-1*	1	*1*	1	*1*
Burns, Conrad (R - MT)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Byrd, Robert (D - WV)	1	*-0.9988*	1	*-1*	1	*1*	1	*1*
Campbell, Ben Nighthorse	0	*1*	-1	*-1*	1	*1*	1	*1*
Cantwell, Maria (D - WA)	1	*0.9735*	1	*-1*	-1	*1*	1	*1*
Carnahan, Jean (D - MO)	1	*1*	1	*-1*	-1	*1*	1	*1*
Carper, Thomas (D - DE)	1	*1*	1	*-1*	-1	*1*	1	*1*
Chafee, Lincoln (R - RI)	1	*1*	1	*-1*	1	*1*	1	*1*
Cleland, Max (D - GA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Clinton, Hillary (D - NY)	1	*1*	1	*-1*	-1	*1*	1	*1*
Cochran, Thad (R - MS)	-1	*-1*	1	*-1*	1	*1*	1	*1*
Collins, Susan (R - ME)	1	*1*	1	*-1*	1	*1*	1	*1*
Conrad, Kent (D - ND)	1	*-0.9946*	1	*-1*	1	*1*	1	*1*
Corzine, Jon (D - NJ)	1	*1*	1	*-1*	-1	*1*	1	*1*
Craig, Larry (R - ID)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Crapo, Mike (R - ID)	-1	*-0.9993*	-1	*-1*	1	*1*	1	*1*
Daschle, Thomas (D - SD)	1	*1*	1	*-1*	-1	*1*	1	*1*
Dayton, Mark (D - MN)	1	*1*	1	*-1*	-1	*1*	1	*1*
DeWine, Mike (R - OH)	1	*-1*	-1	*-1*	1	*1*	1	*1*
Dodd, Christopher (D - CT)	1	*1*	1	*-1*	1	*1*	1	*1*
Domenici, Pete (R - NM)	0	*1*	1	*-1*	1	*1*	0	*1*
Dorgan, Byron (D - ND)	1	*1*	1	*-1*	1	*-1*	1	*1*
Durbin, Richard (D - IL)	1	*1*	1	*-1*	-1	*1*	1	*1*
Edwards, John (D - NC)	1	*1*	1	*-1*	-1	*1*	1	*1*
Ensign, John (R - NV)	-1	*-0.995*	-1	*-1*	1	*1*	1	*1*
Enzi, Mike (R - WY)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Feingold, Russell (D - WI)	1	*1*	1	*-1*	1	*1*	-1	*1*
Feinstein, Dianne (D - CA)	1	*1*	1	*-1*	-1	*-1*	1	*1*
Fitzgerald, Peter (R - IL)	1	*1*	1	*-1*	1	*1*	1	*-1*
Frist, William (R - TN)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Graham, Bob (D - FL)	1	*1*	1	*-1*	-1	*1*	1	*1*
Gramm, Phil (R - TX)	0	*1*	-1	*-1*	1	*1*	1	*1*
Grassley, Chuck (R - IA)	-1	*0.2732*	-1	*-1*	1	*1*	1	*1*
Gregg, Judd (R - NH)	-1	*-0.4151*	-1	*-1*	1	*1*	1	*1*
Hagel, Charles (R - NE)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Harkin, Tom (D - IA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Hatch, Orrin (R - UT)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Helms, Jesse (R - NC)	-1	*-1*	-1	*-1*	1	*1*	0	*1*
Hollings, Ernest (D - SC)	1	*-1*	-1	*-1*	-1	*1*	1	*1*
Hutchinson, Tim (R - AR)	-1	*0.9802*	-1	*-1*	1	*1*	1	*1*
Hutchison, Kay Bailey (R -	-1	*1*	-1	*-1*	1	*1*	1	*1*
Inhofe, James (R - OK)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Inouye, Daniel (D - HI)	1	*1*	1	*-1*	-1	*1*	1	*1*
Jeffords, James (I - VT)	-1	*1*	1	*-1*	1	*-1*	1	*1*
Johnson, Tim (D - SD)	1	*1*	1	*-0.9954*	-1	*1*	1	*1*
Kennedy, Edward (D - MA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Kerry, John (D - MA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Kohl, Herb (D - WI)	1	*1*	1	*-1*	-1	*1*	1	*1*
Kyl, Jon (R - AZ)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Landrieu, Mary (D - LA)	1	*1*	1	*-0.9997*	-1	*1*	1	*1*
Leahy, Patrick (D - VT)	1	*1*	1	*-1*	-1	*1*	1	*1*
Levin, Carl (D - MI)	1	*1*	1	*-1*	-1	*1*	1	*1*
Lieberman, Joseph (D - CT)	1	*1*	1	*-1*	-1	*1*	1	*1*
Lincoln, Blanche (D - AR)	1	*1*	1	*-1*	-1	*0.9962*	1	*1*
Lott, Trent (R - MS)	0	*1*	-1	*-1*	1	*1*	1	*1*
Lugar, Richard (R - IN)	-1	*-1*	1	*-1*	1	*1*	1	*1*
McCain, John (R - AZ)	1	*1*	1	*-1*	1	*1*	1	*1*
McConnell, Mitch (R - KY)	-1	*-1*	-1	*-1*	1	*1*	1	*1*
Mikulski, Barbara (D - MD)	1	*1*	1	*-1*	-1	*1*	1	*1*
Miller, Zell (D - GA)	1	*1*	1	*-1*	1	*1*	1	*1*
Murkowski, Frank (R - AK)	0	*1*	-1	*-1*	1	*1*	1	*1*
Murray, Patty (D - WA)	1	*1*	1	*-1*	-1	*1*	1	*1*
Nelson, Bill (D - FL)	1	*1*	1	*-1*	-1	*1*	1	*1*
Nelson, Ben (D - NE)	1	*-1*	-1	*-1*	1	*1*	1	*1*

APPENDIX B: VOTE PREDICTION CONTINUED

	Patient Protection A	*Prediction*	Campaign Finance	*Prediction*	Ashcroft Nomiation	*Prediction*	Anti-Terrorism	*Prediction*	
Nickles, Don (R - OK)	-1	*-1*	-1	*-1*	1	*1*	1	*1*	
Reed, Jack (D - RI)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Reid, Harry (D - NV)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Roberts, Pat (R - KS)	-1	*-1*	-1	*-1*	1	*1*	1	*1*	
Rockefeller IV, John (D -	1	*1*	1	*-1*	-1	*1*	1	*1*	
Santorum, Rick (R - PA)	-1	*-1*	-1	*-1*	1	*1*	1	*1*	
Sarbanes, Paul (D - MD)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Schumer, Charles (D - NY)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Sessions, Jeff (R - AL)	-1	*-1*	-1	*-1*	1	*1*	1	*1*	
Shelby, Richard (R - AL)	-1	*0.9999*	-1	*-1*	1	*1*	1	*1*	
Smith, Bob (R - NH)	-1	*1*	-1	*-1*	1	*1*	1	*1*	
Smith, Gordon (R - OR)	1	*1*	-1	*-1*	1	*1*	1	*1*	
Snowe, Olympia (R - ME)	1	*1*	1	*-1*	1	*1*	1	*1*	
Specter, Arlen (R - PA)	1	*1*	1	*-1*	1	*1*	1	*1*	
Stabenow, Debbie (D - MI)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Stevens, Ted (R - AK)	-1	*-1*	1	*-1*	1	*1*	1	*1*	
Thomas, Craig (R - WY)	-1	*-1*	-1	*-1*	1	*1*	1	*1*	
Thompson, Fred (R - TN)	-1	*-1*	1	*-1*	1	*1*	1	*1*	
Thurmond, Strom (R - SC)	-1	*-1*	-1	*-1*	1	*1*	0	*1*	
Torricelli, Robert (D - NJ)	1	*1*	1	*-0.9999*	-1	*1*	1	*1*	
Voinovich, George (R - OH)	-1	*0.9838*	-1	*-1*	1	*1*	1	*1*	
Warner, John (R - VA)	1	*1*	-1	*-1*	1	*1*	1	*1*	
Wellstone, Paul (D - MN)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Wyden, Ron (D - OR)	1	*1*	1	*-1*	-1	*1*	1	*1*	
Total Red		*11*		*57*		*43*		*2*	
Total Green		*8*		*0*		*0*		*3*	
% Correct		*89*		*43*		*57*		*98*	
Sum Deviation		*28.8*		*117*		*87*		*7*	

References

Hassoun, Mohamad. Fundamentals of Artificial Neural Networks. MIT Press: Cambridge,1995.

Haykin, Simon. Nueral Networks A Comprehensive Foundation. Pretencie Hall: New Jersey, 1999.

Would You Like to Play a Game? - An Evolutionary Programming Approach to Three-dimensional Tic-Tac-Toe

Matthew Herberg and Andrew Lovett
Yale University, Neural Networks
New Haven, CT 06520

Abstract

We attempt to create an ideal three-dimensional tic-tac-toe player. Examining the strengths and weakness of previous game-playing algorithms, we explore the plausibility of combining these algorithms in order to create a player which trains against a specific adversary until it becomes as skilled as possible at defeating that opponent. We do this by implementing a genetic algorithm that uses a marker-based encoding system patterned loosely off actual DNA structures. This algorithm uses crossover and mutation to find the ideal neural network. We compare this to simpler genetic algorithms which have previously been used to play tic-tac-toe. We test our algorithms against AI opponents of varying difficulty to see whether our networks can learn to beat the adversaries.
Keywords – neural networks, genetic algorithms, evolutionary programming, sexual selection, mutation, marker-based encoding, three-dimensional tic-tac-toe, qubic

1. INTRODUCTION

When studying game learning, it is often most enlightening to examine human strategies. Though we can imagine Kasparov or a similar expert in a game "computing" a solution by searching deeper and deeper among possible game trees, in reality one study experts in chess reported that a move just appears to them as being correct (deGroot, 1965). In fact, deGroot found that most chess masters have spent thousands of hours staring at chess boards studying chess and that they have stored in memory the moves from thousands of games. They do not, as one would expect, search any more moves than a novice player would. Instead, their choice in moves is based off a kind of pattern

recognition. Moriarty and Miikkulainen (1995) adopted a similar "human-like" pattern recognition strategy to train neural networks to play Othello. Without any heuristics, game strategies, or look-ahead, they successfully taught a network to look at a board and return a "good" move. Their network even successfully generated strategies for beating artificial intelligence algorithms. In our project, we hope to emulate their success with the game of three-dimensional tic-tac-toe.

Three-dimensional tic-tac-toe is derived from traditional tic-tac-toe, which is a very simple game in which two players alternate moves placing a symbol on the board thereby hoping to make a line of three of their symbols in a row. On the traditional board of nine squares, there exist only eight possible lines of three symbols. Three-dimensional tic-tac-toe, on the other hand, has a far larger state space and consequently is far more complicated and can be played on any N x N x N board. For purpose of our project, we have decided to use the two most common variations. The first variation is played on a 3 x 3 x 3 board and the second is known as "qubic" and is played on a 4 x 4 x 4 board. For example, in qubic the board consists of 64 squares and there are 76 possible winning lines of four symbols in a row. An example of an ending to one particular game can be found in *Figure 1.*

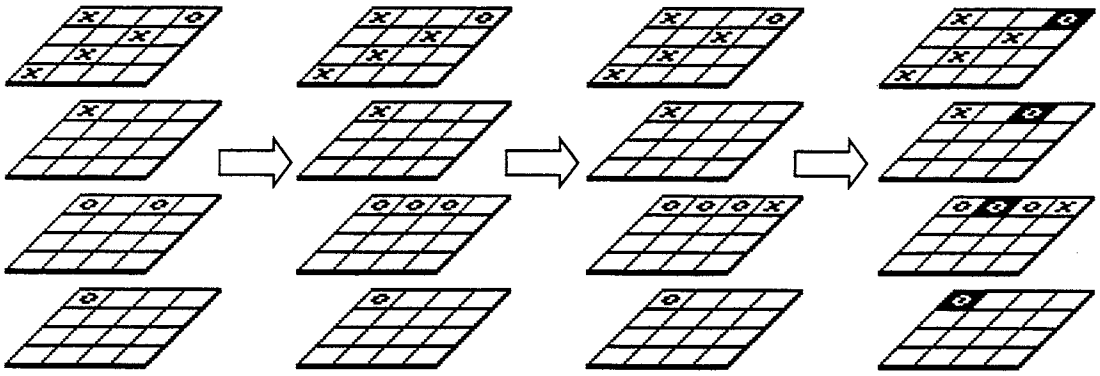


Figure 1: *Player "O" fills in a spot to create three in a row, "X" blocks "O", and "O" completes a line of four, winning the game.*

In typical artificial intelligence (AI) implementations of a computer player, game playing rules are hard-coded. One such example is when the AI examines

at every possible move and determines the priority of each move. Such implementations also often look-ahead at all possible responses to their moves (often using minimax and alpha-beta pruning to eliminate some of the possible moves) and use a formula, called a heuristic estimator, to evaluate their resulting positions (Russell & Norvig, 1995).

There are a few problems with the traditional AI approach. First of all, state space pruning usually assumes the opponent is making the best possible moves (according to its heuristic). Consequently, it is possible to trick the traditional player by using unconventional strategies. Similarly, because the heuristic estimator is hard-coded, a change in the dimensions of the game would almost necessarily require rewriting the estimator. In many cases it has been shown that game playing techniques that combine evolutionary algorithms with neural networks can be successful, even when limited training is given to the network (Chellapilla & Fogel, 1999). The goal of this paper is to discuss and examine strategies for training neural nets to play tic-tac-toe against specific opponents on three-dimensional boards of any size.

We will design and implement two different genetic algorithms. The first one, based on Fogel's experiments with two-dimensional tic-tac-toe, will use simple mutation to reproduce neural networks. The second one, based on Othello-playing strategies developed by Moriarty and Miikkulainen, will evolve networks using marker-based encoding, a system that allows networks to reproduce sexually. These two algorithms will be tested against a variety of adversaries, some based on proven artificial intelligence techniques.

2. EXPERIMENT ONE

2.1.1. A Simple Model

David Fogel has conducted extensive experiments using evolutionary programming and neural networks to play two-dimensional tic-tac-toe (Fogel, 1993; Fogel, 2000). Although this is a much simpler game, many of the concepts he applied to the creation of his neural net are worth examining because Fogel's strategies can be applied to higher-dimensional tic-tac-toe. Fogel's feed-forward net had nine input values. These values corresponded to the state at each spot

on a 3 x 3 tic-tac-toe board. Each value could be [-1, 0, 1], representing an "X", an empty square, and an "O." The input layer was completely connected to a hidden layer consisting of 1 to 9 neurons. These neurons, in turn, were connected to an output layer of nine neurons. This layer was a modification of the input layer that suggested the next move through competition, i.e. the neuron with the highest weighted sum represented the suggested legal move. The algorithm began by creating a family of 50 neural networks. It trained the neural networks by testing every net in the family against an AI tic-tac-toe player. Each generation, the nets that performed especially well produced offspring through mutation, in which the weights of the offspring mutated randomly and there was a chance that the hidden layer would increase or decrease by one neuron. Those nets that performed the worst were dropped.

2.1.2. Applying Fogel's Model to a Three-dimensional Space

For our initial attempt, we decided to create a relatively simple neural network, loosely based on Fogel's work. The first genetic algorithm was designed to work with boards of all size, but as our initial attempts involved only a 3 x 3 x 3 board, we will describe it in the terms of such a board. This completely connected feed-forward net consisted of 27 inputs, a hidden layer with 54 neurons, and 27 outputs. We decided to use a much greater number of hidden neurons than Fogel used because of the significantly greater complexity of our game. We decided it was unnecessary to vary the number of neurons in the hidden layer because the possibility of the outputs from a given neuron having weights of or close to 0 had the same effect. All weights, including biases, were initially set to a value between -5 and 5 determined randomly when the net was created. Our gain function returned 1 or –1 for the output of each neuron.

2.1.3. Implementing our First Genetic Algorithm

The genetic algorithm used a family of 20 neural nets. Each generation, these 20 nets played 50 games against the adversary. Each net's total fitness was the average of its fitness scores over all the games. This fitness in a given game was determined in the following manner using the total number of moves made in a given game (t):

1. If the neural net won, the score was 128 - t.
2. If the neural net lost, the score was -128 + t.
3. In the case of a tie, the score was 0.

This system was chosen because we wanted a win in a smaller number of moves or a loss in a greater number of moves to be recognized as an improvement, but we also required there to be a clear distinction between any win and any loss.

After the fitness of each net in the family was determined, the family was sorted according to fitness. The bottom three quarters were discarded. Each net in the top quarter was replicated three times and mutated to replace the discarded families. Neural networks were mutated by adding a random value from the set of either {-1, 0, 1} or {-2, -1, 0, 1, 2} to each weight. The next generation then commenced using the new family.

2.2. Results

We first tested our genetic algorithm by training our neural nets to play against a random player. This player randomly selected a legal move each turn. Our genetic algorithm worked very well against this adversary on a 3 x 3 x 3 board. As *Figure 2* shows, after merely 50 generations, the highest-performing net in the family achieved a fitness of around 121. This means that, on average, the neural net was defeating the random player in just 7 moves. In repeated tests, these results always held.

