Neural Net Applications ‘04

Willard L. Miranker

TR-1315

Table of Contents

Colony Learning Under Duress

William Barley
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract
This project aims to study the effect of recurring noisy influences in locally informed colony learning. Several characterizations of noisy, or "rogue," data sources are used to skew the observations of a group of neural networks during their learning process. While the networks learn by attempting to emulate those around them, they cannot distinguish between good and poor data sources. I conclude that little long-run disruption occurs in the learning process of colony members.

1. INTRODUCTION

Ant colonies function in ways completely oblique to those conventions and norms we are used to as humans, operating in our own societies. This may be a reason their mere existence fascinates us so much. Being accustomed to many types of government and social organization, we might easily conclude that an ant colony has little to no apparent organization; yet, clearly the colony achieves what appear to be organized tasks of complex nature: ants naturally solve NP-Hard problems (Bonabeau, et al., *Swarm Intelligence*).

Not only are ant colonies superficially unorganized while extremely capable, but they are comprised of very simple individuals. Each ant, though well equipped in strength, is a simple creature with quite limited abilities.

Research has been conducted by students at Yale on the training of ants solely through the use of local information (Lerner and Wallace, *Distributed Colony Learning*). That is, the model, used here and in the Yale study, assumes that newborn ants know nothing of how to perform the tasks that will be expected of them, but there are many experienced ants in the colony that do have this knowledge; each young ant's only instinct is to emulate the behavior he observes. The hypothesis is that observation of other ants is sufficient for an ignorant ant to learn the knowledge of the experienced. This has been demonstrated (Lerner and Wallace) and will be used in this project .

While we might accept that new ants watching experienced ants will learn, we might wonder what effect the presence of bad examples might have on their ability to learn the good example. This project examines several types of recurring noisy, or "rogue," influences on the ants learning process.

2. DEFINING THE METAPHOR

2.1 The Colony

For this experiment an ant colony has been modeled as a collection of neural networks. Each ant is represented by a simple feed-forward network, accepting a binary vector of inputs and giving a single output. The networks are adjusted using Back Propagation, with the observed error for each ant based on the difference between his output and the output he observes. The specific learning each network or 'ant' experiences is uniquely based upon his own observations of other ants.

2.2 Three Types of Ants

There are three types of observable outputs, or 'ants,' in the defined environment: there are the new ants, or 'students,' which are the only observers; there are the experienced ants, which we call 'teachers'; and there are ants that give outputs inconsistent with that of the teachers. We will refer to this third type of ant as a 'rogue.'

The students are born into the colony as networks with randomized weights. The randomization models the lack of experience, since a trained network requires relationships between weights in order to achieve proper output. To achieve proper output, an ant does not have to achieve the exact same weights as another, but his network will be equivalent to that of another ant that has also fully learned.

Under the arrangements of an ant colony, we suppose that there are no centralized systems of education, and that an ant learns through the use of local information. For this experiment, it has been assumed that the student ants are unable to differentiate between other students, teachers, and rogues; a student considers any ant it can observe to be a teacher. Learning, for these young ants, means adjusting network weights through Back-Propagating a measure of error. Each ant makes his own estimation of the error in his output by noting what the other ants in his vicinity are doing. In sum, an ant strives to make his output more like the average output that he observes.

The teachers within the colony represent ants that have, by some means, acquired the knowledge necessary to perform the tasks required of them. The teachers have no effective interaction with the students; they simply go about their tasks within the vicinity of the learning ants. Within the context of this study they take binary inputs, just as the new ants, and give a binary. Many binary functions have been used in this experiment, but for all graphs show below, the XOR function is used.

The rogue ants can be conceptualized in any number of instantiations. It could be that environmental externalities have caused some ants to misbehave; it could also be that the ants are enemies, from another colony. There are many forms they could take, and the present study has focused on four. These four realizations of rogue output have been selected for their characteristic behavior, not by their conceptual justification.

2.3 The Rogue Influence

The polymorphisms of rogue ants have two variable characteristics: the regularity with which they appear in the colony and the consistency of their output. The rogues appear either on a fixed interval or randomly with a fixed frequency. They output either randomly or by a rule to be specified.

The number of rogues that occur at once, as well as the duration of their influence has been varied in preliminary experiments, but in this analysis, they should be assumed to be fixed and equal to the number of teachers present. Specifically, the rogues were set up to appear in number equal to the number of teachers present. And, while the teachers were present in every stage, the rogues were allowed to appear in a small fraction of stages. Since the ants have no ability to discriminate between good and bad observations, it would be hard to define a rogue influence that was completely persistent; a rogue influence that that was always present would have to be considered a teacher. In such a case, the ants would have to use means other than observation to filter it from their learning.

2.4 Performance Measures

As a means of measuring the ability of the ants to learn, data were collected to detail at each time step, each ants experience in terms of his observations, his output, and the difference between his output and that of the teacher. Root-mean-squares (RMS error) were taken of the difference between the teachers average output and the average output of the student ants in order to measure the students' convergence toward the teachers. These data were not available to the ants, however. The students simply observe the outputs of other ants, be they students, teachers, or rogues.

3. EXPERIMENT

3.1 The Student Network

During preliminary study, the students' network composition was varied; for the experiment, the network was chosen to be a feed forward network with two inputs, one hidden layer of four neurons, and a single output node. While two and three neurons in the hidden layer produced similar results, I chose four neurons which generally seemed to produce results with less training. As stated above the input was a binary vector.

3.2 Learning from Teachers without Rogues

It is important to first accept that untrained ants, by trying to emulate those around them, will converge on the output of teachers. To that end, a group of six student ants were trained with three teachers and no rogue influence. See figure 1 which depicts the ants' convergence on proper XOR output. The students have little trouble converging in relatively few epochs.

3.3 Final Parameters

The experiment consisted of students training under varying conditions of rogue influence. In tests prior to the experiment, groups were sized around 3:2:2, students to teachers to rogues. In all scenarios with teachers, the portion of teachers was raised to three as with the rogues. It was reasoned that since the teachers are always present and the rogues appear intermittently, when the rogues are present their influence should be at least as powerful as that of the teachers.

The rogues in all cases were allowed to appear about one percent of the time for a duration of around five percent of the time. This means that in all, the rogues were present about 5 percent of the time the students were learning.

All simulations in the primary experiment were conducted with the teachers outputting according the to the XOR function. In each epoch all possible inputs binary inputs are given to the ants. That is, since the ants are computing a two-input binary function, they are given four binary vectors of input in sequence. The students take note of the other ants' outputs for each input.

3.4 Expectations

Going into this experiment, I had hopes of finding that certain types of rogue disturbances in the learning process were beneficial. I thought that while converging to some functions might be difficult to achieve it might be the case that disturbances help to shake up the network weights in a way that resituates the network in a better approach toward convergence.

After some tests, I began to think that the rogue influences would rarely help the network to converge. It seems intuitive that an undisturbed teacher-student group would converge fastest.

3.5 Results

In most all of the trials conducted, the rogues had a significant influence on the students' outputs. The influence was not sustained though. While the ants were very sensitive to the added presence of the rogues, they were just as sensitive to the departure of the bad influence. The ant networks, relieved of rogue input quickly recovered.

See figure 2 for an example. In this scenario the rogues had a fixed interval of appearance and a fixed output. While, the students output changes drastically during periods of rogue influence, it quickly returns to its limiting form.

It can also be observed that the rogue influence was more disruptive when the rogues were using random outputs. See figures 3 and 4. In figure 3, the ants experience a random rogue occurrence, but fixed rogue output. They seem to deal better with the fixed output than with the random output of the regularly appearing rogues in Figure 4.

Looking to the ants' output in figure 5, it can be seen that the randomly occurring rogues had the opportunity to disrupt the learning process early on. This appears to be more significant than later disruptions. Though the ants are still able to converge quickly, their outputs had more trouble converging initially than in other cases. See the contrast in Figure 2 in which the rogues are regularly occurring.

4. CONCLUSION

Even with several different types of rogue influence, the ants remain very steady in their convergence toward the teachers. After temporary shocks, they are able to get right back on track with little overall effect. In fact, the rogue influence on the ants had little effect overall in rate of convergence.

With greater sustained rogue presence the networks would have been more substantially jolted, but without persistence, the teachers would definitely pull the ants into convergence.

I still think that the noisy influence of rogues might be helpful in some contexts. It is possible that, lacking a good training function, the disruptions might prove helpful from time to time. Certainly we know that in nature, diversity and mutation have always contributed to success in the long run.

It is also possible that with more and more difficult solutions, there will appear more chances to fall into local, but non-optimal solutions, in which case perturbations might allow networks to escape from a local solution.

5. REFERENCES

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation. New Jersey: Prentice Hall.
Bonabeau, E., Doringo, M., & Theraulaz, G. (1999). Swarm Intelligence: From Natural to Artificial Systems. New York: Oxford University Press.
Benjamin Lerner, Benjamin Wallace. (2002) Distributed Colony Learning. Yale University.

Figure 1 - Students Converge to Teachers

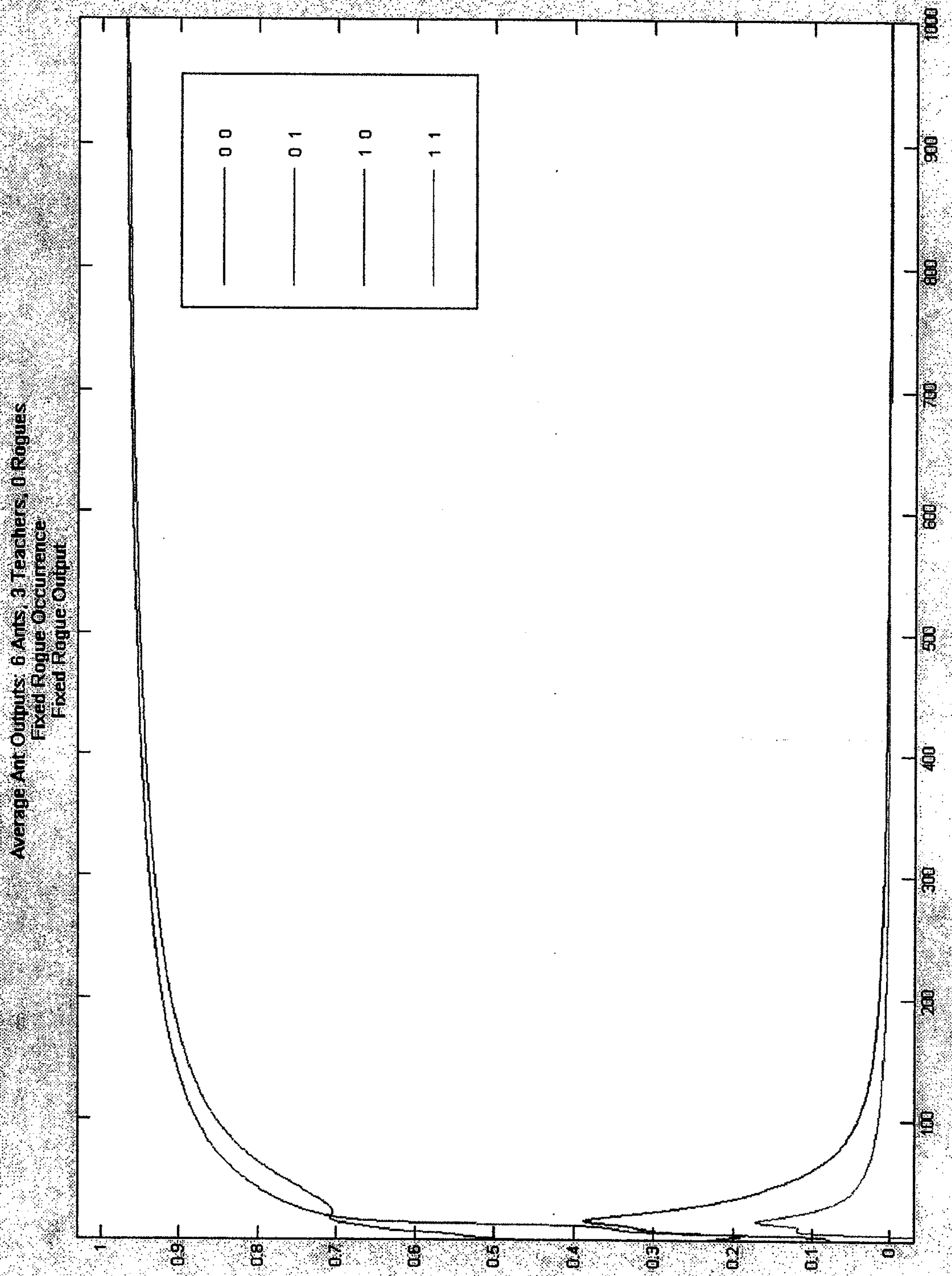


Figure 2 - Students Recover from Rogue Influence

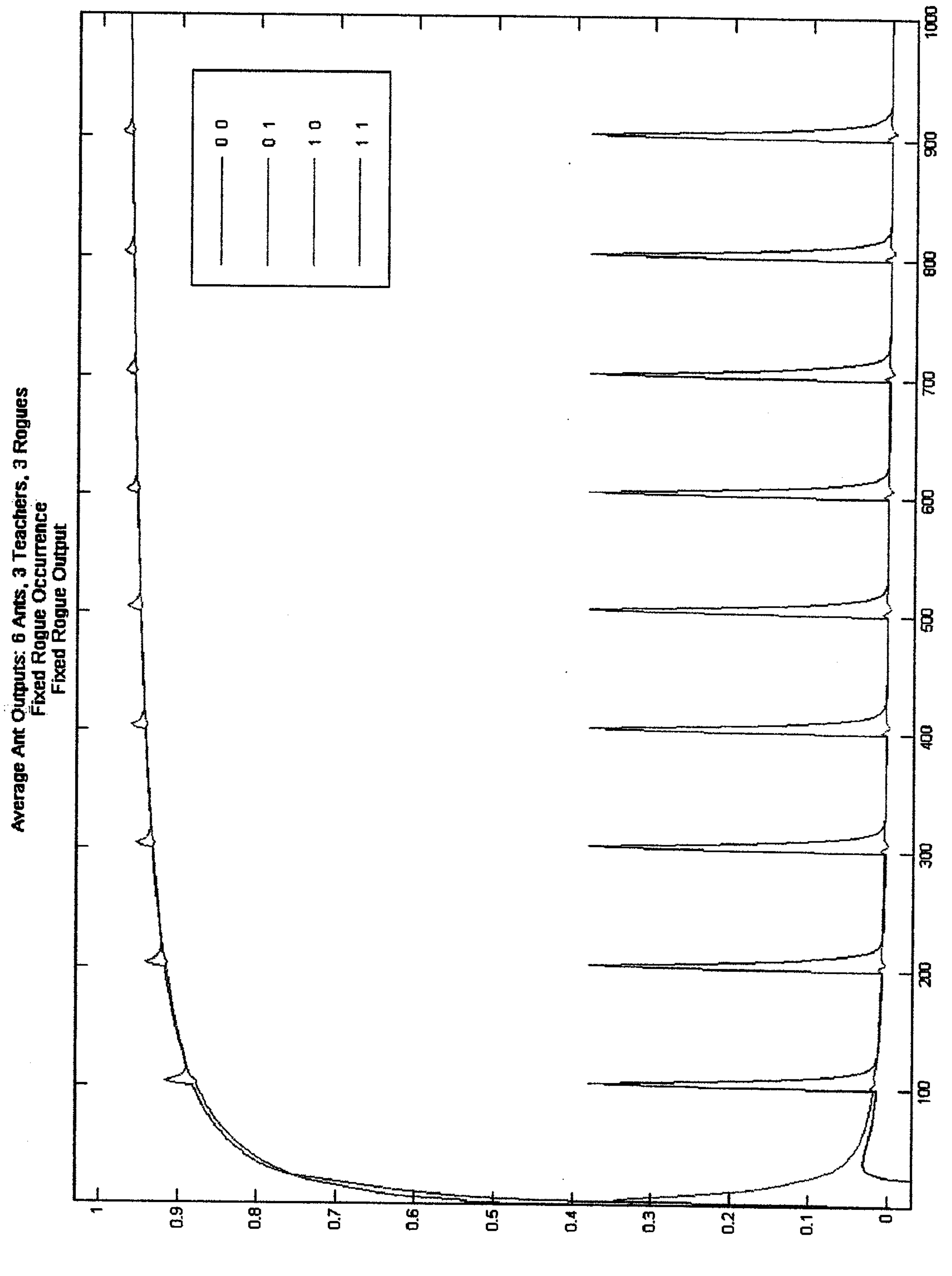


Figure 3 - Ants' Error with Random Rogue Occurrence

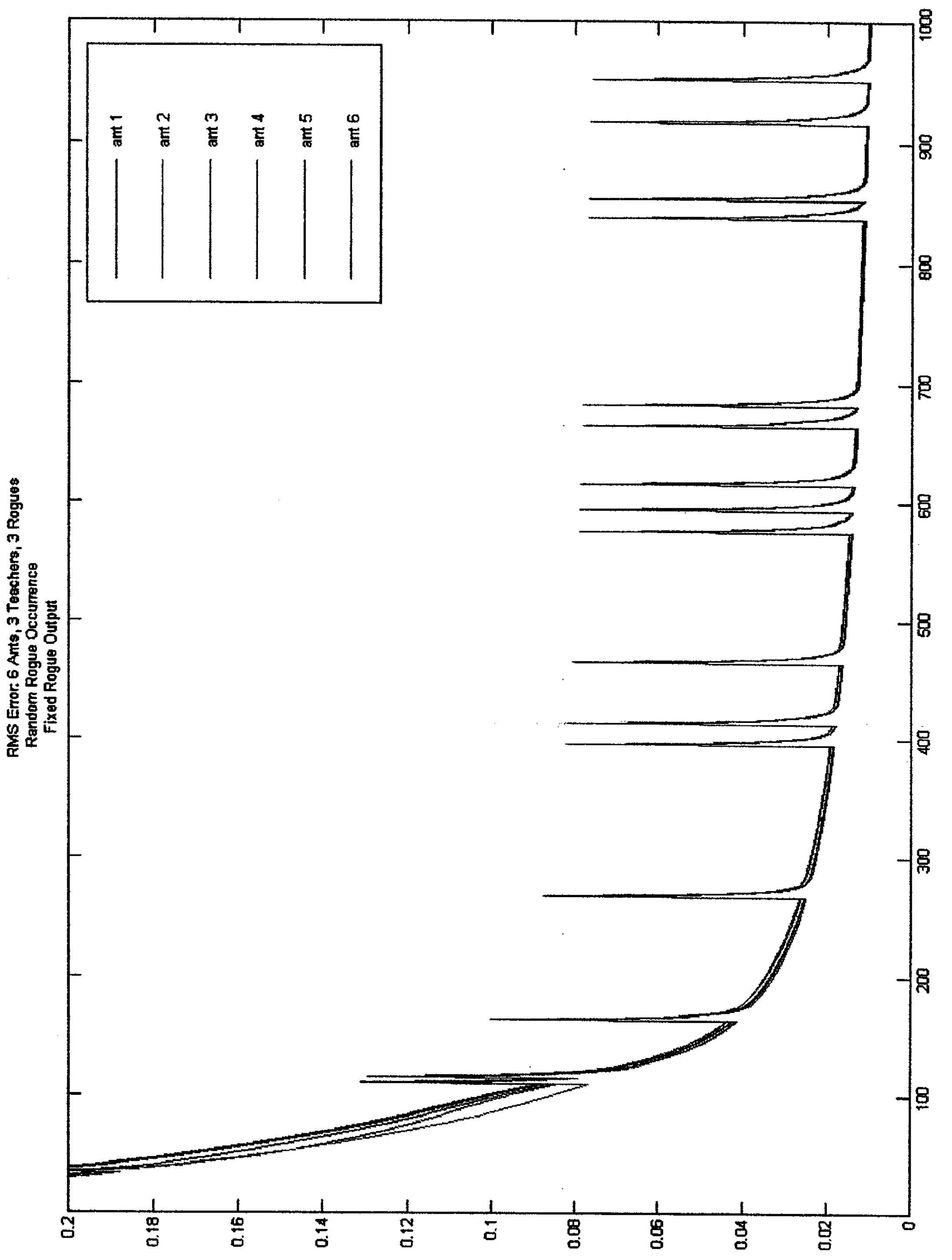


Figure 4 - Ants' Error with Random Rogue Outputs

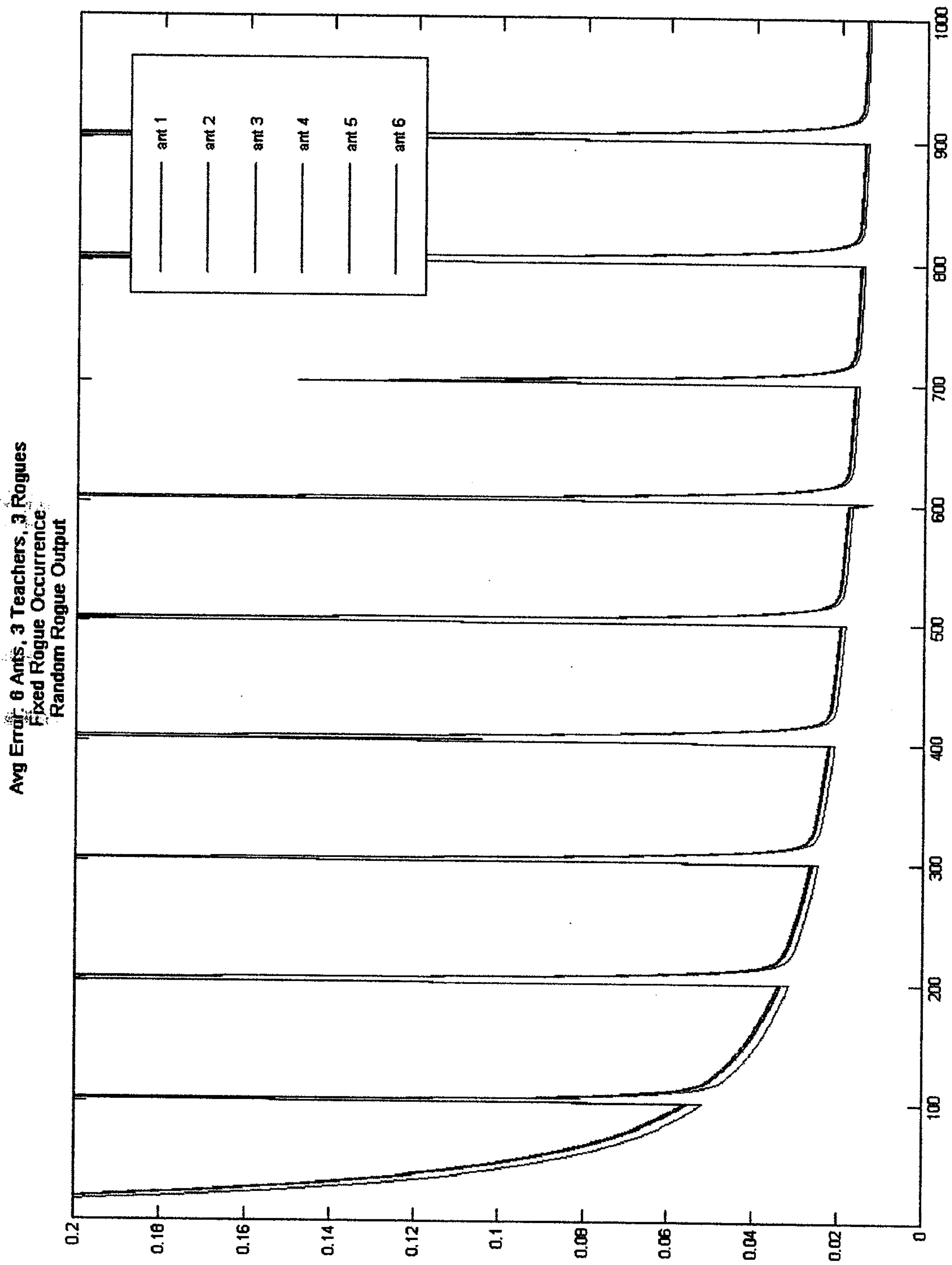


Figure 5 - Ants' Output with Random Rogue Occurrence, Fixed Output

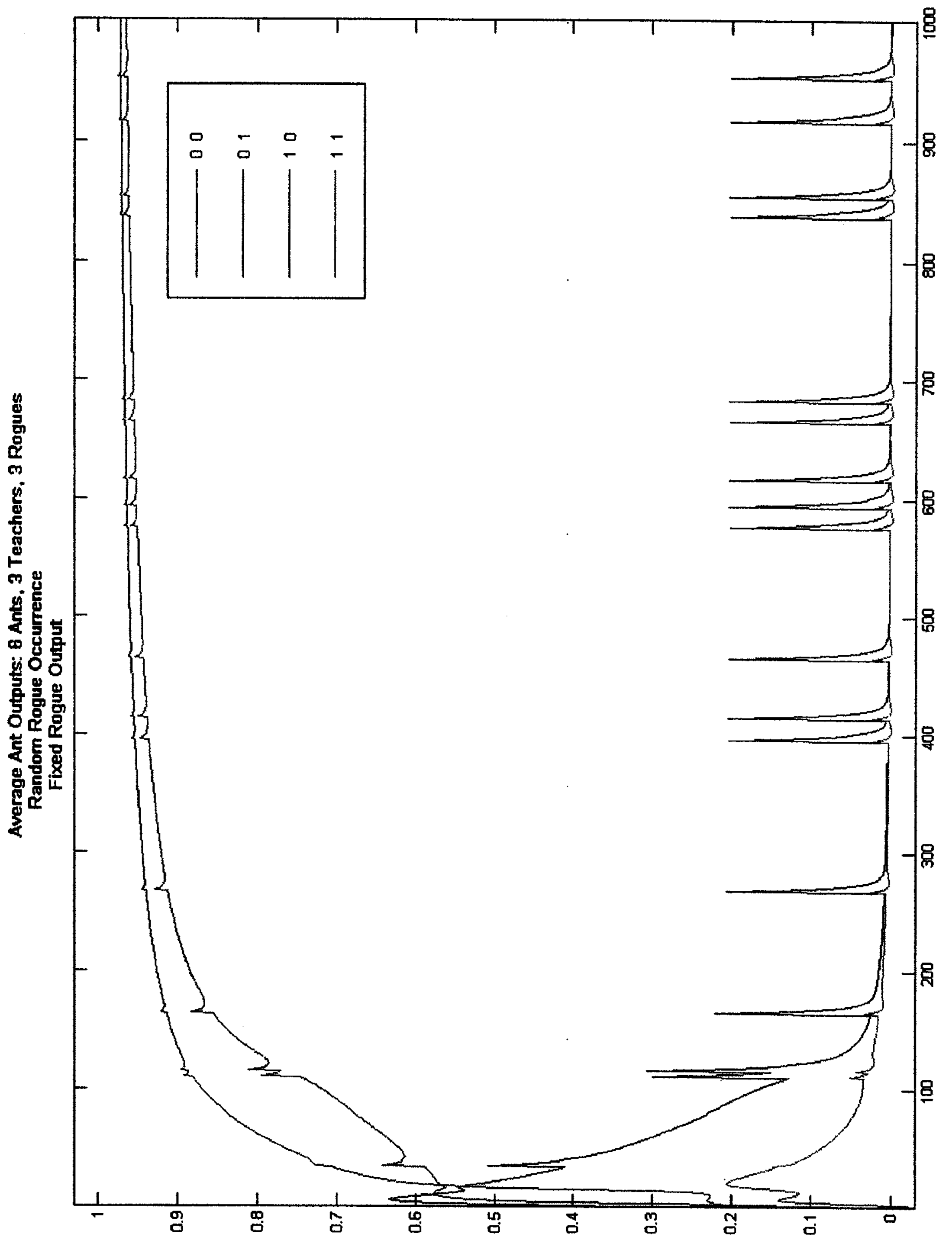


Figure-Ground Segregation using a Neural Network

Timothy Brady
Yale University
Department of Cognitive Science

Abstract

A neural network was used to determine the reliability of two forms of figure-ground segregation, and the practicality of the brain implementing one or both of them. One model posits the existence of purely excitatory connections between neurons that respond to the same type of feature, causing an object to "glow" out of the background when it shares activations in many feature maps. The other posits the existence of lateral inhibition between neurons that code for nearby regions of space as well as similar features, causing the edges between objects to become enhanced. The reliability of both systems and the results produced suggest that the brain needs both types of connections in order to engage in the accurate figure-ground segregation that is typical of our perceptual experiences.

1. INTRODUCTION

1.1 Figure-ground segregation

Our brain is extremely adept at separating the visual field into objects and their background, as we may note every time we look around a room. We find it quite easy to focus our attention on one specific object, even if it is embedded in a collection of many other objects very much like it. For example, on a chessboard we do not find it difficult to see the pieces as separate entities from the squares beneath them, even if they share colors and overlap shapes.

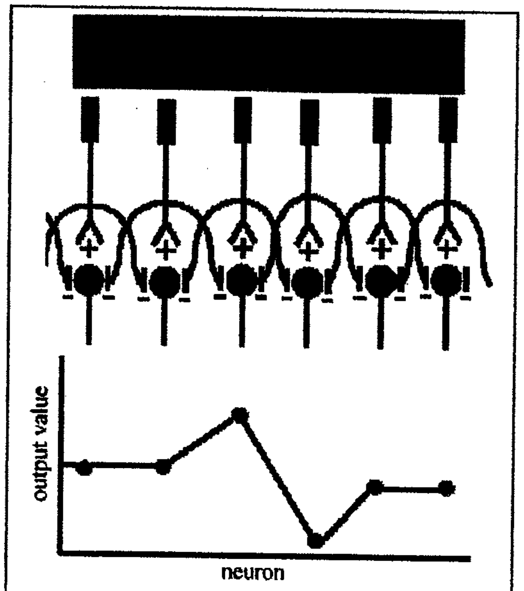
Many different cues are exploited to detect boundaries between objects and their background, but in general regions that show abrupt changes are assigned to be boundaries whereas relatively homogenous regions are grouped together (Lamme, 1995). This is especially clear in the areas of color and depth: when an area of the visual field abruptly changes color or becomes farther or closer away than the adjacent area, we almost always perceive the two areas as different objects (Wolfe, 1994). This allows us to see objects as different from their backgrounds and as distinct from other nearby and possibly overlapping objects, and have a perceptual experience that more closely mirrors the actual existence of objects in the real world.

1.2 The Lateral Inhibition (LI) Model

It was originally posited by Stemmler, Usher & Neibur (1995) that this figure-ground segregation was achieved in the striate cortex in much the same way that edge detection is implemented in the connections of the retina. That is, it was assumed that lateral inhibition between nearby neurons was used to enhance the contrast at boundaries, since the inhibitory input to boundary neurons would only be coming from the side of the image that was similar to the current point and not from the opposite side. This would result in a ring of less inhibition and thus greater activation in the areas that were the boundaries in a given feature map[1], helping us to differentiate them as important borders.

[1] A feature map is an activation map representing the amount of a given feature at each location. For example, we can consider a color feature map, an orientation feature map, a depth feature map, etc. See Wolfe (1994) for a further explanation of the idea of feature maps in vision.

This strategy of lateral inhibition would be repeated in each feature map, and the areas that showed up as boundaries in multiple feature maps would be considered to be the borders between objects. Thus an area that showed abrupt changes in color, depth, and orientation would be marked as a clear object boundary, whereas one that showed only a change in color might be considered only a candidate for a boundary.



A diagram of how the LI model helps differentiate boundaries

In general, this model seems to be a fairly good explanation of how object boundaries might be parsed out, but it is not entirely satisfying as a full explanation of feature-ground segregation. In particular this model is convincing from a phenomenological standpoint, because we believe we perceive objects, not their boundaries. For example, when we look at a car on the road we like to believe that what we perceive is a car with the road behind it, not the edge between the car and the road. While it is possible to think of this as first perceiving the edge and extrapolating it into objects based on topographical clues in some higher order process, the low level at which object processing happens[2] seems to indicate otherwise. Thus, while lateral inhibition is clearly used in determining the edges of light in retinotopic space (Purves, et al, 2004) and may be useful early on in the process of feature-ground segregation to help

[2] In particular, object representations are clearly available preattentively (Scholl, 2001), and there are suggestions that differential activation based on the object an area of the visual field is part of may be seen as early as V1 (Lamme, Zipser & Spekreijse 2002)

determine the boundaries of objects, it does not seem possible to perform all of the processing relevant to our entire perceptual experience using lateral inhibition alone.