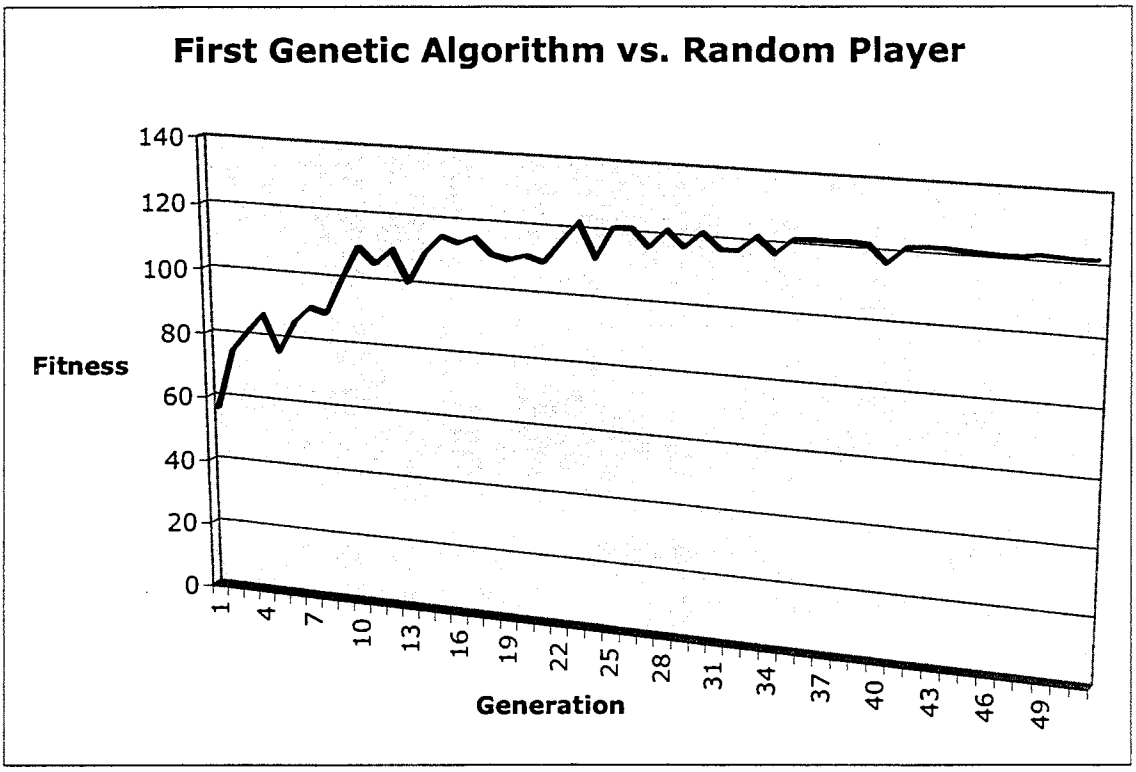


Figure 2: The first genetic algorithm easily beat the random player, stabilizing on a fitness of 120 by the 50th generation

We next trained our neural nets to play against a simple AI opponent. This player moved to block or take a win whenever it had the option of doing so or otherwise moved randomly. The genetic algorithm was also successful in overcoming this adversary, although the training time was much greater. After 543 generations, the top neural net of the first genetic algorithm consistently scored a fitness level above 120 (*Figure 3*). This meant that the algorithm was winning in approximately 7 to 9 moves.

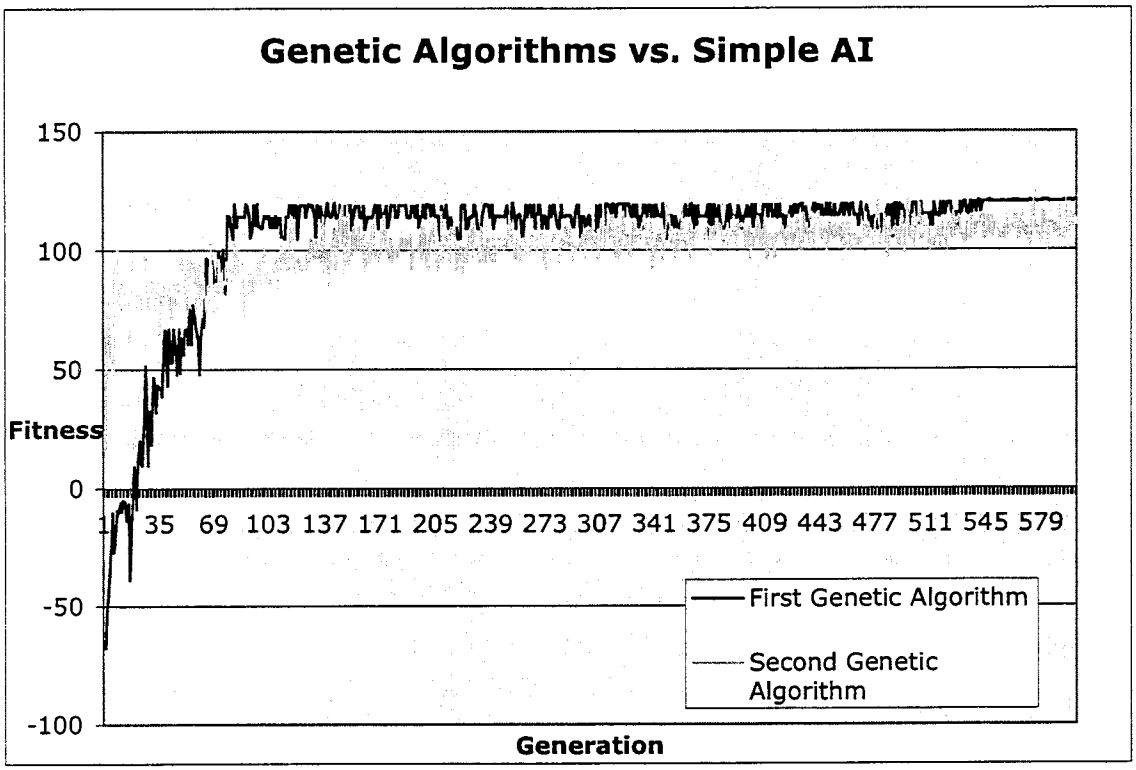


Figure 3: The performance of both genetic algorithms against a simple AI algorithm

We next used a more complicated AI which analyzed the board and found the move with the highest priority based partially on ideas presented in a article from an online artificial intelligence website (Kao, http://www.generation5.org). Initial tests against this adversary were also very promising. However, we soon discovered that much of our success was due to flaws in the AI's design. Indeed, we have concluded that one of the greatest uses of neural networks may be testing AI algorithms, as they are able to quickly find any weaknesses. All we had to do was examine how the net which had been training against the AI played to determine what the AI was doing wrong. We eventually managed to fix most of the flaws in the AI, only to discover that our neural net had a considerably greater amount of difficulty training against the new AI. It was able to reach a fitness of around 30, meaning it was playing slightly better than the AI, without too much difficulty, but it was unable to improve beyond this level. Unlike playing a simpler AI in which it won in an average of 7 moves, a fitness of 30 implies that it took on average close to 90 moves to win.

3. EXPERIMENT TWO

3.1.1. Marker-based Encoding

Marker-based encoding is a direct encoding of a neural network onto a chromosome consisting of integer alleles (Moriarty & Miikkulainen, 1995). Certain allele values act as *start markers* or *end markers*. The neural network generated is defined by node definitions, which are read sequentially from the chromosome. Each node is begun by a valid start marker and ends only when terminated by a valid end marker (any start markers within a node definition are ignored).

Inspired by the biological structure of DNA, Moriarty and Miikkulainen used markers to define separate nodes in a neural network. Each node, or gene, contained a start and end marker. In between these two markers were 8-bit integers that represented keys, labels, and weights. The first value was the label, which specified the identity of that particular neuron. Key-label-weight triplets followed that specified the connections between that neuron and the input layer, the output layer, and other neurons. Neurons could even send input to themselves. Recurrency was implemented so that a net could adapt to a specific opponent's strategy during a game. The input and output layers, rather than being specified by the genes, were created beforehand. There were 128 inputs representing the board's current state. The 128 inputs represented all the spaces on the 8 x 8 board taken by white pieces and all the spaces taken by black pieces (those inputs for spaces that were taken were set to 1). There were 64 outputs, representing the weight given to each potential move.

3.1.2. Defining Our Encoding System

Our marker-based encoding system was similar to the one used by Moriarty (*Figure 4*). Our algorithm constructed a neural net from a string of 5000 bytes (numbers between the values of -128 and 127), using these values to build the hidden layer connecting the input and output layers. Genes representing neurons in the neural network were found between start markers and end markers. If the value of a number MOD 25 was 1 it was a start marker, and if the value of the number MOD 25 was 2 it was an end marker. The first label

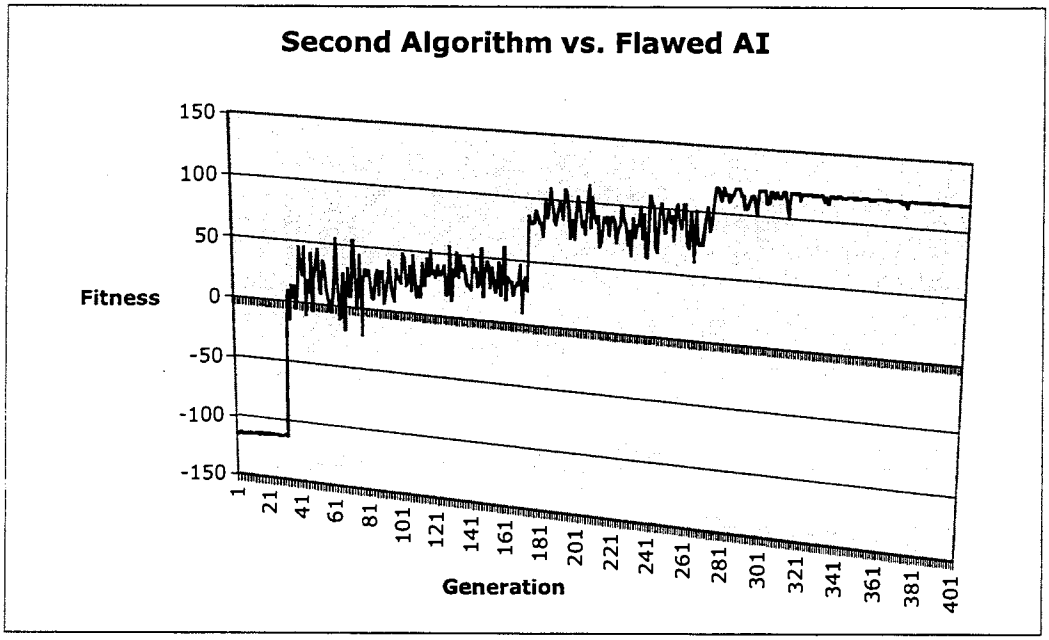


Figure 5: Second genetic algorithm training against an 4 x 4 x 4 AI algorithm with a flaw.

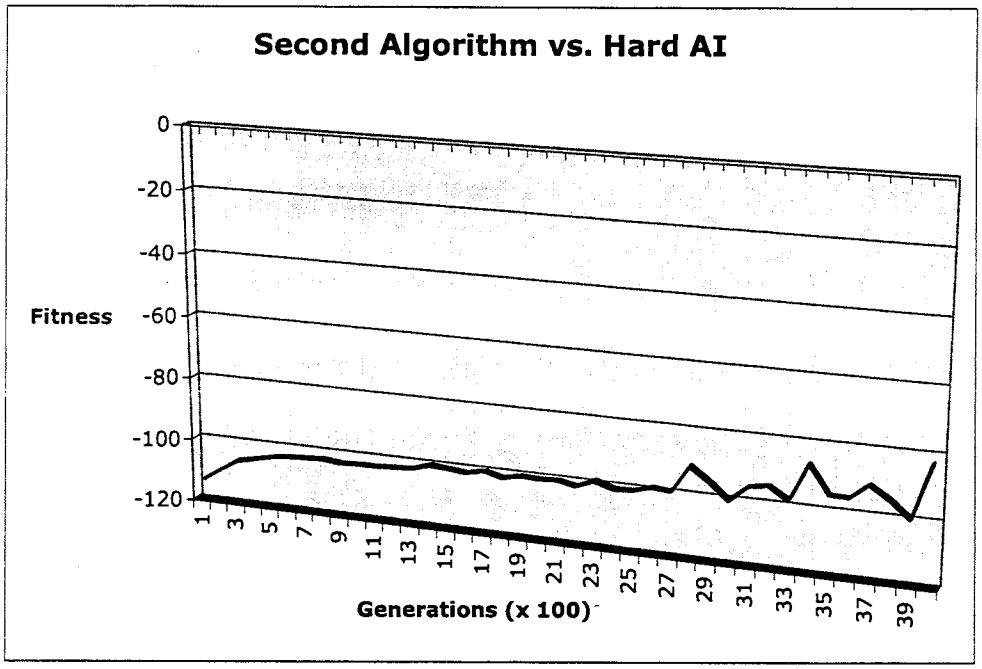


Figure 6: Second genetic algorithm training against a near perfect 4 x 4 x 4 AI algorithm.

identified the neuron, and the remaining values in the gene describe connections to inputs, outputs, and other hidden neuron. Because labels for neurons were bytes, there could be up to 256 neurons in the hidden layer.

Figure 4: *A typical "gene" as defined by our marker-based encoding system*

The main difference was that we eliminated the recurrency. Moriarty himself stated that the recurrency was not necessary and was only used in the Othello program to aid the net in adapting to specific opponents' strategies. Because we planned to train our net against specific opponents, this seemed unnecessary. In order to avoid recurrency, we ensured that each hidden neuron could only receive input from other hidden neurons with lower labels. This meant that, as in Moriarty's model, the neural net could have anywhere from one to 256 hidden layers, but that as long as the neurons were updated in order, according to their labels, no neuron would be updated before another neuron from which it received input. All neurons, however, could receive input from the input layer or send output to the output layer. Because all input values were either 1 or 0, we switched to a gain function that returned 1 or 0.

3.1.3. Implementing Our Second Genetic Algorithm

In our genetic algorithm, we created a family of 50 networks. Again, these networks played 50 games each against an adversary, and the fitness was determined in the same manner as before. After sorting the networks according to fitness, we followed Moriarty's strategy of taking the top 12 networks and reproducing each of them with another one of the top 12 to produce two child networks. These 24 new networks replaced the 24 networks from the previous generation with the lowest fitness. Reproduction occurred by picking two points at random along the genetic string of 5000 bytes representing the nets. The net consisted of a combination of the bytes that came between the two points in one parent string and the bytes found outside of those two points on the other parent string. There was a 10% chance that each byte would mutate by having a value

between -5 and 5 added to it. We choose these values for mutation because they were similar in scale to mutations done in similar literature and worked well in practice.

3.2. Improving Our AI Adversary

To aid in testing our new genetic algorithm, we modified the AI to play on a 4 x 4 x 4 board. Remembering our problems with our old AI, we changed our board analysis to use a ranking system. Under this system, a move which gave the AI four-in-a-row was awarded the top rank of 20. Moves which accomplished lower-priority goals, such as blocking an opponent's four-in-a-row (19), achieving three-in-a-row in two directions (18), blocking an opponent's four-in-a-row in two directions (16), etc, were given lower ranks. It should be noted that modifying the AI to play on a 4 x 4 x 4 board took a considerable amount of effort. One of our initial points was that a genetic algorithm is better suited for adapting to changes in board size than an artificial intelligence player.

3.3. Results

The second genetic algorithm performed even better than the first against random tic-tac-toe players. On both a 3 x 3 x 3 board and a 4 x 4 x 4 board, the algorithm trained a network with a fitness of around 120 in under 10 generations. The algorithm also performed well against the simple AI on a 3 x 3 x 3 board (*Figure 3*). It produced a network with a fitness near 100 in just 10 generations. Unlike the first genetic algorithm, it did not manage to consistently score a fitness of 120 within the first 600 generations. However, it did manage to achieve an average fitness of about 110.

The real test came when we pitted it against our 4 x 4 x 4 AI. In the initial run, the algorithm performed well, reaching an optimal fitness (*Figure 5*). However, we discovered that it was taking advantage of a flaw in the AI's program which caused it to miss diagonals that transcended all three dimensions. After we corrected this fault, the AI became unbeatable. Even after 2000 generations, the best neural net was unable to perform at a fitness level above -106. While this was an improvement over the initial fitness of -118, it was clear that the net was not improving beyond this level. We tried weakening our

AI somewhat by eliminating its ability to recognize anything with a lower priority than blocking opponent's three-in-a-row (in other words, it could not recognize the chance to get two-in-a-row), but the genetic algorithm still performed poorly, reaching a fitness above –90 only after 4,000 generations (*Figure 6*).

4. CONCLUSION

Both of our genetic algorithms created populations of networks that slowly improved by means of mutation. Our second genetic algorithm also generated new networks by using sexual selection and encouraging stronger networks to recombine with each other. The success of our algorithms against both a random opponent and a simple AI demonstrates the advantage of using these strategies.

The difference between the performances of our two genetic algorithms shows the strengths and weaknesses of the algorithms. The second algorithm was able to achieve a high fitness level quickly because of its use of sexual neural net reproduction to combine strong neural nets. However, this algorithm had greater difficulty actually achieving the optimal fitness level against a simple AI opponent. The greater changes introduced in the offspring of the neural nets, resulting both from the use of sexual reproduction and in the greater sensitivity to minute changes in the gene structure, allowed the nets to adapt quickly but made it difficult for them to undergo the slight changes required to achieve perfection.

Our lack of success against more skilled AI algorithms shows that we are a long way from teaching a neural network to play well enough to defeat a human. There are several possible reasons for this failure. It is difficult to determine the ideal level of mutation. A greater amount of mutation might have aided our networks in converging on a high fitness value. It is also possible that our neural networks were not complex enough to store all the rules required to play three-dimensional tic-tac-toe. The second algorithm was designed to allow a wide range of complexities, with up to 256 hidden neurons, but neural nets produced from a string of 5000 bytes generally had closer to 60 neurons.

Modifications to the gene interpretation strategy may aid the algorithm in learning more rules at a greater rate.