1.2 The "Glow" Model

The problems with the LI model led to the postulation of excitatory connections between the neurons of each feature map by Roelfsema, Lamme & Spekreijse (2000). They hypothesized that if all neurons that coded for a specific feature (eg, red) had excitatory links to each other that allowed them to reciprocally excite each other, that areas of space with like features would seem to 'glow' (i.e., stand out as a result of having more activity) from their background in the given feature map (eg, a color map). Thus, when we presented a vertically oriented red object on a different colored background, the red object would appear to 'glow' in both the red color and vertical orientation feature maps. This is because the neurons that coded for red would reciprocally excite each other and would be more active than the neurons responding to the black background, and the same for the vertical neurons in the orientation feature map. This would help to differentiate the vertically oriented red object from its background, leading to early figure-ground segregation.

This Glow model seems to succeed exactly where the LI model failed – by grouping objects early in perception as combined regions sharing mutual activation, it allows us to justify our phenomenological experience of what it is like to perceive the world. This is because rather than emphasizing boundaries like the LI model does, it segregates the entire object into one region, which is how we believe we see objects (e.g., the car on the road example above). However, it does not have other features of the LI model, such as the ability to easily extract a border between objects that may be similar in several feature maps. For such objects, the LI model's approach of mapping boundaries from multiple feature maps is likely to elucidate much more accurate figure-ground segregation than the approach of the Glow model.

1.3 Two Contradictory Models

In general, there appear to be merits to both the LI and Glow models of figure-ground segregation, particularly in how they fulfill our expectation of what a good model of figure-ground segregation should look like. In such a model, we would like to see strong segregation between objects, but also a general activation of entire objects rather than an emphasis on the boundaries between objects. Thus, it seems like it would be nice to combine the two models in order to get the best of both: an improved ability to separate like-featured objects with boundary enhancement, and an overall activation of the entire area of the object in order to satisfy our phenomenological expectations of objects being grouped together properly.

However, these two views employ directly contradicting connections between neurons: the LI model posits that neurons coding for a certain feature must inhibit nearby neurons doing the same thing in order to enhance edge detection, whereas the Glow model posits that these like-featured neurons must excite each other in order to cause regions with homologous features to differentiate themselves from their background in a given feature map. This presents a clear dilemma: we cannot have a set of neurons both inhibiting and exciting each other in order to derive our normal object-based perceptual experience, as all inputs would just cancel out and result in no net gain of information (and, consequently no perceptual experience at all!).

In order to get at the implications of this dilemma and to determine the extent to which it is possible for the two models to work together, we used a neural network to create a connectionist implementation of both models and to predict the ability to which they can be combined to get a more satisfying model of figure-ground segregation.

2. METHODS

In order to make the models clearer we used only one feature map in their implementation, color. We choose color because of the ease with which it can be extracted from an image and its unambiguous nature. We used 64x64 pixel images with 3 color streams (Red, Green, & Blue) for input. There was an input neuron for each location in the image and each color value for that location, resulting in 64x64x3 input neurons. These neurons were connected to each other and the output nodes according to the predictions of the two models. That is, in order to implement the LI model each neuron was connected only to neurons that coded for the same color it did (R, G, or B) and these neurons were connected by a weight calculated using (1).

$$W_{ij} = -2 / \sqrt{((x_i-x_j)^2 + (y_i-y_j)^2)} \qquad (1)$$

where x_i, y_i is the location in the image that neuron i codes for. This resulted in greater inhibition between neurons that were close by each other and coded for the same color, as per the LI model.

In the Glow model, all neurons that coded for the same color were connected to each other with a weight of 2, and all neurons that coded for different colors were not connected (see (2)).

$$W_{ij} = \begin{cases} 2, & \text{if } i, j \text{ both code for the same color} \\ 0, & \text{if } i, j \text{ code for different colors} \end{cases} \qquad (2)$$

The output values for each location were derived by using (3), which resulted in a number between 0 and 1 for each location in the image (64x64 output neurons), with the number representing the location's relative activation in each of the three color maps.

$$y_i = {}^{2}/_{(64\mathrm{x}64)} \sum_{\substack{j \\ j \neq i}} ({}^{1}/_{255}\; x_j\, W_{ij}) \qquad (3)$$

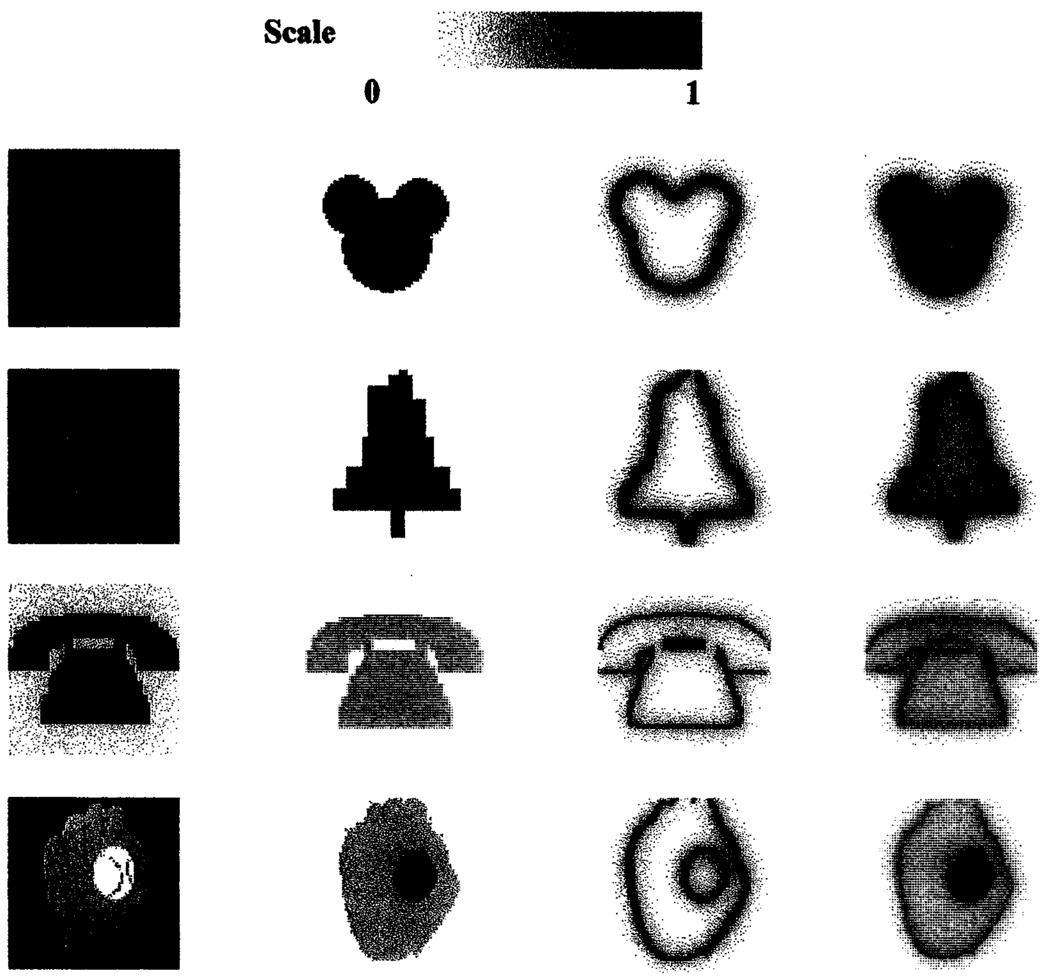
The job of the network was to differentiate between the figure and its background by assigning higher values in the output layer to the figure than to the background. We ran the network using only the LI model's weights, only the Glow model's weights, and then a combination of the two models. The combination used the output values of the other two models as inputs and derived a new output value based on them, using (4).

$$y_i = \theta_1\, y_{Glow} + \theta_2\, y_{LI} \qquad (4)$$

where the θ values were both set to 1 for the outputs displayed below.

3. RESULTS

The results are presented in image form for clarity; red is considered a value of one, and yellow a value of zero in the output node corresponding to a given spatial location.



The test images – just an object against a background.

The results of the "Glow" model

The results of the LI model

A linear combination of processing the two models separately.

The results of the models are very much as predicted (Roelfsema, Lamme & Spekreijse, 2000; Stemmler, Usher & Neibur, 1995). The LI model produced an image with the edge of the object clearly contrasted, to mark where the boundaries between objects ought to be in this feature map. In the areas of homogenous color (the center of the object and the background), the inhibition is strongest and so they show the least activation. On the boundaries between objects we see the most activation, since they are only being inhibited from the side with the same color as them.

Likewise, using the Glow model with excitation between common features caused the object to 'glow' out of the image compared to its background, as per the model's name. In these images, we see no boundary enhancement but a clear delineation of the object as a whole. In general, both models acted as predicted in isolating particular areas of the images.

Although implementing both models on the same input is not possible because inhibition from the LI model would cancel out the excitation from the Glow model, it is possible if we first calculate their outputs and then combine them. In other words, if we create two separate networks (with two different copies of the input) for the processing of the models and only after their output is already determined combine them in a third layer, we can combine the benefits of both models. Thus by doing a linear combination of their respective outputs, we can create a new model with the appealing properties of both. As you can see from the third set of output images above, this method is indeed effective – it results in images that combine the beneficial boundary detection properties of the LI model with the holistic object activation of the Glow model. These images have enhanced boundaries on the edges of the objects, but also have significant activation of the entire object as compared to the background.

Thus, while it is indeed not popular to combine the LI and Glow models intrinsically, if we process them separately it is possible to combine their outputs to create a new model of figure-ground segregation. This model allows for both a satisfying phenomenological explanation of complete object activation and the benefit of easy segregation of similar objects based on shared boundaries across feature maps.

4. DISCUSSION

The model presented above calls for the input to processed twice, once according to the LI model and once according to the Glow model, and for these to be combined later. This seems to be redundant processing, and could be seen as unlikely to exist in our brains. However, most of the visual cortex is made up of redundant processing of information for each spatial location, organized into hypercolumns and stripes (Kandel, Schwartz & Jessell, 2000), so the idea that figure-ground segregation might be another way in which visual information is processed redundantly in order to get a better image of the real world is not an unlikely hypothesis. The connections within each of the two models may each be present in different layers of cortex, for example, processed independently. They could later be combined to obtain a version of figure-ground segregation with the useful properties of both, so long as both were already fully processed when they were combined -- as we showed above using a connectionist architecture.

In conclusion, our connectionist modeling of the two major theories of figure-ground segregation helps to show that their author's predictions of their effects on image processing were indeed accurate. The LI model is in fact effective at enhancing the boundaries between an object and its background, and the Glow model is quite effective at causing a holistic activation of an object. Our modeling also pointed to the idea that while the LI and Glow models may seem mutually exclusive on the surface, it is in fact possible to combine them and get many of the benefits of both. We have also attempted to explain how this is plausible implementation of figure-ground segregation in the brain, regardless of its redundancy, because of the inherent redundancy of much of the visual processing in the brain.

REFERENCES

Kandel, E. R., Schwartz, J. H., & Jessell, T. M. (2000). *Principles of Neural Science, Fourth Edition.* New York: McGraw-Hill Health Professions Division.

Lamme V. A. F., Zipser K, Spekreijse H. (2002). Masking interrupts figure-ground signals in V1. *Journal of Cognitive Neuroscience*, Vol. 14, No. 7, pp. 1044-1053

Lamme, V. A. F. (1995). The Neurophysiology of Figure-Ground Segregation in Primary Visual Cortex. *Journal of Neuroscience*, Vol. 15, No. 2, pp. 1605-1615

Purves, D., Augustine, G. J., Fitzpatrick, D., Hall, W. C., LaMantia, A. S., McNamara, J. O., & Williams, S. M. (2004). *Neuroscience, Third Edition.* Massachusetts: Sinauer Associates, Inc.

Pylyshyn, Z. W, & Storm, R. W. (1988). Tracking multiple independent targets: evidence for a parallel tracking mechanism. *Spatial Vision*, Vol. 3, pp. 179-197

Roelfsema, P. R, Lamme, V. A. F, & Spekreijse, H. (2000). The implementation of visual routines. *Vision Research*, Vol. 40, pp. 1385-1411

Scholl, B. J. (2001). Objects and attention: the state of the art. *Cognition*, Vol. 80, pp. 1-46

Stemmler, M, Usher M, & Neibur, E. (1995). Lateral interactions in primary visual cortex: A model bridging physiology and psychophysics. *Science*, Vol. 269, pp. 1877-1880

Learning Rates and Information Content in Hebbian Networks

Christopher Crick
Department of Computer Science, Yale University
New Haven, CT 06520

Abstract
We investigate the behavior of a neural network model of the hippocampus during the learning process, and demonstrate how differences in initial distribution of synaptic weights lead to differences in the ease with which the network assimilates new information. Specifically, we examine the rate at which a neural network learns, by means of unsupervised Hebbian reinforcement, to recognize the letters of the Roman alphabet. We then examine how the rate at which the network learns affects how well it later undertakes a similar task in learning Greek, while incorporating a model of hippocampal apoptosis and neurogenesis and investigating its effect on the learning rate. It turns out that networks that converge very quickly on the Roman alphabet take much longer to handle the Greek, while networks which converge over an extended timeframe can then adapt very quickly to the new language. We find that the effect becomes more and more pronounced as the number of neurons in the dentate gyrus layer decreases, and identify a strong correlation between cases where the Roman alphabet is quickly learned and cases where a few neurons saturate many of their weights almost immediately, preventing other neurons from participating much at all. Cases where the Roman alphabet requires more time result in many more neurons participating with a larger diversity in weights. We present an information-theoretic argument about why this implies a better, more flexible learning system and why it leads to faster Greek alphabet learning, and propose that the reason that apoptosis and neurogenesis work is that they promote this effect.
Keywords - neural networks, information theory, neurogenesis, Hebbian learning

1. INTRODUCTION

The hippocampus, a brain structure located in the medial temporal lobe, appears to be responsible for establishing novel associations during the learning process. As the brain forms new associative memories, hippocampal neurons change their stimulus-selective response patterns [5]. This change in response patterns suggests similarities to the learning processes of artificial neural networks. Since the detailed internal behavior of real neurons is difficult to investigate, we can hope to gain insight by studying the behavior of their simulated parallels.

Evidence also suggests that the hippocampus is a lively site for adult neurogenesis, the generation of new neurons throughout an individual's lifetime [2]. Previous studies have suggested, in computer models, that this process assists the hippocampus in assimilating new information [1]. Selective replacement of neurons in hippocampus models seems to improve the network's plasticity, enabling faster learning.

These discoveries lead naturally to more questions. Why does this cycle of cell death and rebirth increase learning plasticity? What is happening in the individual neurons during the learning process? What are the circumstances and conditions that lead to successful learning? Information is obviously stored within neurons, but how can we characterize that information? How can we recognize it?

Using W. Miranker's apoptosis model as a starting point [4], we investigated the behavior of a neural network intended to represent the hippocampus as it attempted to learn first the Roman and then the Greek alphabets. We uncovered surprising behavior with regards to the ease with which a network learned the two alphabets. In a nutshell, when we present a randomly-weighted network with the Roman alphabet, sometimes convergence occurs very swiftly, other times more slowly, and yet others not at all. We discovered that quick-converging networks have a much more difficult time when presented subsequently with the Greek alphabet than do those that were required to invest more time in learning the Roman. To draw an analogy to human learning behavior, it is as if networks lucky enough to be born with a knack for a particular topic are at a disadvantage when faced with something new, as opposed to their peers which had to figure it all out the hard way.

This characterization is obviously a little glib, but the tendency is very evident. We tried to understand why this might be so, and discovered patterns based on the neuron participation rate and the level of saturated neurons. This paper presents these results. We briefly summarize our experimental setup, which draws heavily on previous work. We then present our new findings and provide some analysis of the neurons' behavior, including a brief look at neuron weights and firing patterns from an information-theoretic perspective. We conclude with a discussion of the work's contributions and potential extensions.

2. EXPERIMENTAL SETUP

We performed the experiments on a three-layer neural network using Hebbian learning to update weights in an unsupervised fashion. The details of the setup are very similar to the model described in [4], though the circumstances under which certain neurons undergo apoptosis are somewhat different. We summarize the model and point out the differences; a more detailed treatment can be found in the referenced paper.

Three layers comprise the human hippocampus – the endorhynal cortex, the dentate gyrus (DG) and CA3. Our model, likewise, contains three layers of neurons. The input layer consists of 35 neurons, each of which represents a single bit of a 7x5-bit vector portraying a pixelated version of a Roman or Greek letter. Each of these neurons feeds its output to neurons in the second layer (12 to 24 neurons), which connect in turn to the neurons of the output (third) layer (11 to 13 neurons).

In addition, the second and third layers both contain lateral connections. While the forward links between layers are excitatory, the intralayer connections inhibit neuron firing. Inital weights were chosen randomly between 0 and 0.1 for the forward synapses, while the weights of the intralayer connections were chosen randomly between -0.1 and 0.

We experimented with different levels of connectivity. In the simplest arrangement, each node was completely connected – each neuron's output connected to every neuron in its own layer and every neuron in the succeeding layer. These connections could disappear if the learning process drove their associated weights to zero, but every synapse began the process with a nonzero value. In other tests, each weight had a certain chance of being set to zero during the network's initialization, representing a nonexistent connection, and we furthermore prevented these weights from changing during the learning process. We used values of 25%, 50% and 75% respectively for the proportion of these nonexistent connections. Motivated by the process of neurogenesis, we were trying to represent neurons that don't necessarily spring into being fully connected with all of their counterparts.

Each neuron used a simple McCulloch-Pitts threshold function, namely

$$v_i = \Sigma_{j \neq i} w_{ij} y_j + \Sigma_k w_{ik} y_k$$

$$y_i = \begin{cases} 1, & v_i \geq \Theta \\ 0, & v_i < \Theta \end{cases}$$

Here, y_i is the neuron's output, neurons indexed by j are neurons that share the same layer, and those indexed by k belong to the previous layer. w_{ij} and w_{ik} are the weights associated with the connection between each neuron pair. Θ is set to 0.1.

Hebb's law specified the learning process. Each time a neuron has the opportunity to fire (whether it actually fires or not), it updates its weights according to the following equations:

$\Delta w_{ik} = \tau(1.5(y_i y_k) - 0.5(y_i) - 0.5(y_k))$ (forward connections)

$\Delta w_{ij} = \tau(-1.5(y_i y_j) + 0.5(y_i) + 0.5(y_j))$ (lateral connections)

Weights were not permitted to grow indefinitely; maximum and minimum values were capped at 0 and ± 0.125 for the excitatory and inhibitory connections, respectively.

τ is the learning rate constant. For these experiments we found that a fixed rate of 0.0001 worked best; we tried several strategies for updating the rate according to the elapsed time or the Hamming distance between two successive epochs, but we found no strategy that performed better than using a fixed constant.

To present an alphabet to the network, each input neuron outputs a 1 or a 0 corresponding to whether a bit is black or white in a particular letter image. One by one, the neurons of the input layer fire, then those of the DG layer, and finally the output neurons. The firing order within a layer is fixed – at the beginning of each learning attempt, a random ordering of the neurons is chosen, and this firing order is maintained from epoch to epoch.

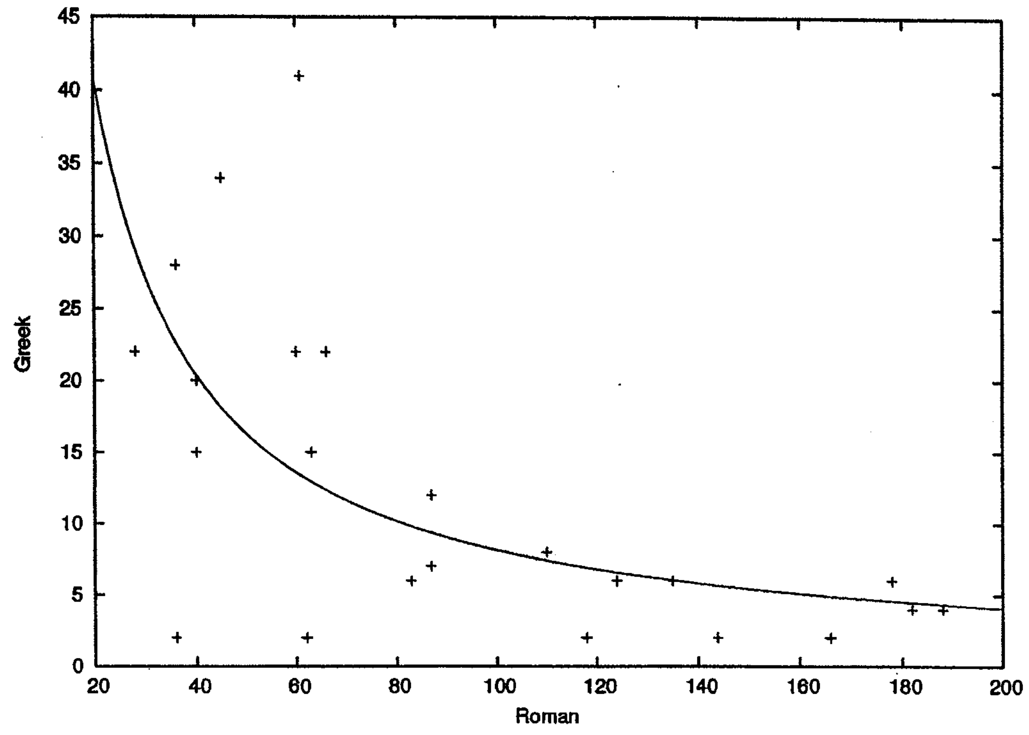


Figure 1: Convergence performance with 16 DG nodes

The output corresponds to a vertex on the unit 2^n-cube, where n is the number of neurons in the output layer.

The learning process proceeded by presenting all of the letters of a given alphabet, in order, until the output was perfectly consistent over three successive alphabet-presentation cycles *and* every letter mapped to a different vertex on the output hypercube. Because true convergence according to this definition rarely emerged, we loosened the requirement, considering convergence to have occured when the network settled on a unique representation for all but 10-15% of the letters.

Finally, we modeled cytotoxicity and neurogenesis by assuming that neurons with highly-saturated weights were more likely to perish from overuse. Whenever we conducted a round of apoptosis, we assigned to each node a probability of death. Each node added the absolute values of all of its weights together, and the probability of cell death increased linearly from zero, according to the distance by which the weight sum exceeded a threshold. This threshold varied based on the total number of inputs leading into each neuron, from about 1.96 to 2.46.

3. RESULTS

We performed a large variety of experiments with this setup, some with very interesting results and others with inconclusive ones. We will first dispense with the latter.

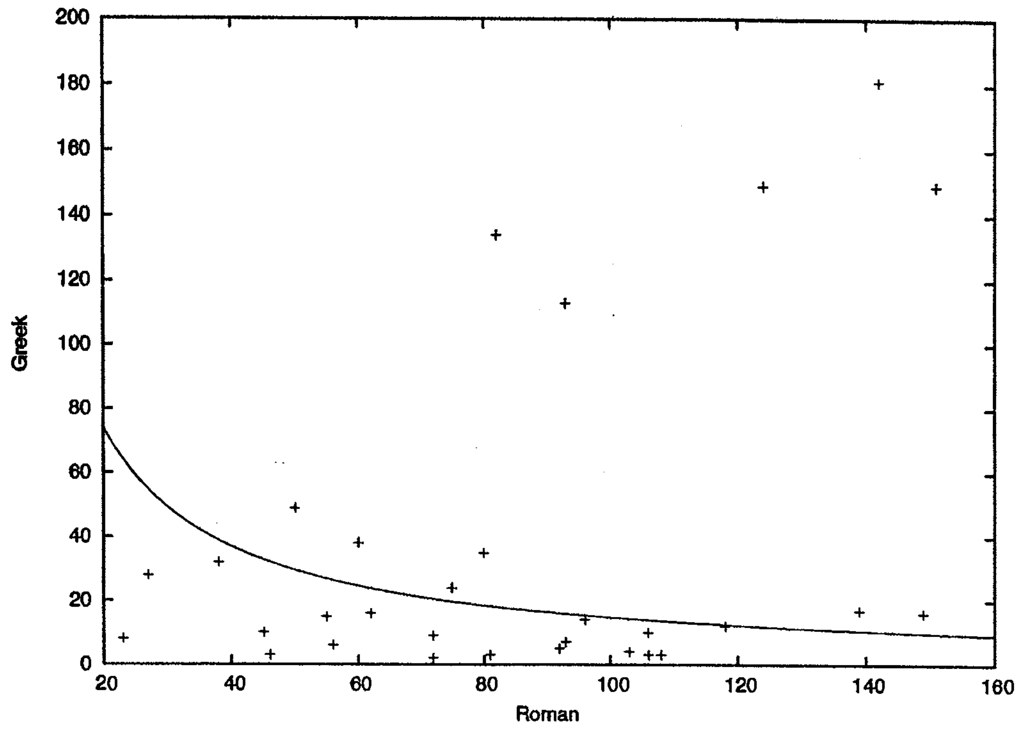


Figure 2: Convergence performance with 24 DG nodes

Our first set of experiments focused on determining performance parameters and convergence criteria. We determined that a convergence criterion of 90% worked well enough – we did not consider a network to have converged until it consistently represented at least this fraction of letters with a unique output encoding. Allowing the network to misclassify more letters improved the number and speed of convergences, of course, but not enough to be worth sacrificing accuracy. Besides, we wanted to investigate differences in convergence timing, and reducing that effect would blunt the data.

We found that an 11-dimensional output vector is usually insufficient to converge on a good encoding, while 13 dimensions seems adequate. With 11, the network converged within 400 epochs only 4% of the time, while with 13 the number fraction was 46%. All of the following experiments therefore use the larger output vector.

We began with tests that allowed a network 400 epochs to converge, but soon noticed that only in 2% of cases did this convergence take place after the 200th iteration. Therefore we changed the ceiling – any trial which failed to converge within 200 epochs was eliminated.

We note that the attempt to model partially-connected networks failed to produce any useful results. With only 25% connectivity, we never managed to produce a network that converged within the time limit. Performance improved at the 50% and 75% connectivity levels, but the only noticeable difference between the results from a fully connected network and those from the partially connected ones was that the latter produced them more rarely.

Figure 1 shows the first interesting result. Here we plot the number of epochs it took for a network to learn the Roman alphabet against the number of iterations it subsequently spent learning the Greek alphabet. The continuous plot (in this and all subsequent figures) is a least-squares best-fit of the function $f(x) = ax^b + c$. In nearly every case, the Greek was easier for a network to learn after having been exposed to the Roman. But the key result is that a network which happened to converge very quickly with the Roman alphabet almost always took a comparatively long time to do the same with Greek, while networks which had to work for a long time before finally learning Roman letters took to Greek like lightning.

Figure 2 shows the same kind of data, but for a network with a larger dentate gyrus (24 as opposed to 16). Here, the effect is much less pronounced, and several outliers spoil the picture. Disregarding those, the trend still exists.

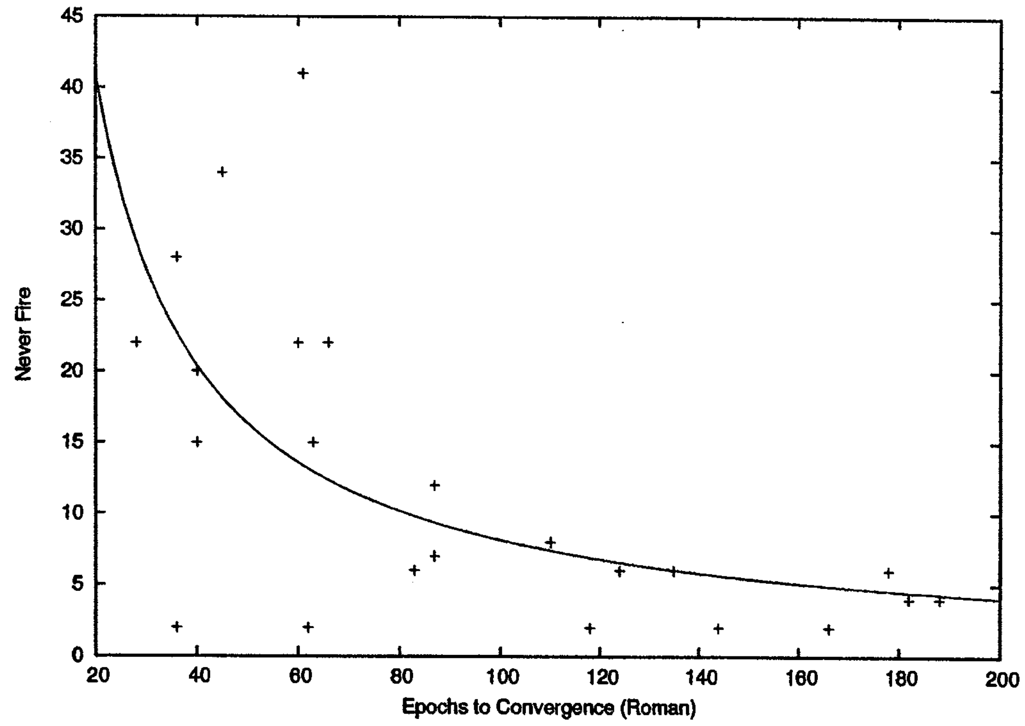


Figure 3: Percentage of DG neuron nonparticipation with 16 DG nodes

Figure 3 shows the fraction of neurons within the dentate gyrus which *never* fire after converging on the Roman alphabet. In other words, these neurons fail to participate in letter identification in any way. Not one of the 26 input patterns elicits any activity from them – they may as well not be there, as far as the letter-recognition task is concerned. Networks that converge quickly have a far greater incidence of these useless neurons.

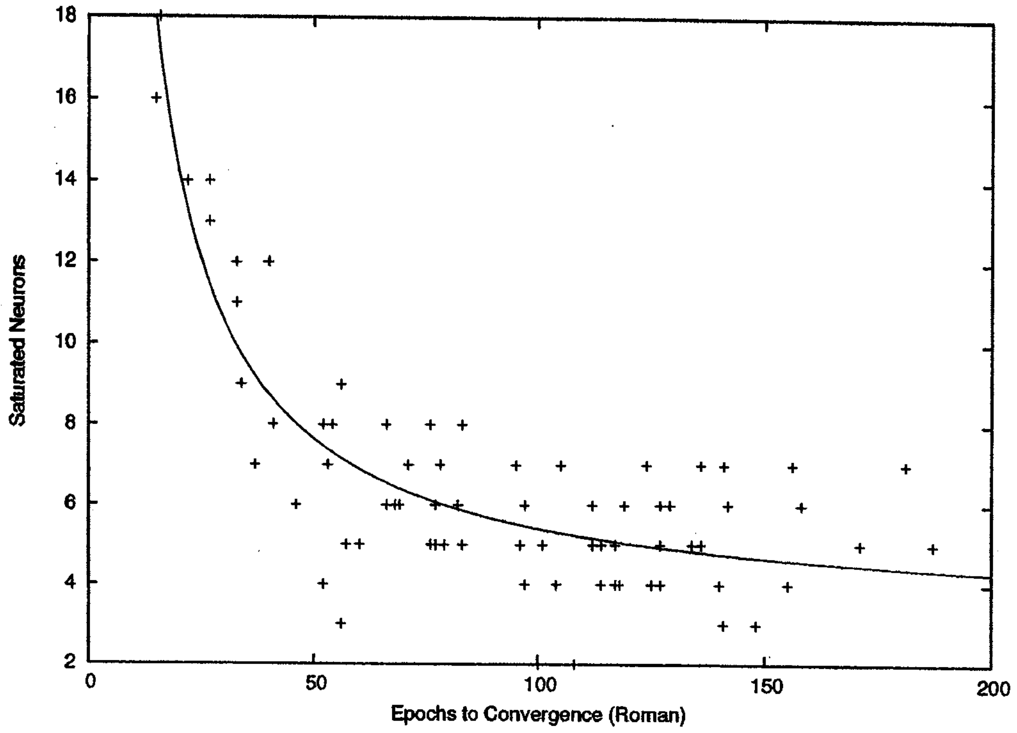


Figure 4: Number of saturated DG neurons after Roman alphabet learning (out of 24 total DG nodes)

Figure 4 shows the number of neurons which are saturated after converging on the Roman alphabet. A saturated neuron is one whose weights exceed the apoptosis threshold, and are thus candidates for potential death. Once again, the same curve emerges. Quick convergence leaves a large number of stressed, saturated neurons in its wake.