One of our biggest problems was the difficulty of accurately determining a network's fitness. Even after playing 50 games and averaging the fitness, the genetic algorithms tended to produce fitness scores that could fluctuate wildly. A given network could score a fitness of 100 one generation and then score a fitness of 60 the next. This certainly would have interfered with our algorithms, which depended on their ability to pick the networks with the highest fitness in each generation. We are unsure how exactly to deal with this problem. We have tried increasing the number of games played to 100, but this slows down the algorithm significantly and does not appear to greatly improve the accuracy of fitness levels. It is possible that the fitness scoring system is inherently flawed. Before further study is done, we ought to examine this system and determine whether there is a superior alternative.

There have been several successful studies demonstrating the potential of neural nets and evolutionary algorithms in the area of game-playing. Our project adds to this body of research by demonstrating how the adaptability of neural networks can be used. While most games can only be played on board of a fixed size, three-dimensional tic-tac-toe can be played on a board of any size. We worked mainly with 3 x 3 x 3 boards and 4 x 4 x 4 boards because these are the only sizes for which an AI can reasonably be constructed. Our genetic algorithms were able to deal with both these sizes easily, and they could have adapted to a significantly larger board. They truly followed a "human-like" approach to learning. In continuing on the lines of this research, it is worth see how marker based encoding can be applied to other problems. This technique may have great potential since it does not require the knowledge of the appropriate network structure for the problem being solved, yet produces very good results.

REFERENCES

1. Chellapilla, K. and Fogel, D.B. (1999). A Co-Evolutionary Approach to Playing Checkers Using Only Win, Lose, or Draw. *Symposium on Computational Intelligence II*, Aerosense99.
2. Fogel, D. B. (1993). Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-tac-toe. *IEEE International Conference on Neural Networks*, 2:875-880.
3. Fogel, D. B. (2000). *Evolutionary computation: Toward a new philosophy of machine intelligence.* 2nd Edition. New York, NY: IEEE Press.
4. Moriarty, D. E. & Miikkulainen, R. (1996). Efficient Reinforcement Learning through Symbiotic Evolution. *Machine Learning*, 22:11-32.
5. Moriarty, D. E. & Miikkulainen, R. (1995). Discovering Complex Othello Strategies through Evolutionary Neural Networks. *Connection Science*, 7(3):195-209.
6. Patashnik, O. (1980). Qubic: 4x4x4 Tic-Tac-Toe. *Mathematics Magazine*, 53(4):202-216.
7. Russell, S. & Norvig, P. (1995). *Artificial intelligence: a modern approach.* Upper Saddle River, New Jersey: Prentice-Hall.
8. deGroot, A. D. (1965). Thought and choice in chess. The Hague: Mouton.

Neural Nets for Sissies

Matthew R. Johnson[1] and Alexander M. Kass[2]
Yale University, Department of Cognitive Science
New Haven, CT 06520

Abstract
In this project we attempt to generate a realistic artificially intelligent network through the use of a genetic algorithm that has direct application to a game from current popular culture. Sissyfight 2000[3] is a multiplayer, turn-based, non-zero-sum game played in the world of cyberspace through the use of the Shockwave™ browser plug-in. Here, we attempt to evolve a net which, given the state of a four-player Sissyfight game, returns the move with the greatest chance of resulting in eventual victory.
Keywords - neural networks, genetic algorithms, online gaming, artificial intelligence, Sissyfight, self-esteem, evolutionary programming, live-and-let-live strategies

1. INTRODUCTION

The popularity of computerized gaming is increasing with every passing year. Nearly every game that is created requires some sort of artificial intelligence (AI) to control computer players in such a way that they will remain competitive with their human opponents, thus keeping the interest of the latter engaged. This kind of AI has been successfully implemented in the past through the use of neural networks which are intended to mimic the behavior of a human player and thus make computerized adversaries both interesting and defeatable (Baba et al., 1996).

Frequently, these algorithms are either simply programmed models of human heuristics for playing a game or brute-force attacks which attempt to map out complete sets of moves and evaluate them using some sort of numerical system devised by the programmers. Neither of these designs, however, is truly able to escape the influence of the programmer and his own conception of the optimal methods for winning the game; they require him to embody his own strategy in the algorithm followed by computerized players. However, as Chellapilla and Fogel (1999) have shown, it is possible to evolve a

[1] matthew.r.johnson@yale.edu, AIM: MattTheGr8, Sissyfight moniker: Bobette
[2] alexander.kass@yale.edu, AIM: Kasskicked, Sissyfight moniker: DJ Cannola
[3] http://www.sissyfight.com

neural network which plays at a near-expert level in some games (their example used the game of checkers) without programming in any additional expert knowledge of gaming strategies. We hope, then, that creating a program which evolves in such a fashion will result in a player robust enough to compete successfully against a number of different strategies without being subject to the pitfalls of a particular heuristic method of play; evidence for the strength of genetically-evolved neural networks in this regard comes from the famous example of the Iterated Prisoner's Dilemma (IPD). In the original programming competition designed by Robert Axelrod, the most successful algorithm was a simple one named Tit-for-Tat, which won two consecutive contests (Axelrod, 1984). However, it has been shown that neural networks evolved with no prior knowledge of the known optimal strategies for solving the IPD, e.g. Tit-for-Tat and its closest competitors, were able to perform on par with Tit-for-Tat itself (Chellapilla and Fogel, 1999).

To this end, we intend to develop a player defined by a feed-forward neural network which is evolved through a model genetic process governed by survival of the fittest. This method is particularly useful for a turn-based game of indefinite length. If one is designing a neural network which takes the state of the game as input and returns the player's next move, it is difficult to use a teaching method such as back-propagation when the ultimate impact of the player's move is not immediately apparent.

To create our player, we had four criteria for the kind of game to simulate. In order to demonstrate the potential applications of our methods, we sought a game currently played by human beings in the real world. To implement the game strategy with a neural network of reasonable size, we sought a game in which each player could be simulated by a neural network that could be given a set of parameters for the current state of the game and then return one decision for the player's next move. To keep the emphasis of the experiment on the players' strategies alone, we sought a game which was independent of random factors but which instead, like classic games such as checkers, chess, and Othello, began each game in the same state and relied only on the players' different strategies to determine the outcome of the game. Finally, to keep the network simple enough that it could be defined by a relatively small number of inputs and outputs, thus keeping computational time down to a reasonable level, we sought a game with a reasonably small set of parameters. One game that fulfilled these four factors was Sissyfight, an online game designed to use the Macromedia Shockwave plug-in which is commonly used with conventional web browsers. The game is also somewhat interesting in a vein similar to that of the IPD; in fact, some of the game's moves are quite reminiscent of the IPD's cooperate-or-defect alternatives. However, it also includes a variety of moves which do not correspond directly to outright cooperation or defection; this kind of inclusion, when tested by networks playing extended versions of the IPD, has caused less mutual cooperation and the generation of more sophisticated and subtle strategies (Darwen and Yao, 2001). Simply by virtue of being a multiplayer game, Sissyfight also

incorporates some of the complexity of the N-person IPD, which is a variant of the IPD containing more than a mere two players (Darwen and Yao, 1995a).

Section 2 sets forth the basic rules of Sissyfight and the version of the game we shall be using in this simulation. Section 3 presents the implementation of the neural networks and controlling shell program that will play the game and evolve the players, respectively. Section 4 contains data indicating our success at creating evolved networks that win games with increasing frequency, and Section 5 details the relevance of these results to the gaming industry and possible further steps.

2. SISSYFIGHT

2.1 Official Sissyfight mission statement

According to the front page of the Sissyfight website, the essential purpose and definition of Sissyfight is as follows:

> "SiSSYFiGHT 2000 is, like, an intense war between a bunch of girls who are all out to ruin each other's popularity and self-esteem. The object is to physically attack and majorly dis your enemies until they are totally mortified beyond belief. You'll never come out on top without making the right friends, so be careful who you're nice to. Because in the end, only the shrewdest will survive with their social status intact!" (http://www.sissyfight.com, 2001)

2.2 Elucidation of official Sissyfight mission statement

Sissyfight, as played on the Internet by individual human players who connect to the site, takes place on a virtual playground between three to six players at a time[4]. The players are simulated schoolgirls whose goal is to reduce the self-esteem points of all of the other players to zero. Each player begins with ten self-esteem points and has the opportunity to raise or lower her and her opponents' points through the moves she makes in each turn. A single turn consists of each player choosing a move; when all players have selected a move for the turn, the moves are executed simultaneously, and their results – which can vary depending on the interactions of different characters' moves – are displayed onscreen; for the next turn, players' self-esteem points are adjusted.

When a player's self-esteem point total reaches zero, she is forced to drop out of the game. Thus, in a four-player game, the demise of one player results in a smaller three-person game which continues until another player is eliminated. Once only two players remain in the game (or, if multiple players are eliminated on the same turn, one or zero

[4] For the purposes of reducing computational time, our network is only designed to work with a game of four or fewer players; this should not be considered a large limitation since a four-player game is perfectly valid within the Sissyfight universe. Furthermore, our network design simply designates four input neurons for each opponent and does not otherwise distinguish between them, so the extension of the experiment to simulate a six-person game would only require the addition of a few more input and output neurons in the same basic pattern of the four-person model, and thus we should not expect a signicant difference in successful strategies for dealing with five opponents versus those dealing with three.

players could be all that are left standing), the match is over; any players with self-esteem points > 0 are declared winners. A game with no winners is thus possible although not

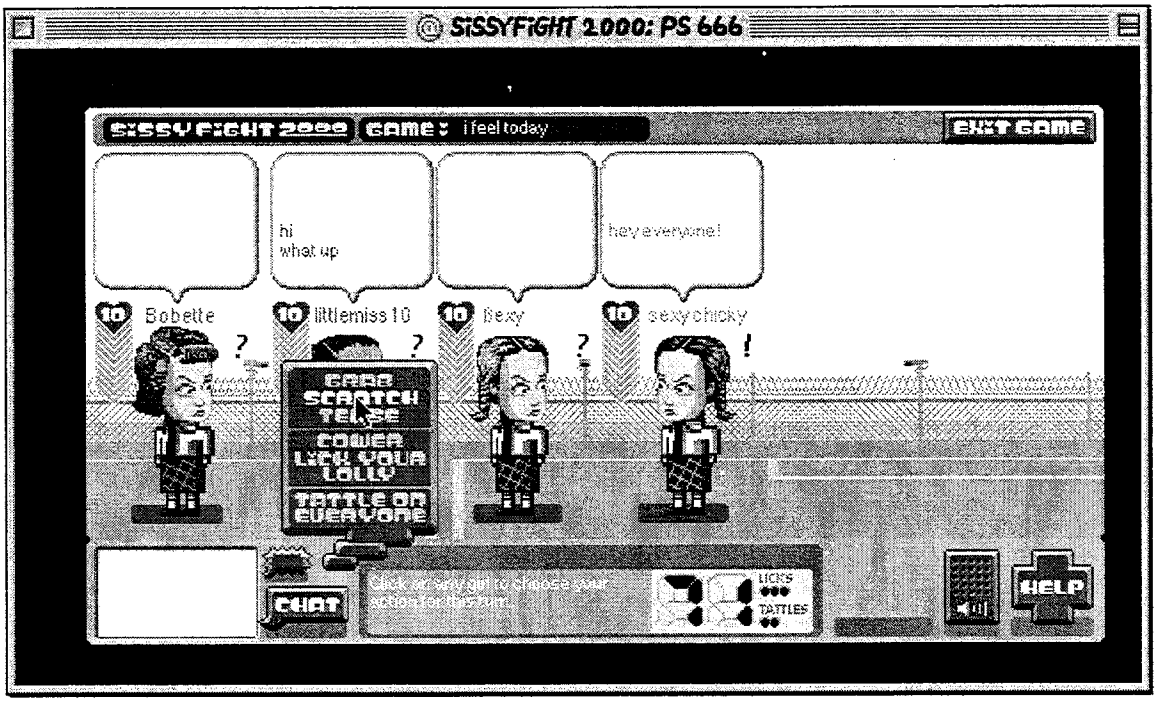


Figure 1. A screen shot from a Sissyfight game in progress; the menu displays all available moves.

very common. The game as played on the website allows collaboration between players through the use of onscreen chat which nonetheless remains visible to all players (secret communication via phone, ICQ, AIM, etc., is strictly prohibited); clearly, though, this level of sophistication is outside the scope of our mission to design an artificially intelligent neural net to succeed on a purely algorithmic basis. Beyond this, any human or artificial strategy, however good, is quickly and easily defeated if it happens to be fighting against three players who already knew each other prior to the game and had agreed to simply cooperate and destroy the first unknown player to join them. In this sense, our networks' competitions are like those between newcomers to the Sissyfight world who have yet to form lasting relationships with other Sissyfighters and thus must contend through strategy alone against an entire battery of players whose methods are unknown. We can liken this sort of situation, rather than to a chess tournament, professional baseball game, or match between frequent Sissyfighters where the players form a society and are familiar with one another's playing styles, to a competition like the television show *Survivor*, wherein contestants begin competition with no knowledge of each other's personalities and must begin formulating strategies and alliances before they really know the people against whom they are playing.

2.3 Moves and rules of Sissyfight

Aside from the general outline provided above, there are obviously quite a few specific rules of Sissyfight that are essential to understanding the game. These rules mostly concern the evaluations of moves on each turn. The basic moves of the game are as follows:

2.3.1 Grab

When one player *grabs* another, the player who is *grabbed* is prevented from making any other offensive or defensive move (see below) during that turn. *Grabbing* in itself, however, does not affect either player's self-esteem points. The only exception is in the case of two different players both executing a *grab* on the same victim; in this circumstance, the victim loses one self-esteem point. Thus, *grab* might be considered an intermediate step between cooperation and defection, a strategy which is intended to defend one from harm while attempting not to provoke additional attack.

2.3.2 Scratch

The *scratch* maneuver is the simplest attack; when one player *scratches* another, the victim loses one self-esteem point. If the victim is already being *grabbed* by another player, she then loses two self-esteem points for every *scratch*. Thus, *scratch* is a direct attack which might inspire retaliation; yet though it is more offensive than *grabbing*, it is still a relatively weak maneuver which may not factor heavily into other players' considerations of whether to hold a grudge.

2.3.3 Tease

A *tease* move by itself inflicts no damage on the intended target player; however, if two or more players collaborate to *tease* a single girl, then the victim loses two self-esteem points for each girl that *teases* her (thus, a successful *tease* attack results in a loss of at least four self-esteem points). Thus, this move, when used properly, simulates cooperation of two players in simultaneous defection against another.

2.3.4 Cower

Cowering is the basic defensive move; although it prevents the player from attacking anyone else during a turn, it also defends against a single physical attack such as *grab* or *scratch*. However, if multiple players attack a victim who is *cowering*, all but one of the attacks can still penetrate the *cower* defense. *Cowering* does not defend against *teasing*. In addition, if a player *cowers* for two consecutive turns without good reason on either turn (i.e., if she is not attacked by any other player during those turns), she will lose one self-esteem point. *Cowering* is a maneuver which, though useful in self-defense, neither lends itself to provoking nor allying oneself with other players.

2.3.5 Lick your lolly

Licking your lolly is the only method by which a player can recover self-esteem points. Ordinarily, if a player *licks her lolly* during a turn, she will recover two lost self-esteem points; however, her total number of self-esteem points can never exceed ten. Each player is limited to three *lolly licks* per game. However, if a player is *licking her lolly* and is *scratched* or *grabbed* by one or more other players, she will choke on the lollypop and suffer two self-esteem points of damage from each assailant. This, too, is primarily a defensive move largely unrelated to one's relationships with her opponents.

2.3.6 Tattle on everyone

Tattling is the game's only method of attacking more than one player at a time. When one player *tattles*, any player who is not *cowering* or *licking her lolly* during that move loses three self-esteem points; in addition, any player sustaining damage from attacks instigated during a *lolly lick* is implicated in the conflict and therefore loses an additional three points due to the *tattling*. *Tattles* are limited to two per player per game. However, if two or more players attempt to *tattle* in the same turn, they will each lose three points without damaging the self-esteem of the other, non-*tattling* players. *Tattle*, then, bears a striking similarity to the basic cooperate/defect choice in the IPD, since a player *tattling* (defecting) alone is highly benefitted by his opponents' loss of points, whereas two players *tattling* simultaneously are both punished.

2.3.7 Extended ramifications of the above

Clearly, the rules above are only a sketch of how moves are evaluated in the actual game; in complex circumstances involving multiple simultanseous grabs and so forth, the outcome of the moves cannot be fully described by the summary above. Although the minutiae of move evaluation are insignificant to discussion of the problem, they are implemented in the real Sissyfight game as well as our network simulation, described below.

3. IMPLEMENTATION

3.1 Outline

Our implementation was programmed entirely in ANSI C and run on an original Macintosh iMac/233 and a Macintosh G3/450. It consists of two main parts: the controlling shell and the neural network players. These parts are described in detail below.

3.2 The controlling shell

The controlling shell essentially plays the same role as the Shockwave plug-in does for human players Sissyfighting over the Internet; it takes moves from the players as input and returns a new state of the game back to the players as its output. However,

unlike the Internet game, our shell has the additional job of generating the players, selecting which ones are allowed to survive at the end of a given round, and mutating them so as to perpetuate the process of evolution.