The final set of experiments involved allowing the networks to revisit the Roman alphabet after they had mastered Greek. We were interested in the persistence of memory, and whether apoptosis would adversely affect recall, just as it enhances learning new data. This does not seem to be the case. There were no statistically significant differences in the relearning rate, whether apoptosis occurred or didn't, and whether Roman or Greek learning went quickly or slowly.

4. ANALYSIS

What could be happening? It appears that rapidly converging networks happen to start out with a few neurons that, by luck, have weights that help a great deal to differentiate among letters of the Roman alphabet. These neurons tend to dominate the network, quickly elbowing out the competition to the point of quiescence, as shown in figure 3.

Furthermore, the weights of the neurons that do participate become highly saturated (figure 4). Thus, when the alphabet is learned quickly, the network consists of many neurons that are effectively ignored and others which fire often and indiscriminately.

From an information-theoretic perspective, the poor capacity for learning additional information exhibited by these idiot-savant network prodigies comes as no real surprise. A network in which all neurons participate, and where each synapse is weighted differently, can convey an enormous amount of information. A neuron with a well-distributed set of weights has a huge number of possible internal states, and therefore potentially a large amount of entropy (in the information-theoretic sense). The better-distributed the weights are, the closer the entropy H will come to the theoretical maximum $\log(2K+1)$, where K is the number of internal states available to the neuron.

Entropy, of course, translates directly to information content. In the case of our Grecophobic networks, this content is quite low. Many neurons don't participate at all, so their potential to encode information is completely ignored. Furthermore, the neurons that do participate have a very limited internal state space – many of their weights are saturated at the maximum, and so the fact that they fire comes as no big surprise. Low surprise means little information.

Thus networks that slowly converge on the Roman alphabet carefully refine their weights, preserving a wide diversity of possible firing patterns and exploiting the capacity of many more neurons. Such networks are simply smarter – they are able to encode and transmit more information than their fast-acting counterparts, and the evidence suggests that this is why they can learn new information much more quickly.

By preferentially replacing ill-behaved neurons which simultaneously browbeat others into silence and carry little information themselves, apoptosis and neurogenesis help to maintain a network information capacity that is closer to ideal. This might help to explain the puzzling result that apoptosis does not have a negative impact on relearning old information. Whatever information encoded by the neurons destined for elimination, it was never very much to begin with – just enough to provide a competitive advantage at the outset of the convergence process. The absence of these heavyweights is more than made up for by the increased responsiveness and activity of the other neurons in the network.

5. CONTRIBUTIONS

We have uncovered striking patterns in the learning behavior of unsupervised neural networks, suggesting a relationship between the time and effort it takes to learn something and the flexibility and adaptibility of that knowledge once learned. To express it in human terms, it is as if a person who develops an unthinking "knack" for a particular process often has a harder time adapting and applying the process to new circumstances than someone who acquired the skill through dint of careful study. Whether human experience bears this out is debatable.

These findings provide more support and justification for the idea that apoptosis and neurogenesis in the hippocampus promote increased learning ability. We have accounted for differences in learning ability by demonstrating the deleterious effects of both saturated and silenced neurons, effects which can be mitigated through the mechanism of cell death and replacement.

Finally, we have introduced some ideas about the theoretical information capacity of neural networks, as illustrated by the alphabet learning experiments. Hopefully these findings will help in developing procedures and rules of thumb for extracting good performance out of a neural network setup in the future.

References

[1] Chambers, R. A., Potenz, M. N., Hoffman, R. E., Miranker, W. I., (2004). Simulated apoptosis/neurogenesis regulates learning and memory capabilities of adaptive neural networks. *Neuropsychopharmacology*, vol. 29, pp. 747-758.

[2] Erikson, P. S., Perfilieva, E., Bjork-Eriksson, T., Alborn, A. M., Nordburg, C., Peterson, D. A., & Gage, F. H., (1998). Neurogenesis in the adult human hippocampus. *Nature Medicine*, vol. 4, pp. 1313-1317.

[3] Haykin, S., (1999). *Neural Networks: A Comprehensive Foundation.* Upper Saddle River: Prentice Hall.

[4] Miranker, W., (2004). Apoptosis/neurogenesis favorably informs memory development. Yale University DCSTR 1234.

[5] Wirth, S., Yanike, M., Frank, L. M., Smith, A. C., Brown, E. N., & Suzuki, W. A., (2003). Single neurons in the monkey hippocampus and learning of new associations. *Science*, v. 300, pp. 1578-1581.

Modeling Parent-Infant Smiling Interactions Using Neural Networks

Rachel Denison
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract
Appropriate social interaction is a critical ability in everyday human life, and one that is central for machines to have if they are to become fully integrated into human society. The properties of normal human social interactions are investigated using a simple feed-forward neural network trained with backpropagation within the specific social domain of the parent-infant smiling interaction. This network was able to be successfully trained to predict a partner's smile level in several model social interactions, and several of the networks trained in this way were able to perform significantly better in a social interaction with a human partner than a network that was trained with random inputs. These findings suggest that neural networks can be used to test models of the underlying properties of successful social interactions and warrant further investigation in this area. They also suggest new directions for exploring and improving human-machine interactions.
Keywords – neural networks, human-machine interaction, social psychology, infant smiling, affective computing

1. INTRODUCTION

Appropriate social interaction is a critical part of everyday human life, yet it is an activity that is largely ignored in the design of both computer hardware and software. Indeed, a substantial number of computer-users in a recent survey admit to feeling frustrated with, swearing at, and even kicking their computers (MORI survey, 1999). The motivation behind this project, therefore, is both to gain a better understanding of the components of a successful social interaction, and to explore the possibility of designing machines that can interact with humans in a socially appropriate fashion. In order to investigate these questions, a neural network was devised to model a very specific social phenomenon.

This phenomenon is that of the parent-infant smiling interaction, a well-researched area in developmental psychology (Wolff, 1963). This type of early social interaction occurs commonly between a pre-verbal infant and the infant's parent, when the infant and parent are looking at each other. The parent smiles at the infant, the infant smiles back at the parent, the parent's smile grows bigger, and the infant gives a bigger smile in return. This type of interaction, which can be described as a sort of positive smiling feedback loop between the parent and infant, has been shown to be an important one for the development of an attachment between the parent and infant, making it a vital early experience for the social, emotional, and cognitive development of the infant (Bowlby, 1969). The nonverbal nature of this type of interaction makes it a good candidate for an exploratory modeling project using neural networks. The goal of this project, therefore, is to create a network that would model this type of parent-infant smiling interaction.

We create a program that interacts with a human partner by smiling at the partner and receiving smiles from the partner in the same way that an infant interacts with its parent by smiling at its parent and responding when the parent smiles back. The smiling behavior of the program is controlled by a neural network and depends on the smiling behavior of its partner. We train networks to respond to a partner using several experimental models of a social interaction. We then test the trained networks by allowing them to interact with human partners, with the goal of determining which training model best equips the network to interact socially with a human partner.

2. APPROACH AND NETWORK DESIGN

Any number of methods might be used by a human infant to solve the problem of smiling appropriately to an adoring parent. The approach used here was to create a neural net that, given its partner's past smiles (be they big frowns, neutral expressions, or huge grins), would predict the next smile of its partner, and respond to its partner with that degree of smile. The initial thought here is that a network trained to predict, or "expect," a certain model pattern of interaction will make the fewest errors in predicting the course of a real interaction if the course of the real interaction matches the course of the model interaction used to train the network. In other words, if the social interaction model used to train the network results in the network's ability to accurately predict its partner's next smile level, then the model used to train the network may reflect the pattern of human social interactions. We ask, then, which social interaction model tested will best train the neural network to respond to a human partner in a socially appropriate manner.

The neural network used is a simple feed-forward network consisting of an input layer, a hidden layer, and an output layer. The input layer has N+1 neurons, where N is the number of input neurons, which take the past N partner inputs. The final neuron in the input layer represents a bias. The network is trained to output a prediction of the partner's next smile based on the partner's past smiles. The smile level outputted by the network also serves as the network's own smile response to the partner's most recent input. Both the network and the partner are limited to five levels of smiling: big frown, little frown, neutral, little smile, and big smile (Figure 1).

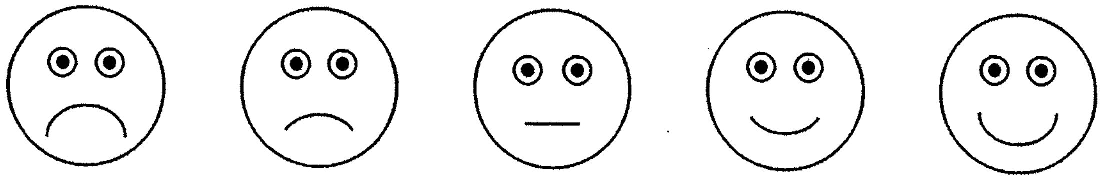


Figure 1. Face pictures showing different levels of smiling used

These were represented in the network by the values [-1, -0.5, 0, 0.5, 1]. Smile levels, then, are an algebraic quantity, in that they have a sign, and they are distributed symmetrically about the neutral face, which has a smile level of 0. The output of the network is transformed by a hyperbolic tangent function to a value between [-1, 1], which, during an interaction, is then scaled to one of the five smile levels by rounding to the nearest smile level, in order to produce a smile level output.

The fully-connected, feed-forward network is trained by backpropagation, as described in Section 3.

3. TRAINING THE NETWORK

3.1 Training Inputs

We train the network using 11 experimental social interaction models. Each model reflects a possible ebb and flow of a dyadic human social interaction, such as the parent-infant smiling interaction, as it develops over time. For example, if a parent and infant look at each other for a long period of time with unchanging neutral expressions, the best model for that interaction would be a long series of neutral smiles. Each model takes the form of a training vector of inputs that are fed to the network. Each training input is one of the five possible smile levels, which are represented on a scale of [-1,1] by the values {-1, -0.5, 0, 0.5, 1}. So, for the unchanging neutral smile example, the model input vector could be {0, 0, 0, 0, 0, 0}, which would repeat over the course of training.

The 11 models used can be divided into four categories. The first category can be termed the *Constant Smile interaction model.* The unchanging neutral smile vector belongs to this category, and this is one of the training input vectors we test. A training vector with constant smile level 1 (big smile) and a training vector with constant smile level -1 (big frown) were also tested. Each constant vector contained 24 0s, 1s, or -1s, depending on whether the vector was designed to model a constantly neutral, constantly smiling, or constantly frowning interaction model, respectively.

The second category can be termed the *Even Wave interaction model.* This model is designed to better reflect the ebb and flow of human social interaction. Each Even Wave model oscillates at an even pace between the values of -1 and 1. Five different "wavelengths" are tested in order to attempt to determine the optimum rate of flow from one emotional level to the next (eg. big smile to little smile) in a social interaction. The model with the shortest wavelength goes directly from one smile level to the next with no repetition. We will say it has a "step-length" of 1. The Even Wave training input vector with step-length 1 is:

```
evenWave1 = {0.5, 1, 0.5, 0, -0.5, -1, -0.5, 0, 0.5, 1, 0.5, 0,
            -0.5, -1, -0.5, 0, 0.5, 1, 0.5, 0, -0.5, -1, -0.5, 0}
```

Likewise, the Even Wave training vector with step-length 3 is:

```
evenWave3 = {0.5, 0.5, 0.5, 1, 1, 1, 0.5, 0.5, 0.5, 0, 0, 0,
            -0.5, -0.5, -0.5, -1, -1, -1, -0.5, -0.5, -0.5, 0, 0,
            0}
```

Input vectors with step-length 5, 7, and 9 are also used in training.

The third category of training input vector used might be termed the *Weighted Wave interaction model.* Two such vectors are used, either weighted toward 1 (big smile) or weighted toward -1 (big frown). The Positive Weighted Wave training vector is:

```
posWeightedWave = {0.5, 0.5, 0.5, 0.5, 1, 1, 1, 1, 1, 0.5, 0.5,
                   0.5, 0.5, 0, 0, 0, -0.5, -0.5, -1, -0.5, -0.5,
                   0, 0, 0}
```

The Negative Weighted Wave training vector is simply the opposite of its positive counterpart – each element of the Positive Weighted Wave training vector is multiplied by -1 to form the Negative Weighted Wave vector.

Each training vector is designed so that it can be repeated seamlessly during the training process. For the purposes of training, this vector represents the smile level responses of the network's partner over time. When training begins, the first N elements of the vector become the N network inputs (the past five responses of the training partner) and element number N+1 becomes the desired network output (the next response of the training partner). Training continues by moving forward one element at a time.

In order to determine the effectiveness of training with a particular model vector, a baseline must be established. So the network is also trained using a fourth category of vectors, representing the *Random interaction model.* This consists simply of a vector of inputs selected at random from [-1,1] – the Random Vector. This vector is generated at the start of a training session and contains the same number of elements as the other training vectors described above. Like the other vectors, the Random Vector cycles through the training session. A sample Random Vector is:

```
randomVector = {0.127349643, -0.669787224, 0.174089899,
0.406058405, 0.96165467, -0.599653321, -0.697442496,
-0.660331357, 0.069319089, -0.950013234, 0.754639832,
-0.960393098, 0.649264477, 0.983325533, 0.967563777, 0.328147145,
0.169444871, -0.577145958, -0.935369808, -0.250663787,
-0.844802401, 0.235151246, -0.843438751, -0.650324565}
```

3.2 Training Method

Training using all vector types consists of backpropagation. Let us recall that the input layer consists of N+1 neurons, taking the past N partner inputs and a bias. In each epoch of training, the network receives N inputs from the training input vector (representing the past N model partner smiles) and a desired output (representing the next model partner smile, partner smile N+1). Based on the current values of its weights, the network calculates an output in response to the inputs. The output of the network can be any real number between [-1,1]. The output of the network is then compared with the desired output, and the error is backpropagated through the network, resulting in changes to the values of the weights between neurons. In the next epoch, the training input vector is cycled forward by one element, such that the earliest smile input is "forgotten" (where v is the vector, v_0 is eliminated and v_n _ v_{n-1}) and the smile that in the previous epoch was the desired output becomes the most recent past partner input (v_{N+1} _ v_N).

It should be noted that there is a very clear critical error value that needs to be met in order to say that the network has been successfully trained. That critical level of error between desired and actual output from the network is 0.25. This is because, during an interaction (but not during training), the output of the network, which can be any number between [-1,1], is converted to a smile level by rounding to the nearest 0.5. This means that for the network to accurately predict a smile level of 0.5, for example, its output can be in the range [0.25,0.75]. Since the range of acceptable values extends 0.25 away from the target output, the critical error value is thus 0.25. Such a relatively large acceptable error is a difference between this and some other neural network applications, in which the difference between the desired and actual output of the network must be miniscule to achieve acceptable performance.

Based on this critical error value, a convergence criterion is set to determine at what point the network is adequately trained. When the convergence criterion is met, training stops, and the network is said to have been successfully trained. We used the following criterion: the network may stop when, over the course of the past input cycle (where the length of one cycle equals the length of the training input vector), 90% of the network outputs have an error smaller than 0.25. In other words, the network may stop when it is predicting the correct smile level at least 90% of the time. The number 90% was chosen based on the observation that in the training of the Random and Weighted Wave vectors, the output of the network generally converged to the desired output, except for periodic error spikes occurring every 24 epochs, the same number of elements in the training input vectors. A convergence criterion of 90% accuracy allows for these spikes while requiring that, 9 times out of 10, the network predict the correct smile level.

In addition to varying the model training vectors used, three different neural network architectures are tested. Each architecture consists of one input layer, one hidden layer, and one output neuron, but the number of input neurons, N+1, is varied: 3, 6, and 10 input neurons are tested. Since one of these neurons always represents a bias, this corresponds to training the network to predict the next partner input based on N = 2, 5, and 9 past partner inputs. In each case, the number of hidden neurons is set to be approximately 70% of the number of input neurons: 2, 4, and 7, respectively. In this way, the network architecture was expanded or shrunk in a scaled fashion in order to test the effectiveness of increasing or decreasing the network's "memory" of past partner inputs.