In the beginning, the shell generates 200 neural network players as described below and sets all of their synaptic weights and biases to random values. The shell then conducts 2500 Sissyfight games per generation, each with four players; the combinations are selected randomly but are distributed in such a way that each player participates in exactly 50 Sissyfight games in that generation. When all the games are completed, the shell preserves the 50 players with the best winning records and discards the rest. The next generation takes as its first 50 players these winning networks without alteration. Another 50 players are mutated versions of these 50, with the mutations being defined as a random value added to or subtracted from each of the synaptic weights and biases; this number varies inversely with the winning rate of the parent network. As an example, a parent neuron has a victory rate each generation defined as:

$$v = \text{(games won) / (games played)}$$

and then the mutation rate of the neuron is defined as:

$$m = (1 - v) / 4$$

such that for each synaptic weight in the network,

$$\Delta w_{ij} = (2 * \text{rand}() - 1) / m$$

where rand() is the standard function returning a random decimal value between 0 and 1, and thus

$$w_{ij}(n) = \begin{cases} w_{ij}(n-1) + \Delta w_{ij}, & \text{where } -1 \le w_{ij}(n-1) + \Delta w_{ij} \le 1 \\ -1, & \text{where } w_{ij}(n-1) + \Delta w_{ij} \le -1 \\ 1, & \text{where } 1 \le w_{ij}(n-1) + \Delta w_{ij} \end{cases}$$

Thus a network whose v in the previous generation was 1.00 will not mutate at all (mutation rate of 0) and will produce an offspring identical to itself; a parent network with a v of only .6 (m=.10), however, will allow mutation of between -0.10 and 0.10 in each of its synaptic weights ($-.10 \le \Delta w_{ij} \le .10$). The final 100 players of each generation are simply more networks whose synaptic weights are generated completely randomly; this is done in order to make sure that the winning neurons have competed against a wide variety of opponents' strategic styles and to make sure that the gene pool of the population stays sufficiently varied. This also allows for the emergence of novelty. Without these outsiders

continually entering the realm of competition, genetically evolved networks tend to compete well against each other but fail when novel strategies are encountered (Darwen and Yao, 1995b).

3.3 The neural network players

Each of our neural network Sissyfight players was defined as follows: The input layers consisted of 16 input neurons with values all normalized so that they ranged from 0 to 1; these values corresponded to the differing values of the 16 different variables used to describe the state of a Sissyfight game at any given time. The network then had a hidden layer of 30 neurons to ensure that its strategy could be sufficiently complex; these all fed into an output layer consisting of 12 neurons corresponding to each of the 12 moves that a player could make at any given point. The move which was returned to the

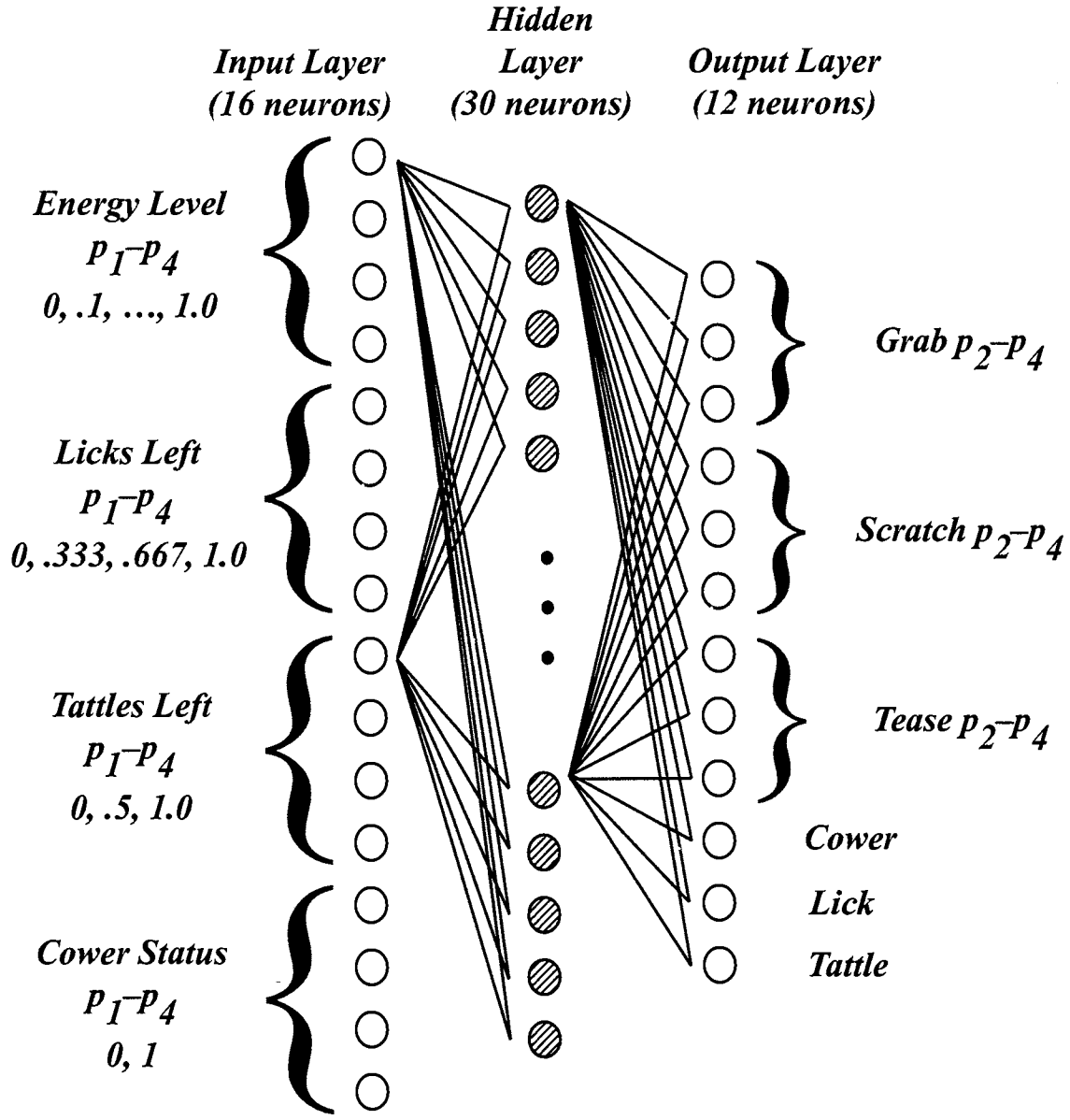


Figure 2. A diagram of one of our neural network Sissyfight players.

controlling shell as the player's move for that turn was selected on a winner-take-all basis among the 12 output neurons; i.e., the output neuron with the highest value was selected as providing the move for that turn. In cases where a network returned a move that was illegal at that time (e.g., attempting to *scratch* a player that had already been eliminated from the game), the next-highest value neuron was selected until an acceptable move was found.

4. RESULTS

Generally, our neural nets evolved to a certain point of game-winning success in a certain number of generations and then exhibited little dramatic variation for the remaining generations. This plateau effect occurred after relatively few generations, as the average number of wins for the 100 evolved nets (fifty winners copied directly from the previous generation and fifty of their mutant offspring) in each generation steadily increased for approximately the first ten generations. After these initial generations, the win count of the evolved group of nets stayed essentially even-keel, hovering near 27 wins per generational tournament, a 54% win ratio (see Figure 3). We then calculated for each generation the difference between the average win count for the evolved group of nets and the average win count for the other 100 nets which were randomly created in each new generation. This difference followed a pattern somewhat similar to the win count of the evolved nets, initially increasing rather sharply for a few generations and then leveling off substantially in later generations, ultimately showing a true separation of approximately 7 wins' difference between the evolved nets and their randomized, newly-created counterparts (see Figure 4). Such evidence showcases the successful learning process of the group of 100 evolved nets and duly supports the notion of how the genetic algorithm

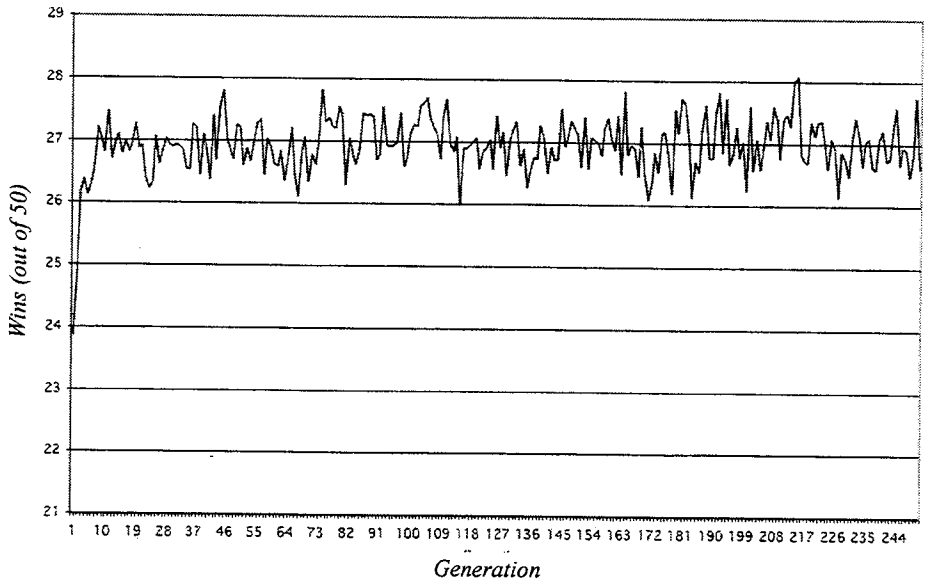


Figure 3. Average wins (out of 50) for evolved nets over 250 generations.

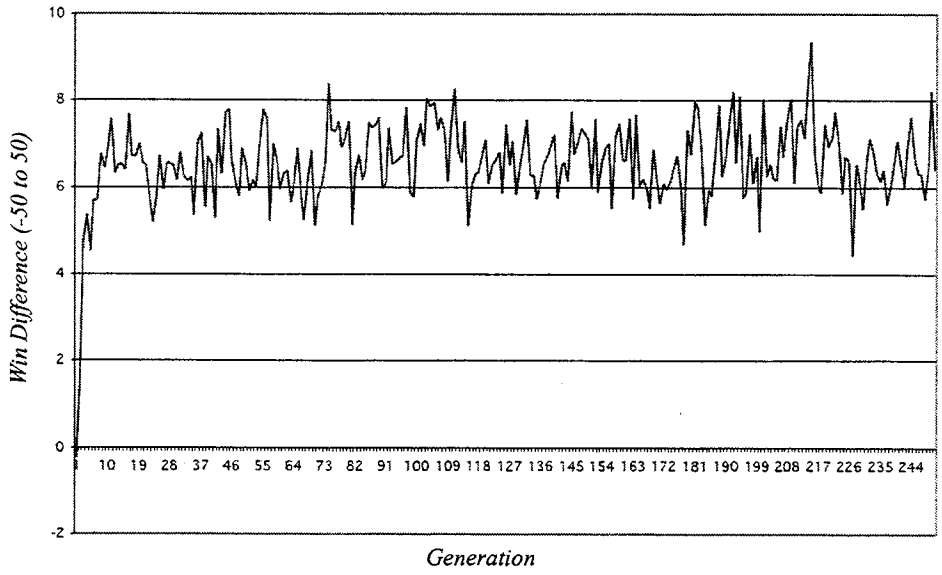


Figure 4. Average win difference for evolved vs. naïve groups.

produced more successful offspring with each generation. Additionally, the most successful network of each generation garnered an average of around 36 wins, ranging from 33 to 42 throughout the 250 tournament runs. Interestingly enough, when one graphs the average performances of these top players over many generations to eliminate some of the tournament-to-tournament variation, these winners' success ratios continue to change and, ultimately, grow with less of a plateau effect than the entire evolved group. Figure 5 shows, for each generation, the win total of the top players from the 20 generations preceding and 20 generations following all averaged together with that generation's own winner. For example, at generation 50, the graph shows the average number of wins for

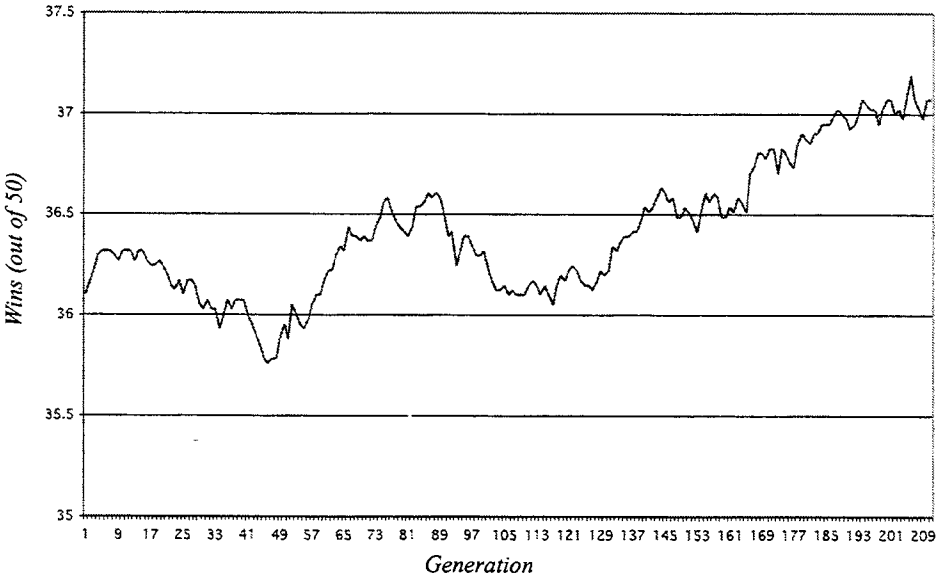


Figure 5. Smoothed graph of tournament champion win totals.

the best players of generations 30 through 70. These tournament champions were almost always drawn from the pool of evolved networks, demonstrating that evolved nets actually do perform above and beyond their naïve opponents.

At an early level in the implementational process, we discovered that certain combinations of players produced effective stalemates, wherein each of the four nets would choose a move that would result in each round of the game giving rise to an identical round, with no changes to the game state that served as the input for the neural network players. Because each computer player would then receive the same input every round, the players would repeat the same combination of moves ad infinitum, thus failing to converge toward an end of the game. In order to eliminate these live-and-let-live circumstances and ensure that each game eventually reached a conclusion, we introduced a rule that would force each of the players to make a random move for one round whenever the shell determined that the game state was no longer changing, and thus some small degree of noise was added to the progression of moves in games that became too static. By recording the number of times this noise addition was needed in the course of many games and many generations, we found that the overall amount of random noise needed to bring games to a conclusion decreased in later generations, with another plateau pattern, this one exhibiting initial sharp dropoff and an eventual period of near-saturation (see Figure 6).

Of course, complete elimination of the need for this randomized correction is impossible, since half of the networks in each generation were newly created with random synaptic weights and were not necessarily good enough players to avoid these live-and-let-live states. Thus, it seems as if the evolved networks took greater responsibility for moving the game along in many of their matches; we could interpret this effect in a few different ways. In one sense, we could say that the playing strategies of the evolved

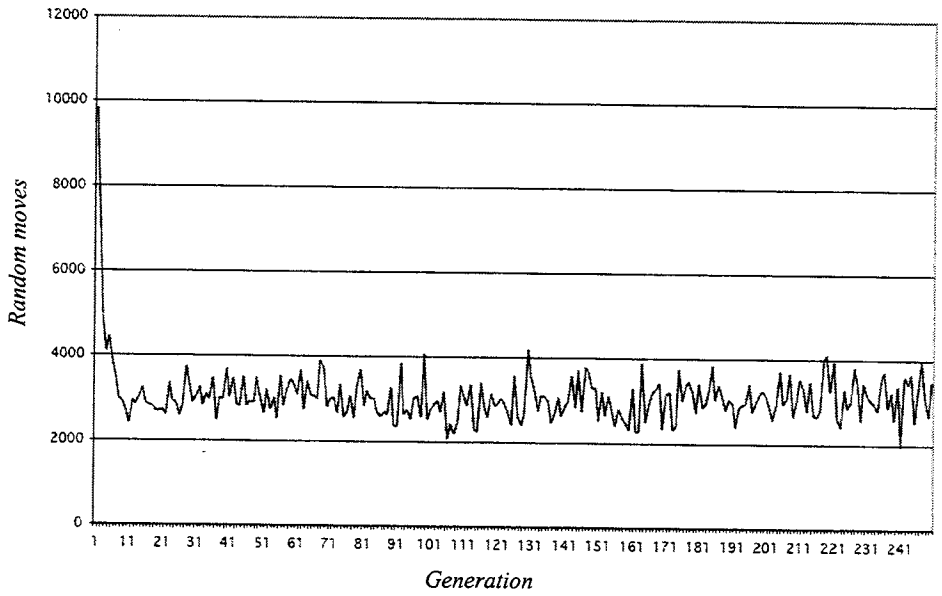


Figure 6. Overall number of random moves required in each generation's tournament.

nets became more aggressive; after all, a static game state is one in which none of the players is effectively attacking any of the others. In such states, a net which attacks its opponents and attempts to make the state of the game more favorable to itself at a time when the others are making more complacent moves or following a live-and-let-live pattern has a distinct advantage over the rest and would be more likely to survive to the next generation. Additionally, we might conjecture that the evolved nets were more sensitive to changes in the game state and developed more subtle strategies to respond to slight variations in their input, since a net that behaved in this way would be less likely to make the same move over and over and thus allow the game state to reach a condition of stasis. In order to investigate this possibility, we would need to make a close study of the exact patterns of moves used by evolved nets and random nets in individual games throughout their evolution.

5. CONCLUSION

Artficial intelligence is an ever-expanding field whose applications are wide-ranging. Eventually, work in AI promises many uses in designing machines with intelligent behavior and creating models of the human intellect, but advances of this type are still far from completion. For now, however, AI still has quite a few practical applications, including the field of computer games. Neural networks may still prove to have great applicability in this particular arena, especially with games that depend more on human decision and judgment than probability and computation. The Iterated Prisoners' Dilemma, mentioned earlier, is a perfect example of this; despite many top scientists' and programmers' efforts to provide complex algorithmic solutions for maximizing success at the game, the best solution for a long time was brief and easily put into simple English: Do whatever your opponent did last time (Axelrod, 1984). Although this strategy required great insight to reach, similar solutions were found through the use of evolved neural networks. Thus, we have attempted to accomplish a similar effect with a game that is far more complex as well as more interesting to the entertainment-oriented gamer, and we have done so with some degree of success.