3.3 Training Results

The 11 training input vectors are tested on the 3 neural network architectures to see if the network will converge to 90% accuracy within 2500 epochs of training. Every training vector is tested on the 6-input architecture, and selected training vectors are tested on the 2- and 10-input architectures. 22 test trainings are carried out in total, and 13 of the 22 trainings result in convergence. The results of training are shown in the table below.

Epochs taken to converge

Input vector	3 Inputs	6 Inputs	10 Inputs
constant1	not tested	25	not tested
constant0	not tested	26	not tested
constant-1	not tested	25	not tested
evenWave1	215	88	46
evenWave3	>2500	478	209
evenWave5	not tested	>2500	486
evenWave7	not tested	>2500	796
evenWave9	not tested	>2500	>2500
posWeightedWave	>2500	968	630
negWeightedWave	not tested	974	not tested
randomVector	>2500	>2500	>2500

Table 1. Training summary showing the number of epochs each network architecture took to converge when trained using the various experimental input vectors

3.3.1 Convergence variations in training with Even Wave vectors

One interesting finding from the training results is that, when trained with the Even Wave vectors, the 3 neural architectures had different convergence results. The 3-input architecture converge successfully only for the Even Wave with step-length 1, the 6-input architecture converge for step-lengths 1 and 3, and the 10-input architecture converge for step-lengths 1, 3, 5, and 7, but not for step-length 9. The step-lengths on which the networks fail to converge bear an interesting relationship to their architectures. The networks only converge when the step-length of the training vector is less than the number of past smiles available to the network. Since one input neuron is a bias, the number of past smiles available is always one less than the number of input neurons. In each case, a network with N+1 inputs (+1 being the bias) fails to converge because of a regular error made when presented with an Even Wave vector with step-length greater than or equal to N. The network is able to make the correct prediction most of the time, but every Nth epoch the network makes an error. As can be seen in Figure 2, for example, the pink series 2 line, which represents the error, spikes every 5th epoch, indicating the occurrence of an error in the output of the network.

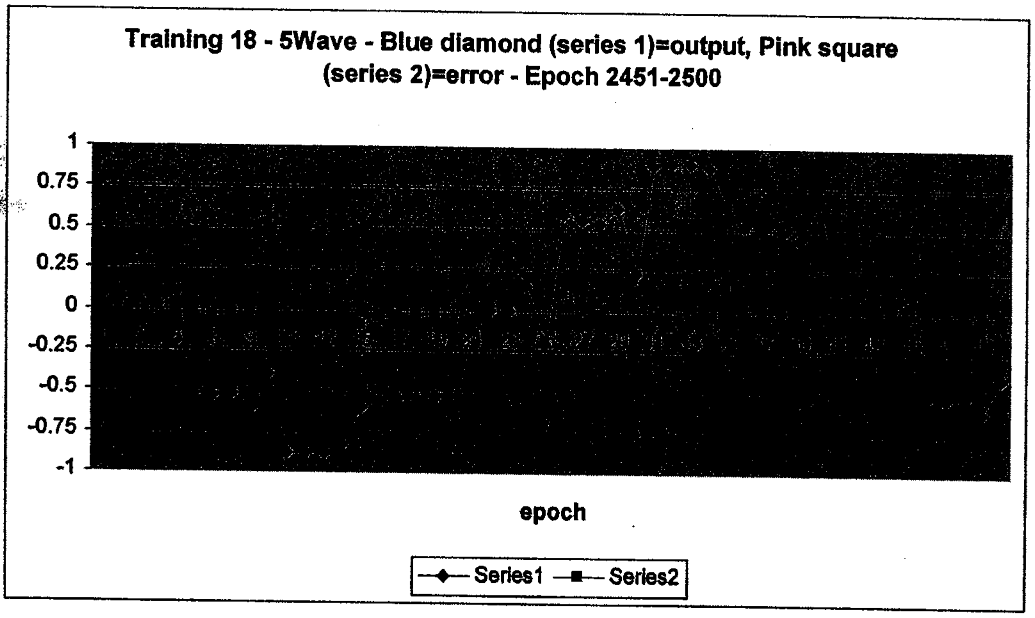


Figure 2. Plot of progress of training with 6-input architecture using Even Wave training vector with step-length 5, epochs 2451-2500. Shows error every 5th epoch

Additionally, the different architectures converge at different rates when trained with the same vector. The difference can be seen clearly in the networks' rates of convergence for the Even Wave vectors. As shown in the following graph, the more inputs the network has, the faster it converges. Additionally, the longer the step-length of the wave, the longer the network takes to converge.

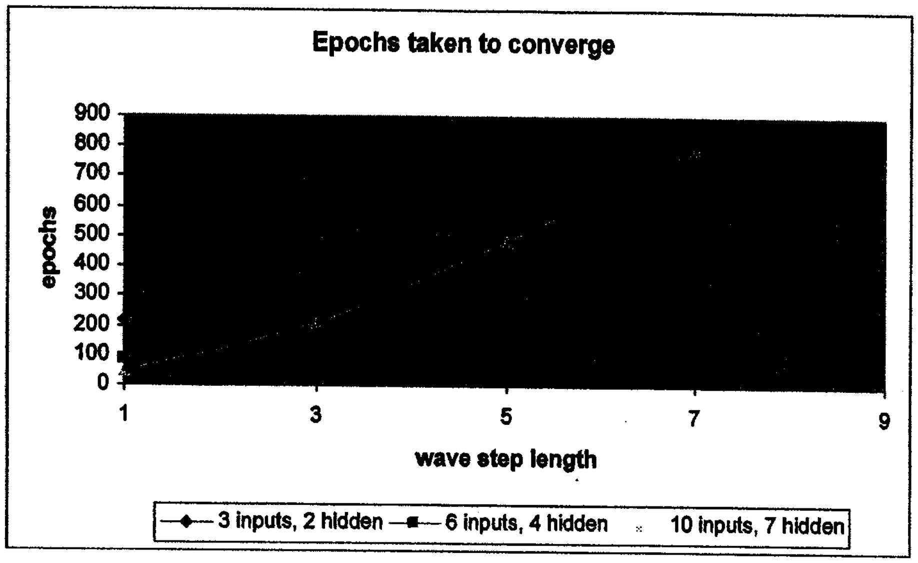


Figure 3. Plot of the number of epochs each of three networks took to converge when trained using Even Wave vectors of step-length 1, 3, 5, 7, and 9

In Figure 3, no point is plotted if the network does not converge within 2500 epochs of training. This occurs when step-length equals 3 for the 3-input architecture, 5 for the 6-input architecture, and 9 for the 10-input architecture, as discussed above. Here we can propose, based on the above graph, that this is not simply a case of too few iterations, but is instead something of a breaking point in the network's ability to learn to predict the next input. If we were to extrapolate from the graph and predict how many epochs it might take for the 10-input network to converge to a wave with step-length 9, we might put the estimate somewhere between 1200 and 1500 epochs. However, in testing, the network does not converge even after 2500 iterations, suggesting the possibility that the network may not be able to learn to predict the next input when the step-length of the wave is greater than or equal to the number of past inputs to which it has access.

3.3.2 Convergence failures in training with Random Vector

Training with the Random Vector does not result in convergence for any of the networks. Since the network is designed to predict the next smile level given the five previous smile levels, it is expected that an attempt to train the network to predict a random number based on five other random numbers would be unsuccessful, and that expectation is borne out here.

It is important to clarify that the random case is not akin to "real life." Social interactions are not random – just the opposite. We do not expect Alice to punch Bob in the face after Bob gives Alice a present, for example. More in line with the paradigm used here, we do not expect Alice to frown when Bob smiles at her. A goal in this study is to try to uncover just what the underlying patterns of social interaction are. Training the network with the Random Vector provides a good baseline against which to compare performances of other training vectors *because* it presupposes no underlying form to social interactions. We suggest that the better other training vectors perform compared to the baseline, the better they represent the underlying pattern of the social interaction.

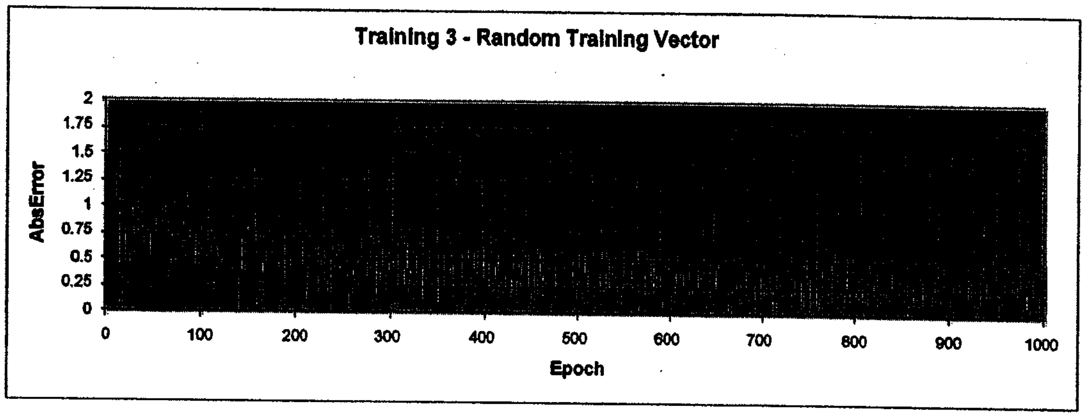


Figure 4. Plot of progress of training using Random Vector as training inputs, 6-input network

3.3.3 Patterns of error over the time-course of trainings that result in convergence

For those trainings that resulted in convergence, several different patterns of error over the course of training were observed. In general, the error decreased to a point where it was below 0.25 more than 90% of the time, but the error seen over the course of training varied. For a network trained with a Constant input vector, for example, the error decreased quickly and steadily over the course of training, while for the Weighted Wave vector, the error was more erratic. Plots of the absolute value of the error over the time course of training for these three sample training runs are:

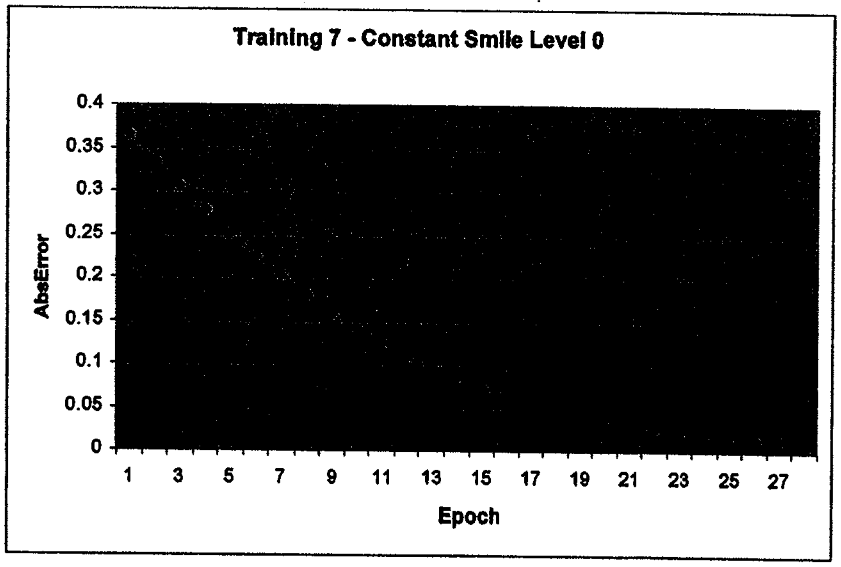


Figure 5. Plot of progress of training using Constant 0 vector as input, 6-input network

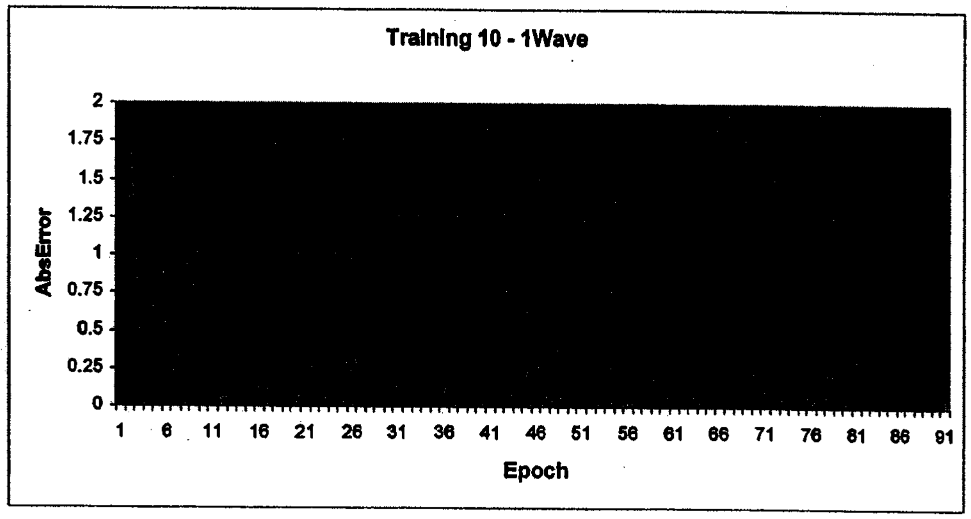


Figure 6. Plot of progress of training using Even Wave vector with step-length 1 as input, 6-input network

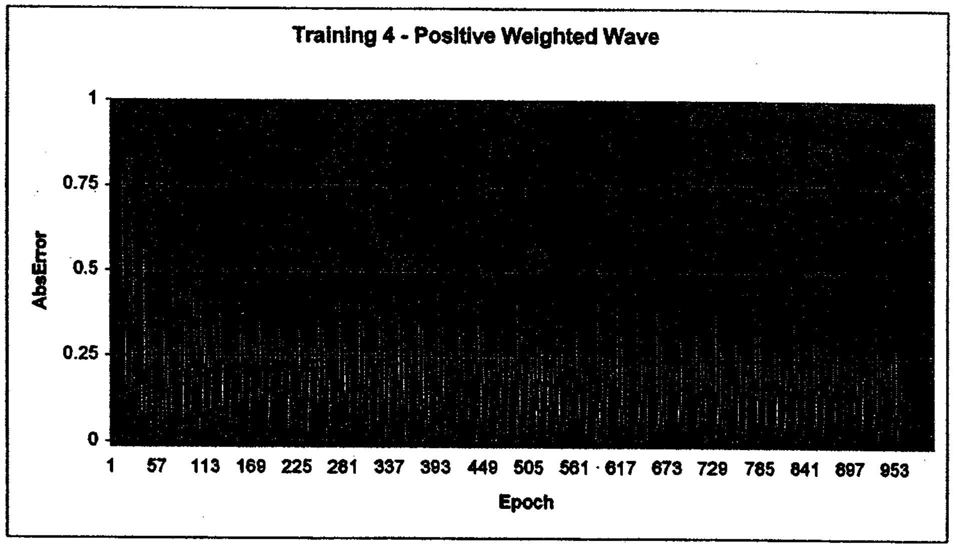


Figure 7. Plot of progress of training using Positive Weighted Wave vector as input, 6-input network

As can be seen in the above graph, periodic error spikes occur every 24 epochs when the network is trained with the Positive Weighted Wave vector. That input vector has 24 elements, and the network spikes at the same position in each cycle of the input vector. Interestingly, it does not spike at the beginning or end of the vector, but at vector position 14, a 0 following four 5s in the Weighted Wave. This is the case for the 3-input, 6-input, and 10-input network when trained with the Positive Weighted Wave. The same pattern is also seen when the 6-input vector is trained with the Negative Weighted Wave – the error spike still occurs at the 14th input vector position. In order to evaluate if there is something "special" about the 14th position in general or only in the Weighted Wave, we can look at the training errors for the Random Vectors, which also produce error spikes every 24 epochs. When the 6-input network is trained with the Random Vector, the 14th position error is seen again. However, for the 10-input network, the error occurs at position 3, and for the 3-input network, the error occurs at position 18, suggesting that although the periodic spiking is seen, the error spike may appear at any position for a network trained with a Random Vector, but seems to regularly appear in the 14th position for all vectors trained on the Positive or Negative Weighted Wave. The reason why the error appears periodically in the 14th position in these Weighted Waves as opposed to in some other position is not resolved here, but bears further investigation.

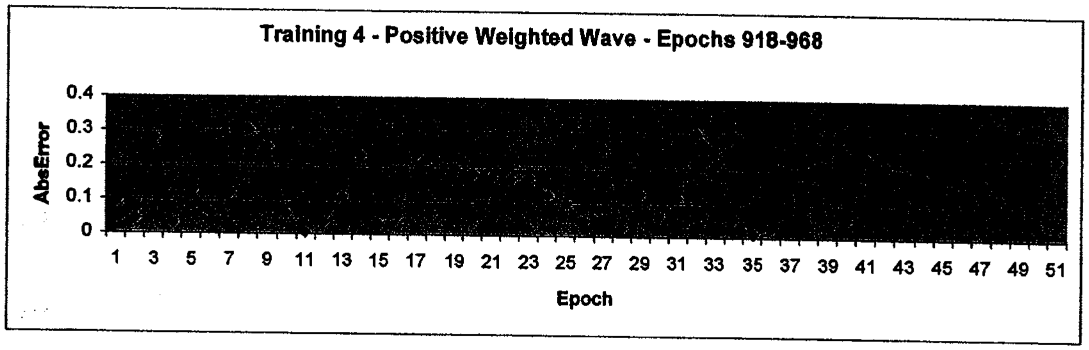


Figure 8. Plot showing the absolute value of the error of the 6-input network over the final 50 epochs of training with the Positive Weighted Wave training vector

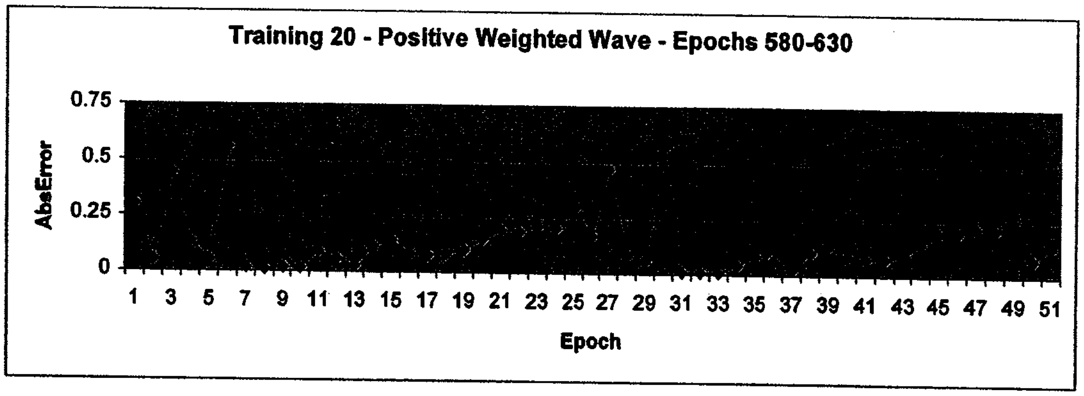


Figure 9. Plot showing the absolute value of the error of the 10-input network over the final 50 epochs of training with the Positive Weighted Wave training vector. Although the error between spikes is more stable for this 10-input network as compared with the 6-input network above, the error still spikes periodically every 24 epochs at the 14th input vector position

As seen above (Figures 5 and 6), other types of input vectors do not show an error spike with period 24. The error of the network trained with Even Wave with step-length

1 oscillates fairly regularly, peaking every four epochs (Figure 6) as a result of not quite reaching the extreme values of -1 and 1 in its output. The error of the network trained with Constant 0 is not at all periodic, but steadily decreases (Figure 5). This suggests the possibility that the irregularity within both the Weighted Wave vectors and the Random Vector forces the network to chunk the network into its smallest repeating segments – in this case, the entire 24-element input vector – in order to learn to predict the next element. How it might be doing this, if indeed it is, is an area requiring further investigation.

4. TESTING THE TRAINED NETWORK

4.1 Method

Since this network is designed to interact with a human partner, its performance must be evaluated by the way it interacts with a real person in real time. In order to test the network trained on the model interaction vectors and compare it with the network trained on the Random Vector, we develop an interface that enables the neural network to interact with a human partner in real time.

The human partner can "smile" at the network by selecting a face with one of five "smile levels" on a graphical user interface. The network, in turn, responds back to its partner's selected smile by displaying a face with one of the same five smile levels. This back-and-forth smiling continues between the program and its human partner just as it might between an infant and its parent. Each testing session consists of 50 epochs of alternating smiles, for 100 smiles total. The smile level that the program chooses is based on its partner's past smile levels. During the interaction, these past smile levels are fed into a trained neural network. The output of the network is a number corresponding to one of the five smile levels. The corresponding smile level is then displayed to the partner graphically.

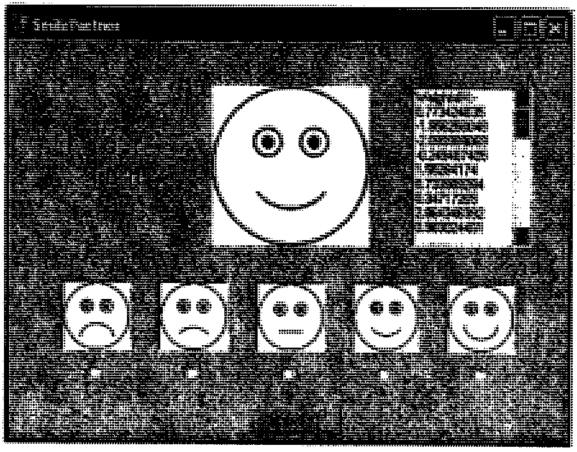


Figure 10. Graphical user interface used to test performance of the neural network when interacting with a human participant

Three human subjects, who did not know anything about the project, were tested on 16 networks using this interface, which had been trained by the various experimental training input vectors on the 3 experimental network architectures.

Properties of trained networks used in human testing

Training	Input vector	Number of inputs	Number of hidden	Epochs trained
3	randomVector	6	4	2500
4	posWeightedWave	6	4	968
5	constant1	6	4	25
6	constant-1	6	4	25
7	constant0	6	4	26
8	negWeightedWave	6	4	974
9	evenWave3	6	4	478
10	evenWave1	6	4	88
14	evenWave5	10	7	486
15	evenWave7	10	7	796
16	evenWave3	10	7	209
17	evenWave1	10	7	46
20	posWeightedWave	10	7	630
24	evenWave1	3	2	215
26	randomVector	3	2	2500
27	randomVector	10	7	2500

Table 2. Networks trained with these 16 input vector-architecture combinations were tested with human subjects.

The subjects were given instructions to interact with the program by trying to make friends with the face on the screen and respond to it as they would to a baby. The 16 test networks were presented to each subject in random order. During the interaction, the responses of the network and the person, and the errors between the network's output and the person's actual subsequent response, were recorded. Following the interaction, the subject was asked to rate his or her experience with the network on a scale of 1 to 10, 1 being "the worst social experience with a baby imaginable, completely unsatisfying" and 10 being "the best social experience with a baby imaginable, incredibly satisfying."

The performance of the network for each training condition was evaluated using two measures: 1) the error of the network, that is, the difference between the output of the network and the subsequent output of the human subject (measure used was root mean square error (RMSE)) and 2) the subjective experience rating given by the subject.

4.2 Results

The results of human testing are not as consistent as we might have hoped, although some clear trends do emerge. The tables and graphs below show the RMSE for three subjects and the subjective experience ratings for two subjects (one subject did not give ratings) who interacted with all 16 test networks.

RMSE

Training	Input vector	Num inputs	S1	S2	S3	Mean
26	randomVector	3	5.4	5.1	6.9	5.3
24	evenWave1	3	4.2	2.3	3.6	3.3
3	randomVector	6	6.2	3.1	4.2	4.7
5	constant1	6	4.1	4.0	3.0	4.1
7	consant0	6	4.1	2.2	4.1	3.2
6	constant-1	6	5.8	3.6	5.8	4.7
10	evenWave1	6	2.9	2.4	2.9	2.7
9	evenWave3	6	4.1	1.5	2.4	2.8
4	posWeightedWave	6	4.2	1.9	3.1	3.1
8	negWeightedWave	6	4.8	2.8	3.3	3.8
27	randomVector	10	7.6	2.4	3.8	5.0
17	evenWave1	10	4.4	4.8	6.5	4.6
16	evenWave3	10	4.3	2.1	2.4	3.2
14	evenWave5	10	4.1	3.2	3.2	3.7
15	evenWave7	10	3.9	4.8	3.6	4.4
20	posWeightedWave	10	4.8	4.7	3.2	4.8

Table 3. Summary of RMSE results for three subjects who interacted with the 16 test networks.

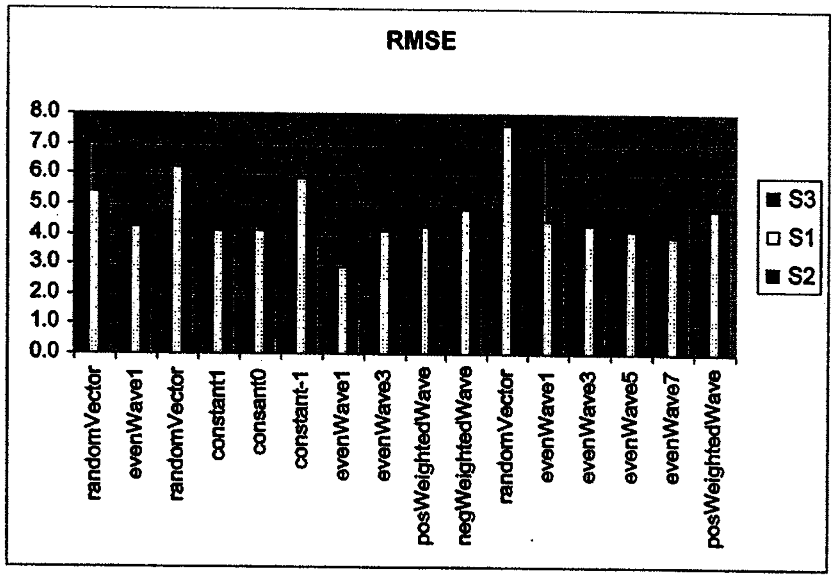


Figure 11. RMSE results for three subjects. Items grouped by training type (labeled) and by network type, so that the first two clusters correspond to the 3-input network, the next eight to the 6-input network, and the final six clusters to the 10-input network.

Subjective Experience Rating

Training	Input vector	Num inputs	S1	S2	Mean
26	randomVector	3	6	7.75	6.875
24	evenWave1	3	5	6.75	5.875
3	randomVector	6	7	4	5.5
5	constant1	6	5	7.5	6.25
7	consant0	6	3	8.5	5.75
6	constant-1	6	5	9	7
10	evenWave1	6	3	6	4.5
9	evenWave3	6	7	5.5	6.25
4	posWeightedWave	6	6	9.5	7.75
8	negWeightedWave	6	6	8	7
27	randomVector	10	5	8.5	6.75
17	evenWave1	10	7	7.5	7.25
16	evenWave3	10	6	6.75	6.375
14	evenWave5	10	6	7	6.5
15	evenWave7	10	7	7	7
20	posWeightedWave	10	8	7.5	7.75

Table 4. Summary of RMSE results for three subjects who interacted with the 16 test networks.

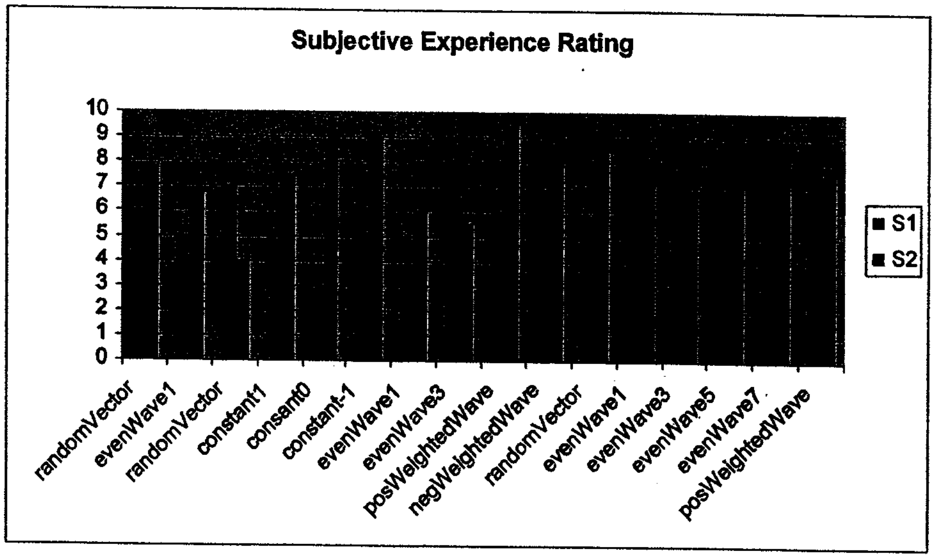


Figure 12. Subjective experience ratings given by two subjects after interacting with each test network. Items grouped by training type (labeled) and by network type, so that the first two clusters correspond to the 3-input network, the next eight to the 6-input network, and the final six clusters to the 10-input network.

Please see Appendix A for graphs showing the mean RMSE and subjective experience rating results for each network.

4.2.1 Performance of model-trained compared to randomly trained networks

One clear finding is that for all three network architectures (3-, 6-, and 10-input) every model interaction vector outperformed the Random Vector as measured by mean RMSE (See Figures 15, 17, and 19), so that in every case, training the network with some non-random interaction model enabled the network to better predict its partner's next smile level than training with a random vector. The subjective experience rating results were less straightforward, with some test networks performing better than the randomly trained network in this measure and some performing worse (See Figures 16, 18, and 20).

For the 3-input network, 0 of 1 model-trained networks outperformed the randomly trained network in both measures. For the 6-input network, 5 of 7 model-trained networks outperformed the randomly trained network in both measures. These were: Constant 1, Constant 0, Even Wave step-length 3, Positive Weighted Wave, and Negative Weighted Wave. For the 10-input network, 3 of 5 model-trained networks outperformed the randomly trained network in both measures: Even Wave step-length 1, Even Wave step-length 7, and Positive Weighted Wave. Of the 3 models tested on multiple networks (Even Wave step-length 1, Even Wave step-length 3, and Positive Weighted Wave), only Positive Weighted Wave-trained networks outperformed the randomly trained networks in all tests.

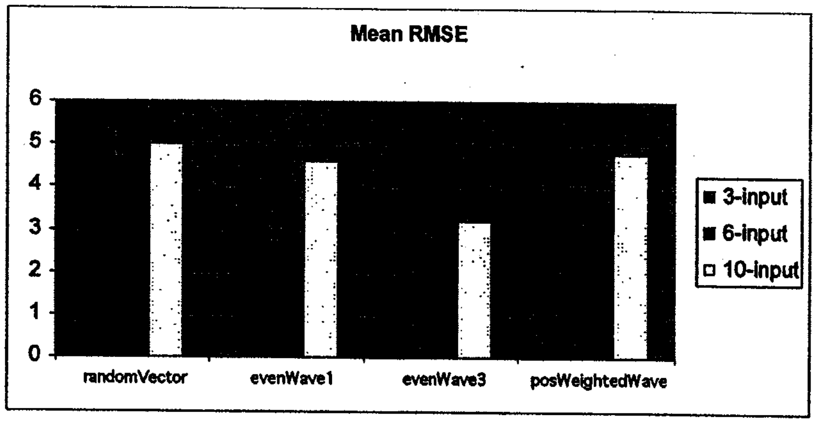


Figure 13. Mean RMSE results comparing the performance of the three different architectures (3-, 6-, and 10-input networks). Two trainings were given to all three network types and four trainings were given to two of the network types

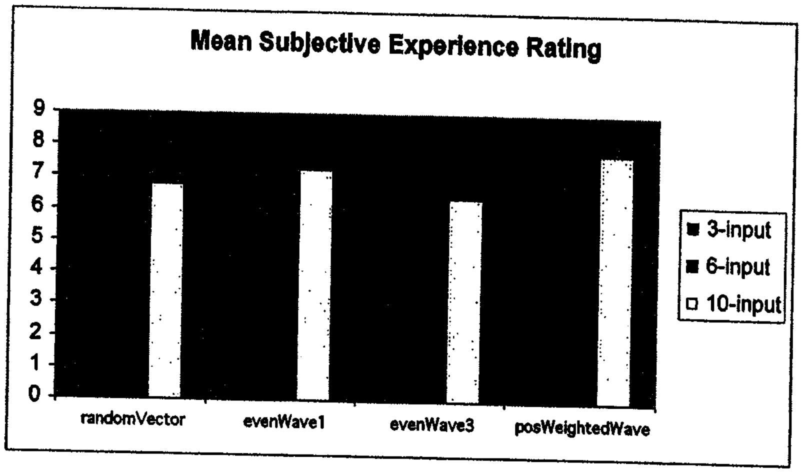


Figure 14. Mean subjective experience rating results comparing the performance of the three different architectures.