Through the use of genetic algorithms, we have managed to evolve players that perform consistently better than randomly-generated networks, with the best players in each generation performing consistently at around a 75% winning ratio, with these rates coming from competition against both their fellow evolved networks and naïve ones. Although the direct comparison of these results with human Sissyfight players is not currently possible since online Sissyfight games can vary from 3 to 6 players instead of being fixed at 4, we can still see that even a master Sissyfighter is nowhere near perfect, as compared with a human chessmaster who can easily defeat the vast majority of the chess-playing population. Visual inspection of the All-Time Champs section of Sissyfight.com reveals that even the highest-ranked Sissyfighters of all time have win-

ning percentages that fall mainly between about 55% and 75% (www.sissyfight.com, 2001). Of course, this comparison with our neural networks also does not take into account the advantages human players have in learning their most frequent opponents' strategies, collaborating with other players via onscreen text, and covertly conspiring via other means, also known as cheating.

From the results presented here, though, there are many aspects of the Sissyfight problem left open to further investigation. Our algorithms is clearly capable of generating nets with consistently high win ratios, but the overall win ratio of the evolved group is still significantly below that of their best members. Does the evolved group at large simply need more time to hone its skills to the level of the best performers? Or does competition among the evolved nets prevent more than a few of them from performing exceptionally well, resulting in a limited number of dominant leaders who become the Big Berthas of the neural network Sissyfighting world?

Recent preliminary results may actually point the way towards answers to the above questions; in an effort to lessen the high level of fluctuation between the win ratios of successive evolved generations, we made a small test run of 50 generations using the same program but decreasing the maximum amount of mutation by a factor of 2.5. By setting this less genetically varied group loose into the jungle with the other naïve nets as before, we indeed obtained a much smoother graph of win rates among evolved nets (see figure 7). These win rates also do not level off as sharply as did those of nets with higher mutation rates, and it remains to be determined whether running more generations will result in continued performance improvement. In fact, the average difference between the win totals of the evolved and naïve groups may also have shown some improvement; as with the win rates of the evolved nets alone, the graph levels off much less sharply than before and may continue to increase with further generations. In these 50 generations

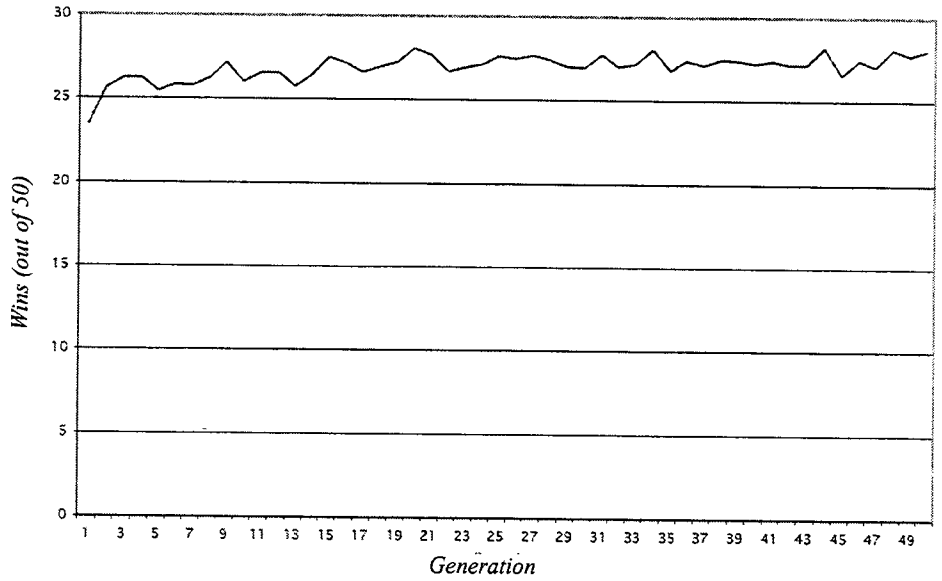


Figure 7. Average win total for evolved groups with a lower mutation rate than before.

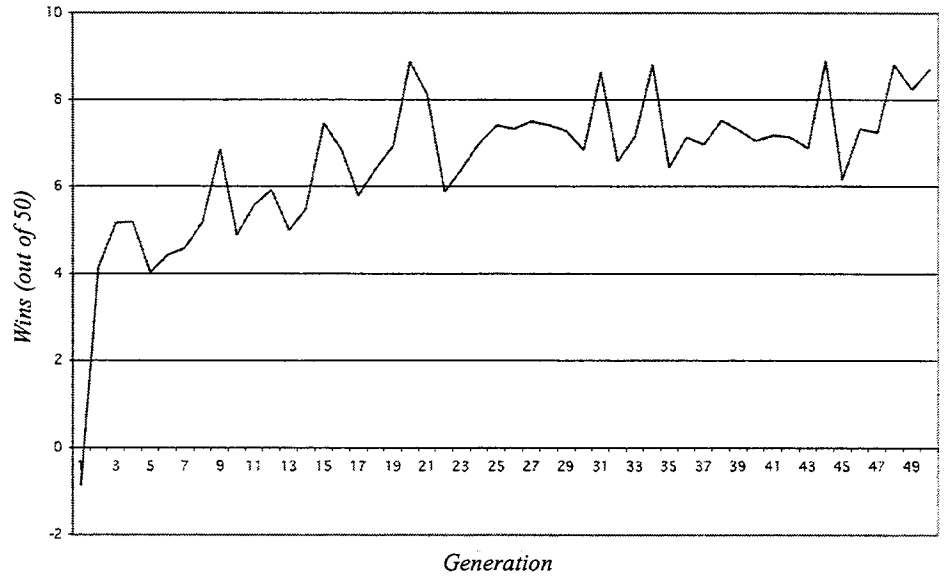


Figure 8. Average win difference for evolved vs. naïve groups with lowered mutation rate.

alone, we can see more frequent peak differences above 8 than in our previous tests with higher mutation rates, but this, too, requires further investigation to determine if it is indeed a continuting trend.

Of course, true analysis of the reasons behind the networks' performance would require inspecting the nuances of computer move patterns in an effort to determine which such patterns might result in better generational success rates. To this end, we could randomly instill patterns of moves into specific nets and subsequently record whether or not such a change has a positive or negative effect on the nets' win-loss records. Doing so would entail subtle programming twists, such as setting a temporally-based parameter that determines if a series of generations has been producing approximately the same win distribution in its evolved nets, and, if so, necessarily introducing the shake-up pattern in an attempt to better equip a handful of nets.

As for our nets' effect of requiring less noise to complete games in later generations, it may be most helpful to look within each game to determine the strategies of the networks which eventually find success. Through careful analysis, we might hope to discover if the more evolved nets' decreased need for random intervention to keep the game moving is actually based upon more aggressive strategies, greater sensitivity to small variations in game state, or other reasons entirely. If prominent patterns are found, then perhaps we could force these patterns upon newly-created random nets and predict that they would also become relatively successful; such a discovery, of course, would allow for smarter and smarter nets to continually compete against themselves, which would lead evolutionarily to certain subgroups of these nets employing more subtle versions of this strategy to overcome the other subgroups, thus taking the evolution of the algorithm one step further.

As with many areas of investigation, research into neural network players of Sissyfight has created as many questions as it has answered. We know now that it is possible to create players of higher playing ability through the use of genetic algorithms, although recent results suggest that we may not have reached the limit of potential improvement. We are also left wondering what the strategies employed by our networks actually are and how these strategies impact upon the need for noise to advance the state of the game. Like our networks, we must revise and refine our techniques based on our findings in order to fully understand their behavior.

REFERENCES

1. Axelrod, R. (1984). *The Evolution of Cooperation*. New York: Basic Books, Inc.
2. Baba, N., Kita, T., Takagawara, Y., Erikawa, Y., and Oda, K. (1996). Computer simulation gaming systems utilizing neural networks and genetic algorithms. *Proceedings of the SPIE – The International Society for Optical Engineering*. Vol. 2760, pp. 495–505.
3. Chellapilla, K., and Fogel, D.B. (1999). Evolution, neural networks, games, and intelligence. Proceedings of the IEEE. Vol. 87, no. 9, pp. 1471–96.
4. Darwen, P.J., and Yao, X. (1995a). An experimental study of N-person Iterated Prisoner's Dilemma games. *Progress in Evolutionary Computation: AI '93 and AI '94 Workshops on Evolutionary Computation*. Berlin: Springer-Verlag.
5. Darwen, P.J., and Yao, X. (1995b). On evolving robust strategies for Iterated Prisoner's Dilemma. *Progress in Evolutionary Computation: AI '93 and AI '94 Workshops on Evolutionary Computation*. Berlin: Springer-Verlag.
6. Darwen, P.J., and Yao, X. (2001). Why more choices cause less cooperation in iterated prisoner's dilemma. *Proceedings of the 2001 Congress on Evolutionary Computation*. Vol. 2, pp. 987–94.
7. http://www.sissyfight.com (2001)

Am I Hot or Not?

Markus Kangas and Fernando Flores
Department of Computer Science, Yale University
New Haven, CT 06520

Abstract
This research project attempts to build a neural net that can capture what makes a person's face beautiful. A feed forward neural net with a back-propagation learning algorithm was used for this. Exemplars from the website www.hotornot.com were used to train the neural net. To test the neural net, several plastic surgery websites containing before and after photos of patients were used, as well as non-exemplar rated images from www.hotornot.com. This software can serve as a useful tool for plastic surgeons, hair stylists, and any other application concerned with physical appearance.
Keywords- Face Detection, Face Extraction, Skin Chromaticity, Image Processing, Image Sampling, Imaging Software, Plastic Surgery, Feed Forward.

1. INTRODUCTION

What makes a person beautiful? Can a neural net be trained to capture it or is beauty really as they say: "In the eye of the beholder"? This is exactly what this research is about; build and train a neural net that given a picture, outputs that person's beautifulness on a scale from 1.0 to 10.0. Then with the use of any third party software[1], a few changes such as face shape or different nose can be made to the picture to get its new rating.

This net will make a very useful tool for plastic surgeons, in aiding the decision of what modifications to make to a persons face. It will be helpful not only in situations were a person's face has been disfigured, either from a birth defect, disease or accident; but also for the more common case were a person would like to improve their attractiveness. The net will also make a good tool for hair stylists, to know what hair cut and hair color fits a person best; make-up consultants and any other profession concerned with physical appearance.

[1] http://www.nausoft.net/eng/products/plastic/index.html

2. EXEMPLARS/IMAGE PROCESSING

2.1 Exemplars for the Neural Net

Photos from the website www.hotornot.com were used as exemplars. This website lets a person post their photo, and anyone who visits the website can rate that person's beauty on a scale from 1.0 to 10.0. Then it tells you the average rating and the number of people that have rated that picture. This was useful for this research, since photos that hadn't been rated by too many people were eliminated.

The first step was to write a script that filtered through each web page from www.hotornot.com and fetched a picture with its rating. Since this website has a cookie based security system this wasn't possible. To solve this problem, 300 distinct images and their rating were saved manually from this website. Whole body pictures were not used as exemplars, since a person's body instead of only face might have biased the rating given to that person.

Before using an image as an exemplar, it has to go through a series of pre-processing steps. This is done to filter noise from the exemplars to improve convergence.

Pre-processing steps:
1. Face extraction from image (see 2.2)
2. Locating of eyes and mouth (see 2.3)
3. Rotation of face (see 2.4)
4. Sampling (see 2.5)

2.2 Face Extraction from an Image:

Skin color retains certain properties regardless of a person's race or living environment. Sampling a large number of people's skin color leads to the conclusion that skin color stays within certain chrominance boundaries. In the commonly used RGB space it is tricky to distinguish skin colors from the background. So the image must be filtered into a different color space to solve this problem.

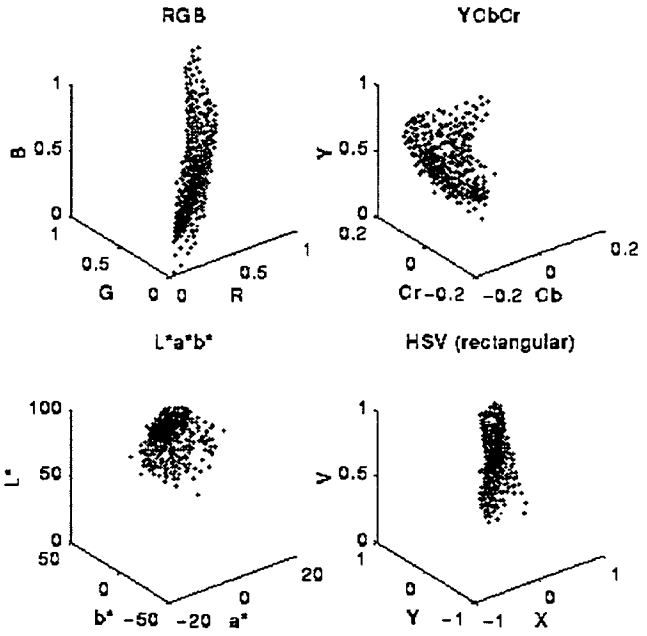


Figure 1. Skin's dispersal in different color spaces
(each point represents an individual's skin color)

There are several different color spaces in which an image can be represented: such as RGB, YCbCr, L*a*b* and HSV. RGB is the most widely used, each pixel in an image is a combination of the intensity of three colors: Red, Green and Blue. Each one of the three elements has an intensity that ranges from 0 to 255. YCbCr measures Luminance, Red Chromaticity, and Blue Chromaticity for each pixel. The other two less common color spaces are L*a*b*, which measures Red-Green Sensation (colors from red to green), Yellow-Blue Sensation (colors from yellow to blue), and Brightness; and HSV, which measures Hue, Saturation, and Value.

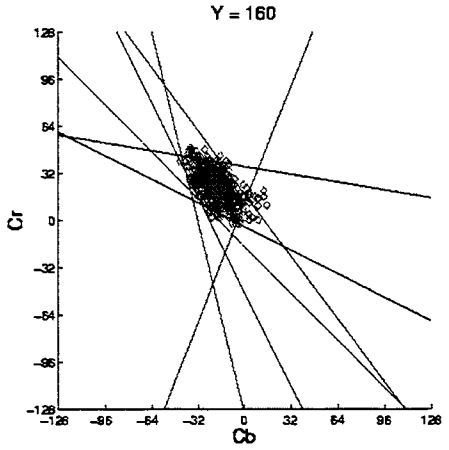


Figure 2. 2-Dimensional Slice of skin chromaticity in YCbCr

It is clear from Figure 1 that in an RGB space, skin color is unbounded, only the R and B elements are confining; in other words, the R and B elements don't vary very much in the sample set of people as shown in Figure 1. So the RGB space is not very useful for extracting the face from an image. After converting the image into the YCbCr space, a clear boundary exists which can be used to locate skin colored pixels within an image (see Figure 2).

To convert a pixel from RGB to YCbCr the following formula is used:

$$\begin{pmatrix} Y \\ C_r \\ C_b \end{pmatrix} = \begin{pmatrix} 0.29900 & 0.58700 & 0.11400 \\ 0.50000 & -0.41869 & -0.0831 \\ -0.16874 & -0.33126 & 0.50000 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

R,G and B are variables that stand for the intensity of each element (0-255).

Figure 3. An image in left)RGB space, right)After skin filtering via YCbCr space using a threshold (0=skin, 255=non-skin).

It can clearly be seen from Figure 3 that in the YCbCr color space the pixels belonging to the face are easily distinguishable from the rest of the image. The skin appears as a dark region as opposed to the rest of the image, which is white.

2.3 Locating of eyes and mouth

Rotating and rescaling of an image is done based on the location of the eyes and mouth. Locating the eyes and mouth within an image can be solved using a technique similar to the one used for face extraction. First the face is converted into the YCbCr space. An analysis of the chrominance components indicates that high Cb and low Cr values are found around the eyes, and that high Cr values are found in the mouth areas.

In our eye detection algorithm, we run a concentric circle filter over the eroded skin map. The eroded skin map is produced by removing lone unwanted pixels through a series of min/max blendings of neighboring regions. Eye regions are the areas which represent tight circular regions in the skin map. False eye regions are filtered out through simple heuristics such as the fact that eyes are relatively horizontal and nearby which removes spurious (invalid) possibilities through deduction. This includes deciding which regions have the largest number of concentric circles surrounding them.

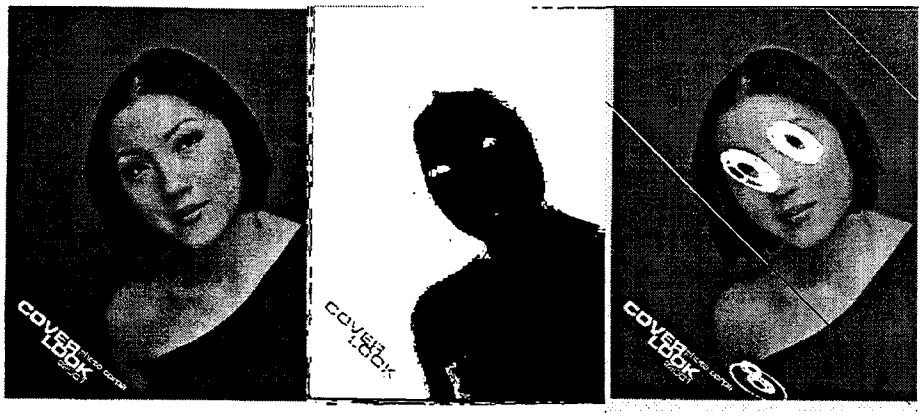


Figure 4. Eye detection, left) original image, center) skin map, right) eye candidates.

The eyes will appear near each other as two separate blobs. If we draw a line from one eye to the other, the middle point between them will represent the top of the nose.

The mouth region contains more red component and smaller blue component than other facial regions. Hence, the chrominance component Cr is greater than the Cb near the mouth areas. We further notice that the mouth has a relatively lower response in the Cr/Cb feature but a high response in Cr^2 (Cr square). Therefore, the difference between Cr^2 and Cr/Cb can emphasize the mouth regions. Adding this heuristic to the eroded skin map generator creates the Eye/Mouth map as shown in Figure 5. Again heuristics are used to filter out spurious false positive, such as removing smaller ring regions that are distanced from the densest ring regions, or removing non-concentric rings that happen to be near each other.

Figure 5. Eye and Mouth Detection, left) Original Image, center) enhanced skin region, right) eye/mouth detection using bounding circles.

2.4 Rotation of face

Once the eyes and mouth have been located, we are ready to take the original image and make a series of transformations. First the distance between the eyes and mouth is calculated using the middle point between the eyes (see above) and the mouth. Then rotate the image either left or right until the eyes are level – the mouth should be directly beneath the eyes as a result, as follows:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Above is the standard matrix for 2-dimensional translation and rotation. Applied to each point in the face map, we can rotate and move the face as necessary. x and y are the coordinates of a pixel P. x' and y' represent P's coordinates after rotation (by angle Theta) and translation (in the x and y directions represented by t_x and t_y).

After rotating the image, it is resized to the desired exemplar image size. This is done by shrinking the image by a factor proportional to the distance between the eyes, so that after resizing all the images have the same distance between the eyes. Then a "box" is cut out which contains the eyes, mouth, and some area above the eyes which includes the forehead and hair. We also include some area below the mouth to get the chin and part of the neck. Rather than just using the detected facial region, we choose this "box"

that contains the whole area immediately surrounding the face. The main reason is to ensure any hair around the face is included in the exemplar.
The size of the box depends on the number of desired inputs for the neural net.

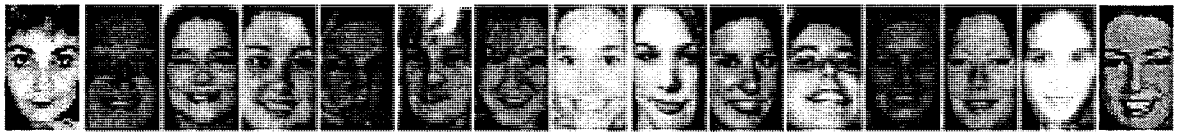
2.5 Sampling

Now we have to decide how we are going to sample the images. For instance we can choose to keep the face in the RGB coordinate system (the way we see things, 3 values used for each pixel), or we can convert it to a grayscale format. Grayscale results in storing one number per pixel as opposed to 3 numbers. Also the width and length of each exemplar in pixels must be specified.

For instance if we take only 64 by 64 pixels in monochrome format we get 64*64*1=4096 inputs. Optionally we could take 32 by 32 pixels in RGB format giving us 32*32*3=3072 inputs. In both cases, each input value is normalized by dividing it by 255, the maximum intensity value in the monochrome and RGB format. After normalization each input value will range between 0 and 1.

Experimentally, our optimal input image is a 40 x 25 grayscale image. This size allows for reasonable convergence times for our chosen database. This also equates to exactly 1000 inputs for the neural net. We also specifically choose a subset of images that vary in ratings from 1.7 to 9.9 on the www.hotornot.com rating scale to serve as exemplars. This wide scale provides the weights in the neural net with more leverage in deciding which features are attractive and which are not.

Here are some of the actual sample images as they appear in our exemplar database (before the grayscale conversion to the luminance field).



3. THE NEURAL NET

3.1 Back-Propagation Network

We will use a feed forward neural net with a back-propagation learning algorithm. It will consist of three layers: one input layer, one hidden layer and one output layer. The input layer consists of 1000 input neurons. The hidden layer will contain 200 neurons, which is

approximately 20% of the number of inputs. The output layer will have only one output, that will be a value between 0.1 and 1. This value will then be multiplied by 10 to get the picture's rating on a scale from 1 to 10.

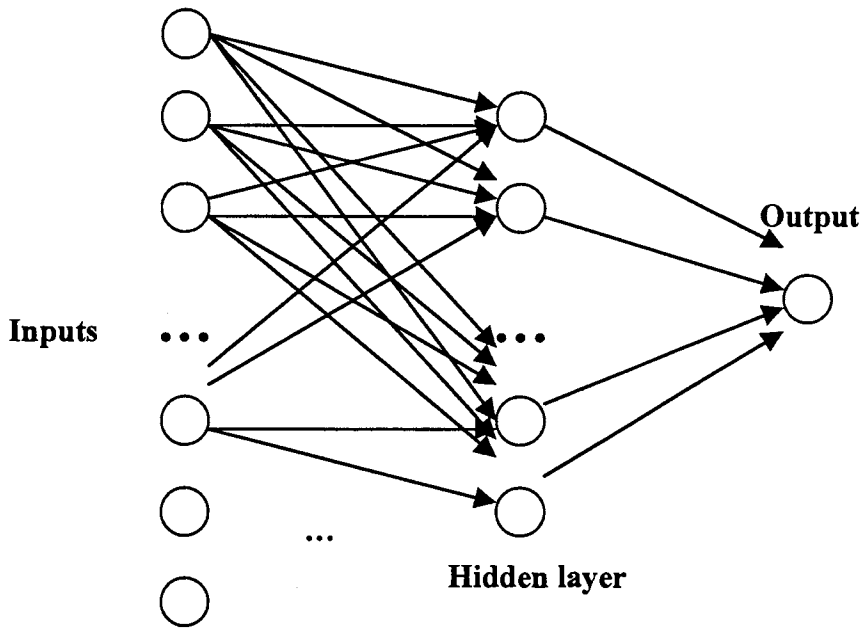


Figure 6: Structure of the Feed Forward Net

The weights of the neural net will be initialized to a random value between –1 and 1. Also each neuron is going to have an extra input with a value of –1. This is to take the bias into account. The weight of the bias will also be initialized to a random value. For the activation function the logistic function will be used. The slope parameter **a** will be 0.5, so the output **y** of each neuron will be:

$$y = \frac{1}{1 + e^{-av}}$$ (where v is the sum of the weighted inputs)

3.2 Training of the Neural Net

The images obtained from the website (which have already been processed) will then be used as inputs for the net and the weights will be updated using the back-propagation algorithm. Since there are 3 layers, the variable **k** in the formula below varies from 0 to 2. Also the learning rate η will be 0.1.

Change in weights:
$\Delta W^{k,k-1} = - \eta * \delta^{k} * y^{k-1}$ (η= learning rate, **y**= output of a neuron, **k** denotes the k'th layer)

Gradient for output layer:
$\delta = e * \underline{y}$ where $e = y - d$ (difference between actual output **y** and desired output **d**)

Gradient of other layers:
$\delta^{k} = y^{k} * \delta^{k+1} * W^{k+1,k}$ (W = weight matrix)

3.3 Testing of the Neural Net

To test the neural net, we will use several images from the website www.hotornot.com that weren't already used as exemplars. If the output of the neural net is within a reasonable distance from the expected rating for that picture, the neural net has successfully captured what makes a person beautiful and now it can be used to give a rating for a "never before seen image".

Some of these "never before seen images" will be already used exemplars, but these will have some modifications done to them to get a new rating. Modification will be made using existing commercially available programs. There are various software packages that create different hairstyles, hair color and facial modifications. If we cannot get a copy of these commercial software packages, we will use Photoshop to make the modifications. Also several before and after images will be used from plastic surgery websites. These websites contain before and after images of people who have had several kinds of surgeries such as nose, eyelids and others done to them. (Since the after photo was taken at a later time, things such as hair color might be different) We can keep modifying a picture until we get a rating close to 10, which will mean that the person's attractiveness will have improved by a lot after these modifications.

4. THE APPLICATION

4.1 Current Application

This is a screenshot from our current application, which was developed in Visual C++ 6.0 to run in Windows 2000. This program allows the user to select an image file, run image filters, and then use it to train a neural net for a given rating.

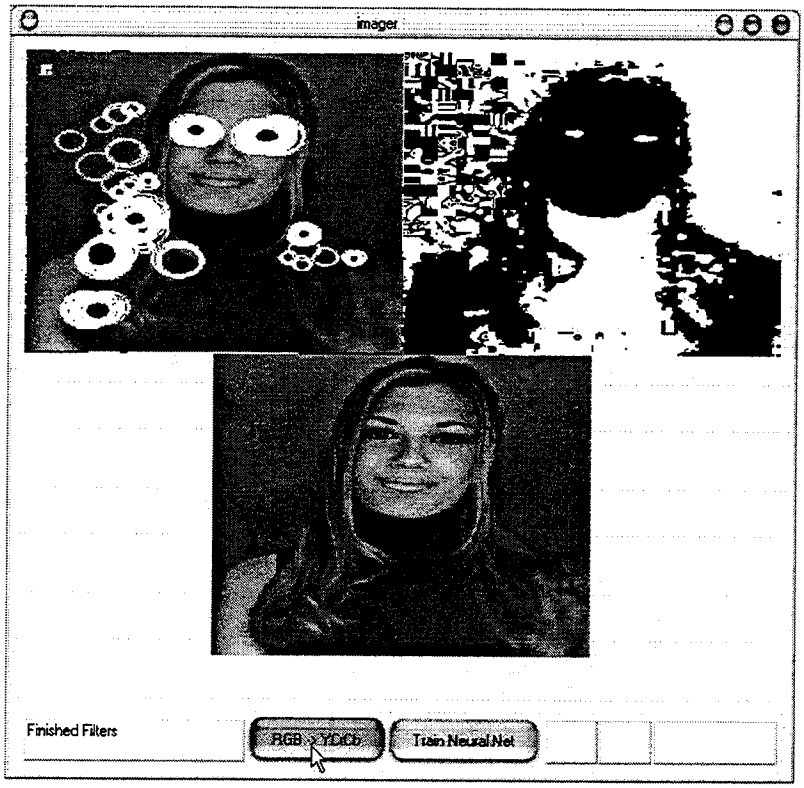


Figure 7. Screenshot of application at work.

Our code consists of several C++ classes including one for the image pre-processing and a separate one for the neural network. This modularity allows us to change pre-processing options and/or the neural net without the need to rebuild the entire application. The lower image shows the raw bitmap before any pre-processing occurs. The top-right image represents the eroded skin-map, and the top-left image represents the detected eye candidates before heuristics that are applied. All images (except for the original source file) are stored and manipulated in RGB-24 format for ease of use.

5. CONCLUSIONS

5.1 Test Setup

Our final neural network ended up with a learning rate of 0.1, **a**=0.5, a threshold of 0.02, and 50 exemplar images. Converging took approximately 7 hours on an AMD Athlon 1.2 GHz machine after 5860 iterations. It seemed unfeasible to add any more exemplars since the time to compute convergence grows exponentially as the number of exemplars increases. In addition a tighter threshold than 0.02 (representing 0.2 points on the rating system), would require exponential more iterations.

The image pre-processing worked for many of the sample images. However, the algorithm was unable to detect eyes on at least one third of our sample set because of poor photo quality, over-exposed lighting conditions, or partially occluded faces. This was an unfortunate feature of the images on the website. Therefore some of our images were pre-processed by hand using Corel Photoshop. A more time-consuming and robust eye detection algorithm based on neural nets would likely have solved this problem. However due to time constraints this was infeasible. However, our eye detection algorithm performed very rapidly and could be applied to many motion video applications.

5.2 Plastic Surgery Results

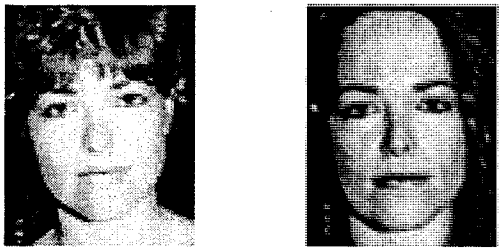


Figure 8. Before(left) and after(right) nose surgery pictures.

These two pictures were downloaded directly from a plastic surgeon's website[2]. The before surgery picture on the left received a rating of 2.1 from the neural net. The after surgery picture on the right received a rating of 6.3 from the neural net. We had similar results with other before and after pictures. Only in one tested case was no increase in rating detected after plastic surgery. The trained neural net seemed to agree with the plastic surgeons' opinions.

5.3 www.hotornot.com Results

We tested the predicting capability of our trained neural net on several rated images from the website. An image which was actually rated by people as 3.5 on the website, our neural net produced an output result of 4.4. For the image which had received a 6.1, the computed rating was 5.1. And finally for the image that received a 9.3 on the website, our neural net produced a result of 6.3. These results provide evidence that the trained neural net did indeed learn some basic attractiveness prediction capabilities. Although the final test was off by 3 entire points, the order of attractiveness among the images was maintained.

---

[2] http://www.facialsurgery.com/PPgbeforeafters_intro.html

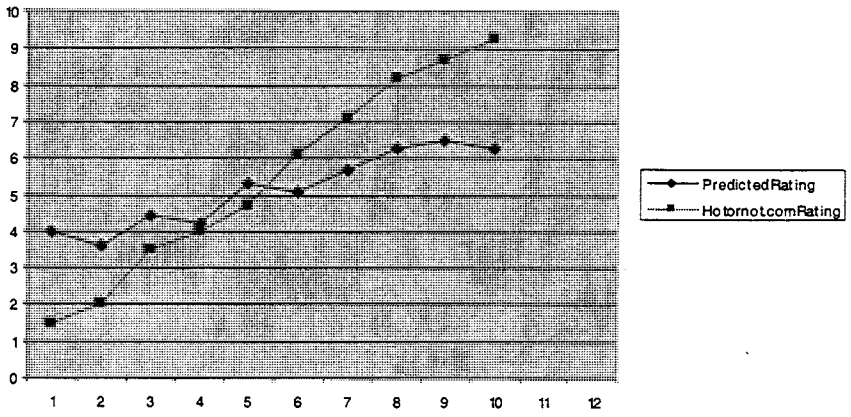


Figure 9. Comparison of Computed Results versus Exemplar Ratings (Hotornot.com Ratings)

5.4 Improving our Results

Clearly a larger exemplar database could have helped produce better results. For instance, if we wished to test a dark-skinned individual there were very few similar exemplars for the net to base its prediction on. In addition, there are a plethora of different face types and facial features that were not part of our exemplar set.

The large variety of different test images also made it difficult for the neural net to learn because of vastly different lighting conditions and other factors as mentioned before. Perhaps a better image normalization filter could help lower this entropy leading to better results.

A larger exemplar image size would certainly have aided in improving the database. Many facial features were in effect removed from the images since they were constrained to such a small sampling area. Unfortunately, the computation time grows exponentially with larger image sizes.

The rating system on the website was also quite arbitrary. Some very similar images had been rated less than the others (i.e. low precision), and others were clearly rated according to something other than the face that appeared in the image. We attempted to alleviate this risk by choosing images that emphasized the facial region. However, there is clearly no absolute rating for any individual. Therefore the sample set is intrinsically flawed.

5.5 Overall Results

Overall, our results were convincing and our results demonstrate that a neural network can indeed be used to rate attractiveness. Plastic surgeons and hair stylists could use this type of software to see how attractive someone is before and after a transformation. However, comparing two very different individuals is always difficult and not always accurate despite how large and ideal the exemplar database is.

5. REFERENCES

Abdel-Mottaleb, Hsu, Jain. *Face Detection in Color Images.* http://www.cse.msu.edu/~hsureinl/facloc/index_facloc.html

Amtoun. *Face Recognition using Back-Propogation.* *http://www.vision.uwindsor.ca/members/amtoun/reports/592/project/*

Cai, Goshtasby, Yu. *Detecting Human Faces in Color Images.* http://www.cs.wright.edu/people/faculty/agoshtas/facechroma.html

Goodridge. *Multimedia Sensor Fusion for Intelligent Camera Control and Human-Computer Interaction.* http://www.ie.ncsu.edu/kay/msf/abstract.htm

Zhuo. *Real-Time People Detection.* http://www.sccs.swarthmore.edu/users/01/xianglan/e27/realtime.html

Neurogenesis Can Help Reduce Noise

F. Anthony Lazenka and Dimitrios Panagoulias
Yale University, Dept. of Computer Science
New Haven, CT 06520

Abstract

An associative memory consisting of a neural network was built on which to test the effects of neurogenesis on learning and recall. Recognizable input patterns and noisy patterns were trained on the network. It was found that the network could recall the initial input with less distortion after neurogenesis followed by retraining than by retraining alone.

Keywords - neurogenesis, associative memory, neural networks

1. INTRODUCTION

It has only been within the past five years that neurogenesis has become a surveyable neurophysiological phenomenon. Neurogenesis in rats, mice, monkeys and humans occurs in the dentate gyrus of the hippocampal formation, allowing the birth of neuronal stem cells which may differentiate into either glial supporting cells, or into actual neurons. These neurons achieve dendritic arborization and axonal synaptic connections capable of neurotransmission [Chambers, 2002]. Neurogenesis has thus been classified as a "turboplastic" mechanism -- a short-term change in the brain that causes long-term effects. To date, investigations of neurogenesis have been limited to post-mortem studies, and have not yet established a biological significance of stem cell birth [Eriksson,

1998]. This paper proposes a neural network model to study mechanisms and effects of neurogenesis.

Neural network simulations can provide simplified models of complex neurophysiological events. These can then be formally studied due to their mathematical basis. A recent model for schizophrenia proposed by Hoffman [1997] used a four-layer neural network to examine the effects of cell death and synaptic elimination on linguistic processing.

Specifically, phonetic patterns were input to a network, fed through a hidden layer, and output in the form of patterns classifiable as syntactic and semantic representations of English words. A temporary storage layer was connected to a hidden layer to simulate cognitive mechanisms of working memory. The network was then trained on a fixed set of phonetic representations. In the trained network, Hoffman found that if he reduced the number of synaptic connections between the hidden layer and the temporary storage layer, accuracy of word recall increased to a point, and then developed into "hallucinations" -- defined as spurious outputs of the network when input was a zero-vector. Both of these results are consistent with psychological studies of synaptic pruning. Moreover, "hallucinations" were non-random and were evoked only by pruning connections between the temporary storage layer and hidden layer, supporting recent psychological evidence that working memory plays a role in schizophrenia.

Hoffman examined several plausible mechanisms of synaptic pruning and cell death in his study, including Darwinian competition and excitotoxicity, respectively. We propose to model corresponding hippocampal mechanisms of neurogenesis and turboplasticity, including cell birth and death mechanisms. The intent will be to demonstrate, in a more general way than Hoffman, the impact of neuronal loss or birth on learning and recall in a neural network.

2. PROCEDURE

An auto-associative memory, defined as a memory which retrieves a stored stimulus when cued with a similar stimulus [Haykin, 1999], will be implemented on a fully connected, single-layer neural network. The network will initially consist of a collection of zero-valued synaptic weights, which will be masked with a random pattern. A set of 8x8 black and white character patterns (Fig 1), with values either 1 or -1, will then be used to train the network employing Hebb's rule of learning:

$$\Delta w = \eta yx - \alpha wy$$

The right-hand term contains a "forgetting factor" to prevent saturation of the network. The factor will be chosen based on empirical measures of the maximum weight expected to result from the training of any single pattern.

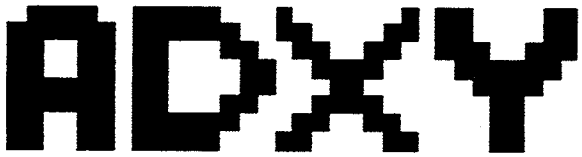


Fig 1: The patterns corresponding to letters A, X, D, Y. Dark intensities have value 1, light intensities value -1.

To simulate random turboplastic events, the network will be trained on a set of random noise patterns in a manner similar to that of the input patterns; however the training using noise will be for iterations several times longer than the training using input patterns. This should have an adverse effect on recall of the network. To counter these effects, two methods will be tried and compared: 1) Implementation of neurogenesis (defined as the resetting of targeted neurons' weight-values), followed by retraining with the input; and 2) retraining without neurogenesis. Specifically, neurogenesis will be implemented on the quarter of the neurons in the network with the greatest sum of input weights. Physiologically this is consistent with excitotoxic effects along the synapse. New weights will then be set based on the weight average of neighboring synapses, corresponding to neighboring neurons in the input pattern. This is consistent with effects of "synaptic leaking", a phenomenon recently observed in hippocampal tissue [Vogt, 1999].

2a. COMMENTS ON NEUROGENESIS

It is plausible that the methods of neurogenesis described above will grant indirect benefits to the network. Excitotoxic elimination will target those weights that noise will affect the most, namely weights which carry the most information from the input. This will remove the noise effects that are concentrated in these synapses, and allow cleaner retraining. Averaging the weight values of neighboring neurons to recalculate the weights of a targeted neuron will also tend to "average-out" the effects of noise.

Effects of synaptic leaking are relevant to character recall since, in intensity bitmaps of characters, pixels tend to cluster alongside pixels of similar intensity according to the form of the character (Fig 2). Thus, in the network representation of the letter "E", a neuron adjacent to another neuron representing pixels in the top horizontal line would likely be connected along weights of similar value.

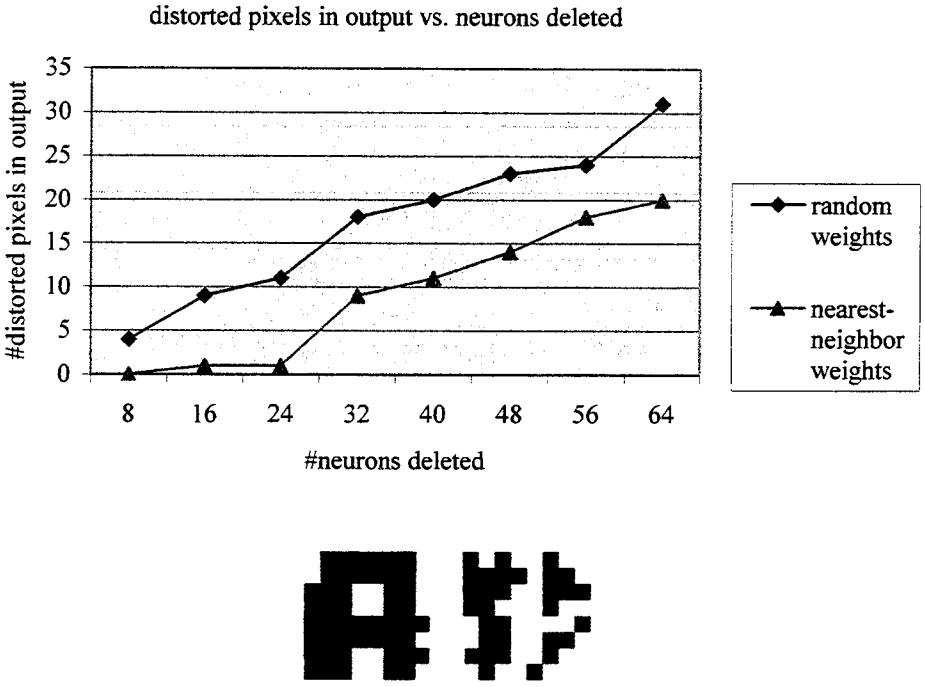


Fig 2: Performance after training followed by neurogenesis when setting weights to the average of the nearest neighbors is compared to random weight settings. The output of the network on the pattern corresponding to the letter A is shown after weights were reset 48 times according to the averaging rule (left) and to the random rule (set to a value between -1 and 1). Note that the form of the pattern is preserved on the left.

2b. COMMENTS ON THE NETWORK

Hebbian learning, because it is a completely local mechanism, loses simplicity on a network of more than one layer. Larger networks are better suited to a backpropagation algorithm, and in fact a network of more than one layer allows that input patterns do not interfere with each other if they are linearly inseparable. Linear inseparable patterns can be compared to noisy patterns in that both are inputs that have an adverse effect on the network; they can be contrasted in that effects due to noise patterns can be reduced, whereas effects due to linear inseparability cannot in a simple associative model.

Although a fully connected one-layer network is limited in its computability by effects of inseparability, its high degree of storage redundancy makes it robust against effects of noise. In this experiment, the patterns for the letters A, D, X, and Y were found in a test network to be linearly separable.

3. RESULTS

For the experiments in which the network was trained on noise for iterations four- and eight-times that of the input patterns, the network performed better after neurogenesis followed by retraining than by retraining alone (Fig 3). Distortion comparisons of the results of the two networks are shown for the four-times paradigm in Fig 4.

	Network trained on 2x noise #pixels distorted in output		Network trained on 4x noise #pixels distorted in output	
input pattern	*w/o neurogenesis*	*w/ neurogenesis*	*w/o neurogenesis*	*w/ neurogenesis*
X	2	3	8	6
D	3	3	12	10
A	0	0	1	1
Y	3	4	12	10
	15 iterations of retraining until perfect recall	15 iterations of retraining until perfect recall	25 iterations of retraining until perfect recall	20 iterations of retraining until perfect recall

	Network trained on 8x noise #pixels distorted in output	
input pattern	*w/o neurogenesis*	*w/ neurogenesis*
X	13	14
D	18	14
A	8	7
Y	20	17
	55 iterations of retraining until perfect recall	55 iterations* of retraining until perfect recall

Fig 3: Shown numerically, the network trained after neurogenesis, when presented with the original input patterns, had fewer distorted pixels in its output in comparison to the network trained without neurogenesis.

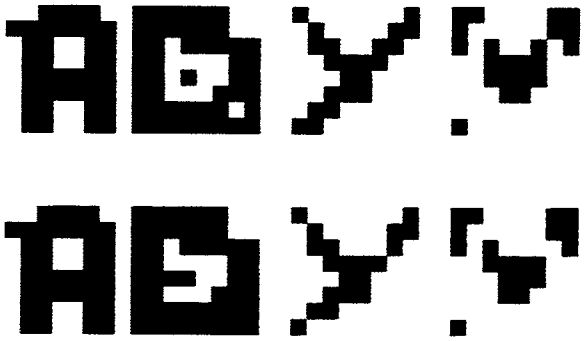


Fig 4: Neurogenesis results for A, D, X, Y are on top, and are less distorted than the results for the network trained without neurogenesis (bottom).

For the experiments in which the networks were trained on noise for iterations two times that of the input patterns, the network with neurogenesis performed no better than the network without. This suggests that neurogenesis targeted on the most-active neurons may have a negative effect on a network presented with a low level of noise. Fig 5 shows the results of retraining the two kinds of neural networks on the four-times noise paradigm after certain numbers of iterations. For all the noise paradigms, it took a similar number of total iterations to accurately retrain both networks. This is consistent with the quick convergence of training that is a property of most associative memories [Hassoun; 1993]. The fact that recall was eventually possible on both networks suggests that the positive effects of neurogenesis weren't related simply to effects of dispelling Hebbian saturation along the synapses. Interestingly, the network to which the most amount of noise was introduced (the network trained on 8x noise in Fig 3) and on which neurogenesis was then implemented could not accurately recall the input pattern corresponding to the letter "X" regardless of more than 100 iterations of retraining. This was most likely a result of saturation, since the distortion in the output pattern was limited to a single pixel.

Progress of retraining, 4x noise

iteration of retraining	#noisy outputs *w/o neurogenesis*	*w/ neurogenesis*
5	X, D, A, Y	X, D, A, Y
10	X, D, Y	X, D, Y
15	D, Y	Y
20	Y	Y
25	Y	None
30	None	None

Fig 5: The letters that could be recalled after certain amounts of retraining are compared for the two types of networks.

4. DISCUSSION

In terms of neurophysiology, the training of noise patterns on a network is analogous to an undesired turboplastic result. In terms of cognitive psychology, this training is perhaps analogous to the learning of undesirable information. It was shown in this paper that neurogenesis can be used to dispel the effects of noise on neural networks, and that recall of input patterns can be measurably better in a network retrained after neurogenesis than in a network retrained straight away. By analogy, neurogenesis may be the brain's mechanism for efficiently relearning information after that information has been degraded in memory. Further research in this area can be directed to more complex networks and to different learning paradigms.

REFERENCES

1. Chambers, "Neural Net Simulation of Hippocampal Neurogenesis [in progress]".

2. Eriksson, et al. "Neurogenesis in the adult human hippocampus". *Nature Medicine*, 4, 11.

3. Hoffman, McGlashan, "Synaptic Elimination, Neurodevelopment, and the Mechanism of Hallucinated 'Voices' in Schizophrenia". *American Journal of Psychiatry*, 154, 12.

4. Vogt, Nicoll. *Proc. Natl. Acad. Sci. USA 96.*

5. Haykin, *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.

6. Hassoun, *Associative Neural Memories: Theory and Implementation.* Oxford Press, 1993.

Distributed Colony Learning

Benjamin Lerner and Benjamin Wallace
Department of Computer Science
Yale University
New Haven, CT 06520

We investigate the interaction of feed-forward networks by defining a colony of simple, dynamically interacting and learning networks. Working within the metaphor of an ant colony, each simple unit of this network is an ant. Each ant will learn by observing other ants, and interfacing with the environment. We conclude that the rate of convergence is coupled to the number of teachers present and the class size.
Keywords: group learning, neural network

1. INTRODUCTION

In real life, if one were to watch a single ant wandering about, feeding, working, resting etc, one would quickly become rather bored. But if one were to watch an entire colony of ants, one would be astounded to find the scale of productivity and diversity of action that the colony displays. Further, one would find that the colony is robust, in that the loss of any single ant's work will have little or no effect on the colony as a whole. Yet the colony performs complex tasks which apparently require global levels of coordination. This takes place despite the severe limitations on each ant, namely that no ant can gauge what the colony as a whole is doing, but rather can only observe its local neighborhood.

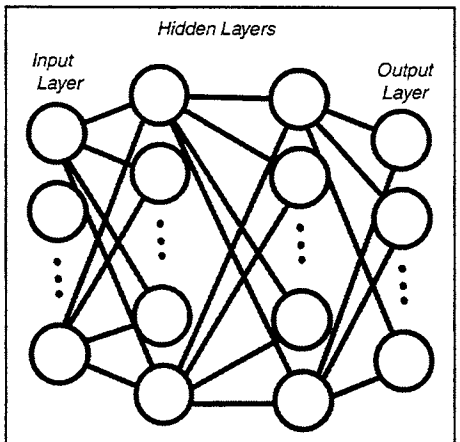
Local information has been shown to be sufficient for solving some classes of complex problems in computer vision, and control tasks. In many of these solutions, the task is broken apart and a small section of the input data is passed to interconnected nodes. Each node examines its piece of the data and its neighbors' actions are doing, then updates its own state by some simple preprogrammed algorithm. In this case, the algorithm that processes the data is predefined – most likely determined in some fashion by the genetic makeup of the animal. But what would happen if that algorithm were not preprogrammed, but rather learned? Consider the case of a "newborn" ant. How does a newborn ant learn how to process local data and thus assimilate into the complex colony? Could imitation of other ants in the environment be sufficient to teach the new ant? We construct an experiment based on this principle, testing whether a collection of untrained ants can learn a complex behavior based on imitation of other ants in its neighborhood.

2. PRELIMINARY EXPERIMENTS

2.1 Local information and convergence: Definition

We start by showing that for a simple case using local information, a collection of networks can converge to isomorphic networks in weight space. In other words, given the same input vector, each network will produce the same output. Further, if we include a "teacher" in the collection, the entire collection will converge to reproducing the teacher's behavior. This teacher provides constant example data, creating a stable focus for the learning algorithms to follow.

We begin by formalizing the concept of a colony of ants, where each ant responds to local stimulus. A colony is a collection of independent, randomly weighted but structurally identical neural networks, which we will call ants. During simulations, some ants will be continuously learning, while some others have their output patterns fixed (i.e., the network is fully trained and is merely computing output without update). We choose a feed-forward net for each ant.



Schematic diagram of a feed-forward network

We let the gain function be the logistic function, so we can use the Back Propagation algorithm to train each ant. In order for Backprop to work, we need a desired output to compute the error and propagate it backwards through the network. While this would normally come from a predefined set of exemplars, we will define the desired output based on the outputs of other ants in the colony. We will call the set of ants that a given network uses to compute its desired output that network's neighborhood. Now we can define the desired output of a given network to be the average output vector of all the ants in the given network's neighborhood. Averaging over several networks reduces the convergence time by making the desired outputs more resistant to

random variations.

Finally, we define two measures of the convergence of the collection of networks based on the mean squared error. If there are p networks in the collection, and $\vec{y}_j(n)$ is the output vector of network j at time t, then:

$$\vec{A}_i(t) = \frac{1}{p-1}\sum_{\substack{j=1 \\ j\neq i}}^{p} \vec{y}_j(t)$$

$$Diversity_j(t) = \sqrt{\sum_{i \in colony} \left(y_j(t) - A_i(t)\right)^2}$$

The diversity is defined as the root mean square of the output's difference from the average output, and is a useful measure of how far away the networks are from each other. (This measure differs from the standard deviation in that we do not divide by the colony size; it is more useful to see the total error in the colony, not the average error.)

When we do have a teacher in the collection, we measure how far the collection is from learning the teacher's behavior: If the desired (teacher's) output is $\vec{d}$ then:

$$Error(t) = \sqrt{\sum_i \left(d_i(t) - A_i(t)\right)^2}$$

This measure shows how far away the group consensus is from the desired output, and is essentially the diversity measurement as seen by a teacher. This measure will be used from here on, as it is the most useful measure of the convergence of the colony.

2.2 Local information and convergence: Method

For each time step t, we let the input be a random vector of length n with each component an evenly distributed binary variable with expected value 0.5.

$$input = (x_1, x_2, ..., x_n)$$
$$x_i \in \{0, 1\}$$

We defined the output to likewise be a binary vector.

We tested defining a network's neighborhood both statically, as a biconnected graph (for example people sitting around a table where the two people on either side of a person are that person's neighborhood), and dynamically (any randomly selected pair of people at that table).

Dynamically creating the neighborhood has some interesting effects on the rate of convergence of a colony.

As with any updating scheme, one can implement it synchronously or asynchronously. While the asynchronous method is much easier to code, by iterating through each network in turn, the synchronous case is much more representative of a real world scenario, as independent people can "update" whenever they so choose, and not according to any schedule. (This may seem counterintuitive. Synchronicity merely means that each person does not have access to the updates made by other people to inform his own update. Regardless of whether the actions took place at identical times or not, if each person only has his own information available, it is essentially synchronous updating.) To achieve this, we first compute the dynamic neighborhood for each network, and then generate the average output of that neighborhood over all the data in an epoch. (Each epoch is suitably randomized set of each input-output vector pair.)

The Backprop algorithm requires two parameters to function – the learning rate and the momentum constant. Due to numerical instabilities and implementation issues, we left the momentum constant at zero, and compensated by taking a small learning rate of $\eta = 0.3$. To prevent saturation problems, we cut back the desired outputs, from 0 and 1 to 0.005 and 0.995, in accordance with a well-known heuristic.

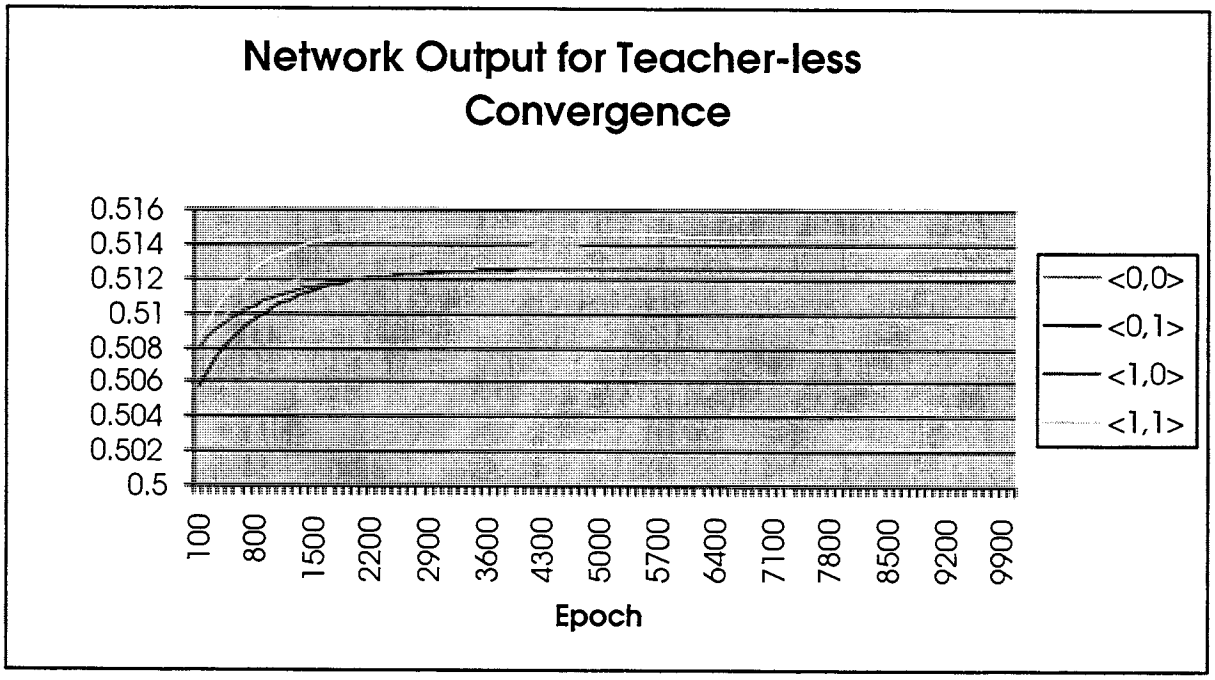


Figure 1 – Output of network stabilizes by epoch 10000.

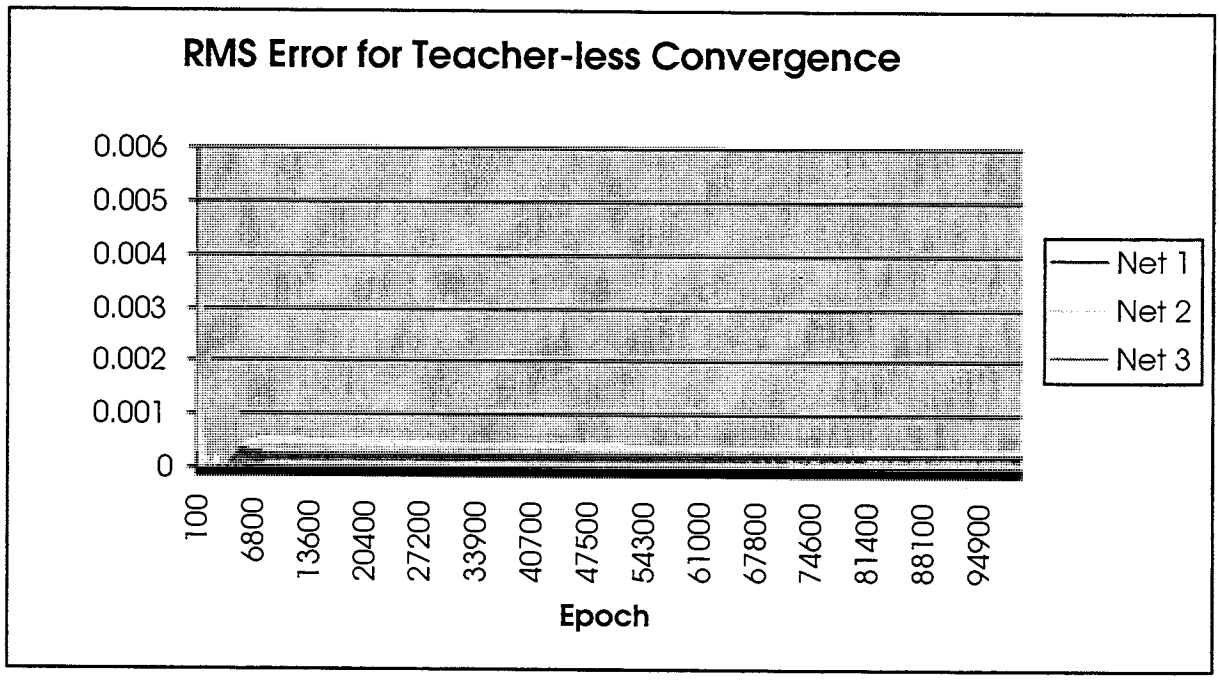


Figure 2 – Error for each net converges to zero.

2.3 Local information and convergence: A colony without teachers

First we demonstrate that a collection of untrained networks can indeed converge to each other. Since we are not trying to teach the ants anything, we assign no significance to the input or output. We merely want to show that after learning from each other, for any given input, all of the ants will exhibit the same output. We used a very simple template for each network, consisting of two inputs, one hidden layer with two neurons, and a one-neuron output layer. We found that the collection of networks actually converged quite quickly (figures 1, 2). The collection converged for both static and dynamic neighborhoods, but converged more quickly for dynamic selection. Though the outputs of all the networks became equivalent after training, we never found a pair of networks with the same internal weights.

2.4 Local information and convergence: Learning XOR

Next, we wanted the input and output to be meaningful, and for the collection to learn a specified behavior. Using the same network template, we taught one to three networks the XOR problem, then placed them, as trained teachers, in a collection of random networks. We then varied the ratio of such teachers to students from 1:9 to 1:3, and the size of any student's neighborhood from 9 to 3 (i.e. each student could see every other student in the class). In most cases the collection converged to correct solutions, except when there were too few teachers, and the neighborhood was too large. We believe this is because adding the output vector of a single teacher does not change the average over a very large neighborhood significantly enough for Backprop to move in that direction fast enough to converge. When the ratio of teachers to

students is higher, a large neighborhood size will likely have a larger fraction of teachers to students, so this converges well. Likewise if the ratio of teachers to students is small and the neighborhood is small, the network will converge well. We believe this is because if a teacher is part of the observed neighborhood then it has a greater effect on the output vector, giving Backprop a sufficiently large error gradient to move down. See figures 3, 4:

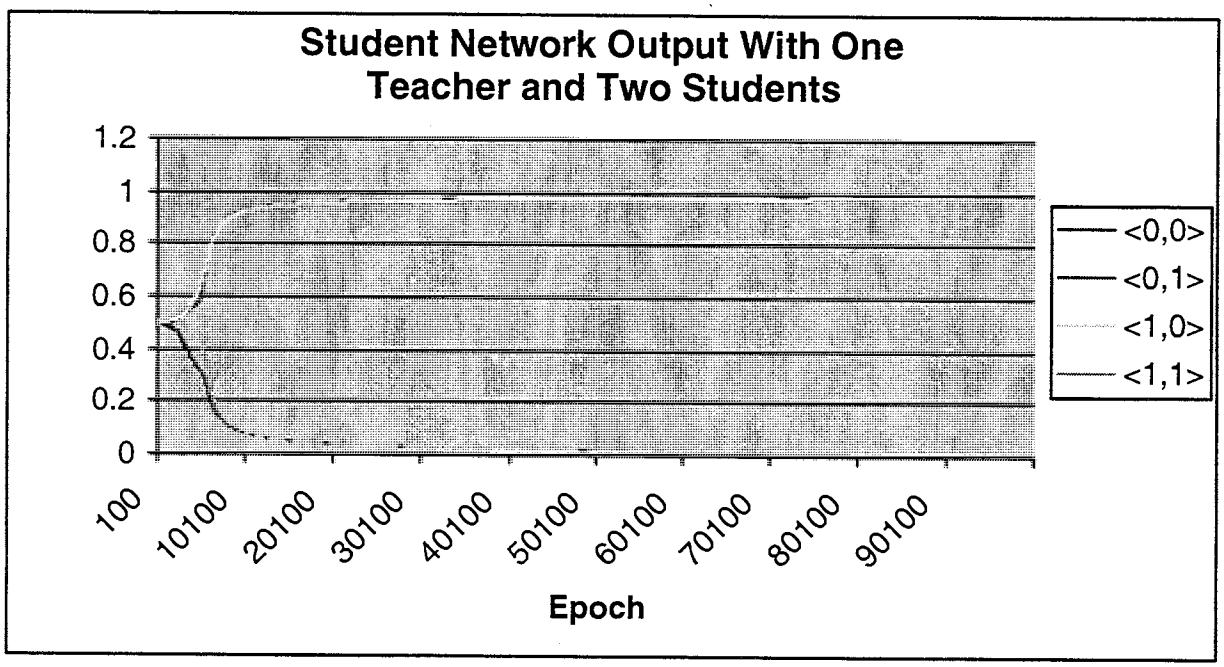


Figure 3 – Student converges to correct outputs for the XOR function.

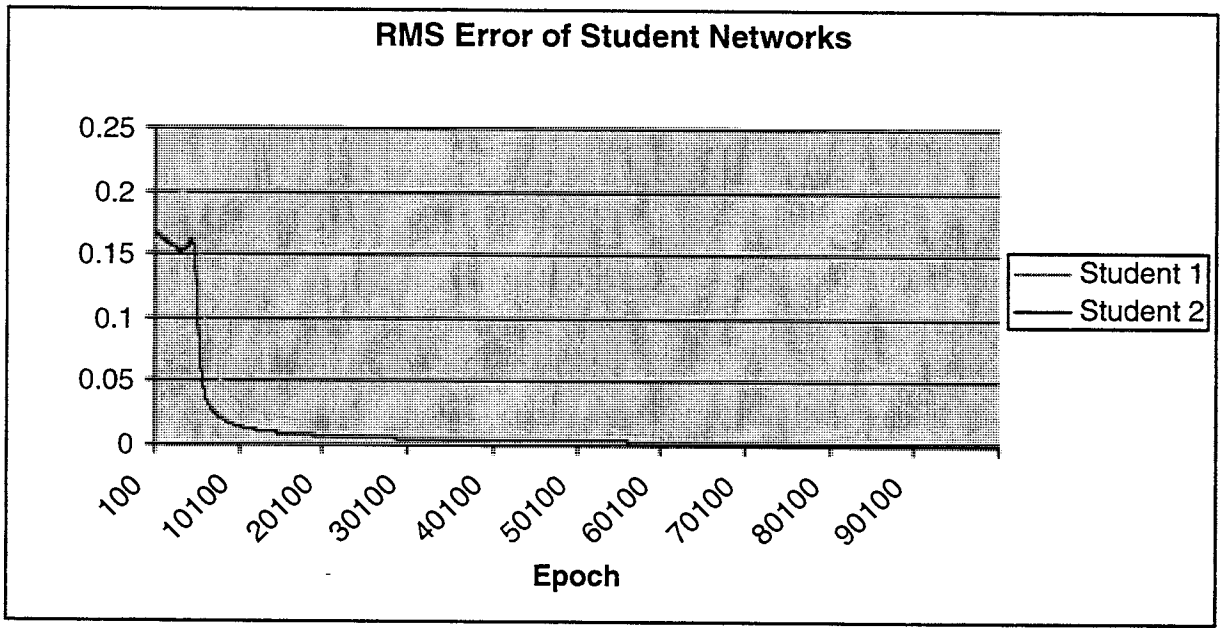


Figure 4 – Student errors converge to zero

3. EXPERIMENT

3.1 Introduction

The construction of our network and the successful convergence of the colony to the XOR function leads to the generalization of attempting to model any arbitrary Boolean logic function. Though this may not seem like a large improvement, the capacity to model any such function implies that this colony can learn to compute any function to sufficient accuracy, as Boolean logic is isomorphic to digital computers, which can do so. (In Future Research, we present a possible method for mapping programs on a digital computer to neural network inputs.)

3.2 Details of the experiment

We chose to simulate the four-input Boolean function $\neg(a \Leftrightarrow b) \wedge (c \vee d)$. (There is no particular significance to this function. It was selected by randomly choosing outputs for each of the possible sixteen input vectors. Any other function could have worked as well.) To accommodate the larger input size, we expanded our network template to include two hidden layers each with four neurons. We added the second layer on the assumption that a conjunction of two terms might require one layer to compute the terms and one to compute the conjunction. This assumption may or may not be founded; nevertheless these networks will converge, as will be seen shortly.

The inputs and output were again chosen to be binary variables, with the same displacement from 0 and 1 as before. The learning rate and momentum constant were held the same as in the prior experiments.

We ran the experiment twice, using a teacher-student ratio 3:6, and a dynamic neighborhood (as explained above) of size three.

3.3 Expectations

Based on our experiences with the XOR problem, which took approximately 16000 epochs to converge, we expected these larger networks would converge, but would likely take much longer, perhaps nearly ten times as long. Also, having observed the various problems that arose the few times that the XOR networks did not converge, we anticipated that the network would correctly classify most of the inputs (i.e. produce outputs of approximately 0.005 or 0.995), but would not quite classify some of them, with outputs around 0.25 or 0.75. If the output of the network were to be used in a discrete simulation, then hard-limiting of these values would indeed produce the correct answers. We chose not to hard limit our results, as we felt the uncertainties

in the answers reflected the possible uncertainties in actual group learning, and were therefore a benefit as opposed to a source of error.

3.4 Results

The networks did indeed converge, but did not take as long as we had expected, nor did they misclassify any inputs. On both runs of the experiment, the networks converged in approximately 16000 epochs, or about as long as the XOR networks did, to within an rms error of 0.04. However, unlike the XOR networks of the prior experiment, which ultimately converged to within $4x10^{-3}$ of the desired output after one million epochs, these networks never converged much closer than 0.02 in the same time span. (See figures 5,6,7).

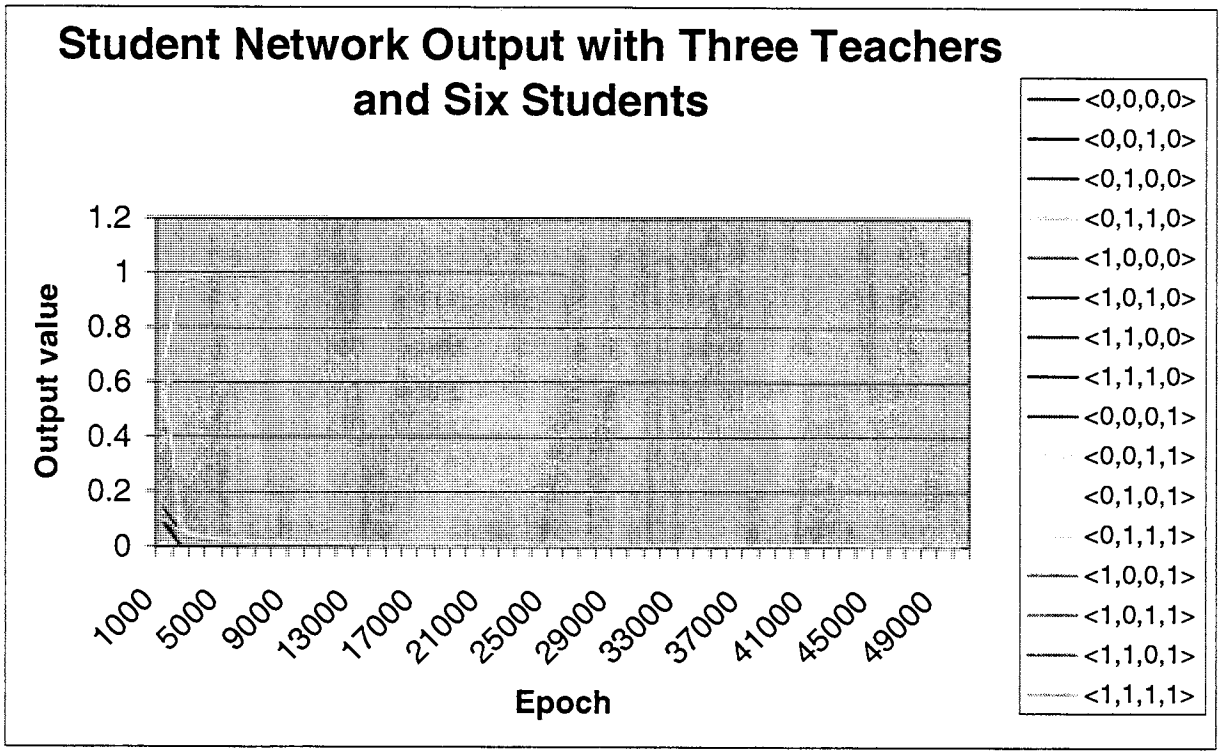


Figure 5 – Student's output converges to the outputs for $\neg(a \Leftrightarrow b) \wedge (c \vee d)$.