4.2.2 Comparisons of network architectures

Both the 3-input and the 10-input architectures consistently perform worse than the 6-input architecture for the trainings they have in common, as measured by mean RMSE. This suggests that 2 past partner smiles may be too few to remember to make an accurate prediction about the partner's next smile, while 9 may be too many, making the optimum somewhere around 5. However, the 6-input architecture receives lower subjective ratings from participants than the other two architectures for trainings with the Random Vector and Even Wave step-length 1. However, the RMSE may be a more reliable measure, as evaluated by the consistency of results for the random trainings. A randomly trained network should be expected to show approximately consistent performance regardless of its number of inputs: the range of mean RMSE values for the three architectures is only 0.6, while the range of mean subjective rating values is 1.4, suggesting greater baseline variability in the subjective experience ratings than in the RMSE measures. More human subjects should be tested in order to reduce the variance in both measures and achieve more reliable results.

In the mean RMSE measure, the two best-performing networks for all three network architectures were Even Wave-trained networks. Specifically, for the 6-input network, the networks with the lowest mean RMSE were trained by Even Wave step-length 1 and Even Wave step-length 3. Even Wave step-length 1 had one of the highest mean RMSEs for the 10-input network; Even Wave step-lengths 3 and 5 were the best performers. (The 3-input network was only tested on one model vector, Even Wave step-length 1, making that the obvious best-performer for that architecture.) The difference in performance in Even Wave step-length 1 for the 10- and 6-input architectures highlights once again that the number of past inputs the network takes into account when interacting with a human participant is not irrelevant, but can result in dramatically different results for different architectures. The overall high performance of the Even Wave vectors suggests that an oscillating interaction pattern may be a natural one, and the consistently good performance of the Even Wave step-length 3 suggests that the length of time required for a participant to smile three times at the network may best approximate the time a human interaction is most likely to stay at one emotional level. One way this could begin to be tested by measuring the time-course of a parent-infant smiling interaction.

4.2.3 Subjective experience ratings and comments

The subjective experience rating results are often at odds with the RMSE results. In fact, the two measures frequently suggest opposite conclusions. The relationships between the RMSE values for a given architecture are almost always very similar to the relationships between the subjective experience values, meaning that, for example, Even Wave step-length 3 has both a lower mean error *and* a lower mean rating than Even Wave step-length 5 for the 10-input architecture. The error measure in this example suggests that the network trained with step-length 3 performs better than that trained with step-length 5, while the rating measure (lower rating for 3) suggests that it performs worse. These differences reinforce the idea that the RMSE and subjective experience ratings are very different measures, and more subjects as well as further investigation will be needed both to confirm and to explain these results.

The comments made by the human participants after each trial may be most illuminating in this respect, and may help to guide further research.

Training	Training input, architecture	S1	S2
3	randomVector, 6-input	perverse at first, became erratic, confused understanding of emotion	bad baby, affect did not match my affect
4	PosWeightedWave, 6-input	good baby, easy to make happy, sort of boring	liked the baby, responsive, did not smile unreasonably, seemed like nice real baby
5	Constant 1, 6-input	[no comment]	[no comment]
6	Constant -1, 6-input	manic or self-absorbed	too happy, possibly brain damaged
7	Constant 0, 6-input	wimpy, sad	felt could cheer it up, needed to be cheered up from time to time
8	NegWeightedWave, 6-input	[participant tried to confuse the baby]	fussy in an understandable way, able to be comforted
9	EvenWave3, 6-input	realistic and happy	high maintenance, ended with big frown twice!
10	EvenWave1, 6-input	figured it out, seemed programmed	discontented, unstable
14	EvenWave5, 10-input	mimicked but no personality	cycled quickly, but generally happy
15	EvenWave7, 10-input	realistic	average baby
16	EvenWave3, 10-input	liked but pitied it, as it lacked confidence, hesitant	slightly fussy

17	EvenWave1, 10-input	did exactly what I did	happy baby
20	posWeightedWave, 10-input	seemed to know that I was just joking around	pretty good baby
24	EvenWave1, 3-input	neutral or unpredictable, possibly not fully engaged	its mood cycled quickly, so required more energy, exhausting, seldom stayed the same!
26	randomVector, 3-input	was sad	[no comment]
27	randomVector, 10-input	impish, mean, perverse, delighted in my misery	[no comment]

Table 5. Comments made by two human participants after interacting with each of the 16 test networks.

These comments show that the human participants readily viewed the network interface as a baby while they were interacting with it. This can be seen by the way the participants attribute very specific personalities to each network. The variability of the responses, and, more significantly, the similarity between the responses the two participants made for each network, suggests that each training produced a unique and specific type of "baby." For example, both participants commented that the training 3 network, trained with a random input vector was socially inappropriate in the way that it responded to their smiles (S1: "perverse at first, became erratic, confused understanding of emotion;" S2: "bad baby, affect did not match my affect"). The strong emotional valence of these comments suggest that the human participants are reacting to the network as if it were a social being, indicating that this interactive tool may be an appropriate one to use in further study.

5: CONCLUSIONS

The results of this study point to some intriguing directions for further research. The smile interaction neural network is a novel tool that can be successfully trained to predict a partner's next input using model interaction training input vectors. Training on the three experimental architectures (3-input, 6-input, 10-input) results in convergence for 10 of the 11 training vectors tested, with the Random Vector being the expected exception. The ability of the network to be trained at all in this way shows that a neural network can be trained to predict patterned time-dependent sequences, and the convergence failures for the three network architectures when presented with an Even Wave vector that had a step-length greater than or equal to the number of inputs to the network shows that this predictive ability breaks down in a regular way, a property which may warrant further investigation.

The preliminary results of human subject testing suggest that the 6-input architecture outperforms both the 3- and 10-input architecture. This could point toward a possible "memory" parameter for the natural pattern of human interactions, particularly nonverbal ones, in which it might be optimal to keep in mind the past several seconds of the interaction – where a shorter "memory" might cause disorientation, and a longer one, distraction. This is a prediction that could be tested with human subjects in order to further evaluate the usefulness of this neural network for modeling social interactions. The best-performing training vectors overall are the Even Wave step-length 3 vector, which gave low errors for both the 6-input and 10-input networks, and the Positive Weighted Wave vector, which outperformed the Random Vector in both the error measure and the rating measure on both the 6-input and 10-input networks. These findings suggest that, of the models tested here, a social interaction model that oscillates over a range of emotions over time, and that spends an average of 3 "units" (here, epochs) at each emotional valence, best represents the time-course of the smiling interaction. This is another prediction made by the network which can be tested on human subjects.

The subjective experience ratings do not always agree with the error measures, which could suggest that a human participant may not always find it agreeable to have a "predictable" interaction. More human subjects are needed to substantiate and further evaluate this finding, as well, perhaps, as more well-defined criteria for rating the experience interacting with the network. Indeed, the comments made by the human participants may prove more illuminating than their numerical ratings. They suggest some level of consistency and agreement in the subjective view of the performance of each trained network, which suggests that training the network using the prediction method might indeed be an effective way to give a specific "character" to a social interaction network.

It is important to note that this is not a proposal that infants actually predict the parent's next smile in order to respond appropriately to the parent. But however they do accomplish the task, it is interesting that a model as simple as a 3-layer, 3-to10-input neural net making predictions based on extremely simple models of a social interaction over time can produce behavior that is significantly closer to being socially appropriate than the behavior produced when a random vector is used to determine the behavior of the network. Just as the early infant smiling interaction with a parent serves the baby as a building block for learning more sophisticated social behavior (not least by making the baby an attractive social partner, so it will gain more social experience from which to learn), a simple implementation of the basic pattern of social-emotional interaction like the network described here could serve as an initial building block when trying to create a machine that can respond successfully in more complex and refined social situations.

REFERENCES

1. Employees get 'it' out of their systems. (1999). Retrieved electronically 7 Dec. 2004 from <http://www.mori.com/polls/1999/rage.shtml>.
2. Wolff, P.H. (1963). Observations on the early development of smiling. In B. M. Foss (Ed.), *Determinants of infant behavior, Vol. 2*. London: Methuen.
3. Bowlby, J. (1969). *Attachment and loss, Vol. 1*: Attachment. London: Hogarth Press and the Institute of Psycho-Analysis.
4. Haykin, S. (1999). *Neural networks: A comprehensive foundation, 2nd ed.* New Jersey: Prentice Hall.

APPENDICES

A. Results of testing with human participants

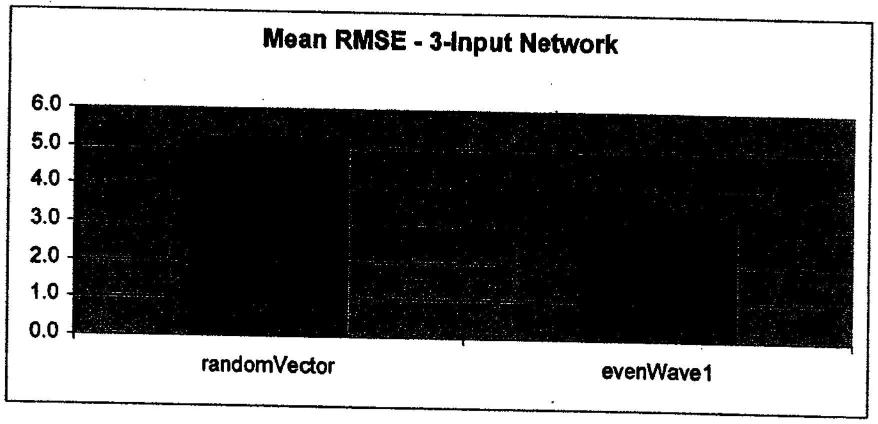


Figure 15. Mean RMSE results for 3-input network

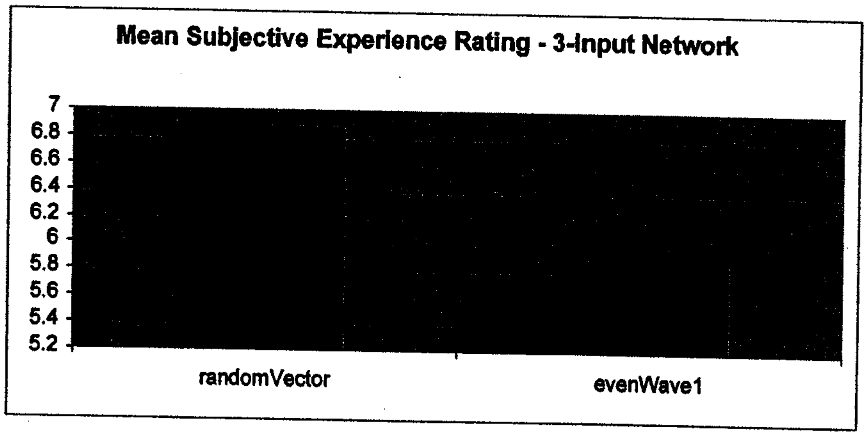


Figure 16. Mean subjective experience rating results for 3-input network

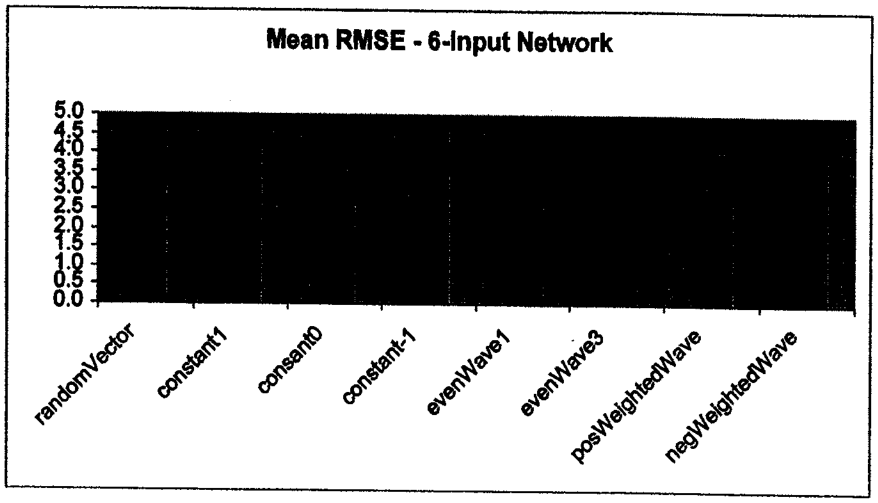


Figure 17. Mean RMSE results for 6-input network

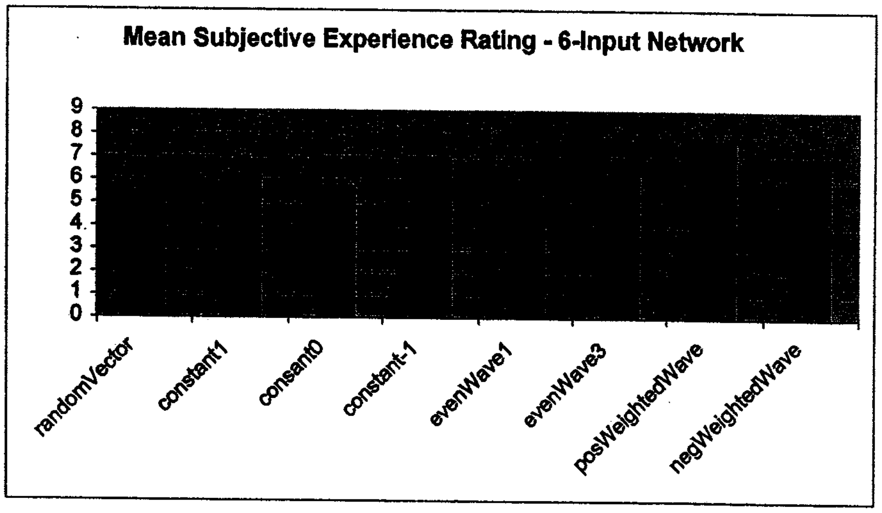


Figure 18. Mean subjective experience rating results for 6-input network

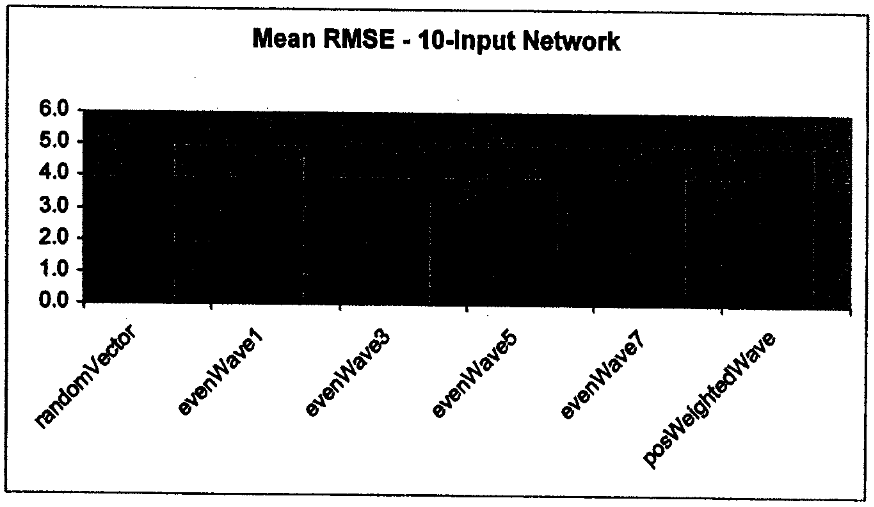


Figure 19. Mean RMSE results for 10-input network

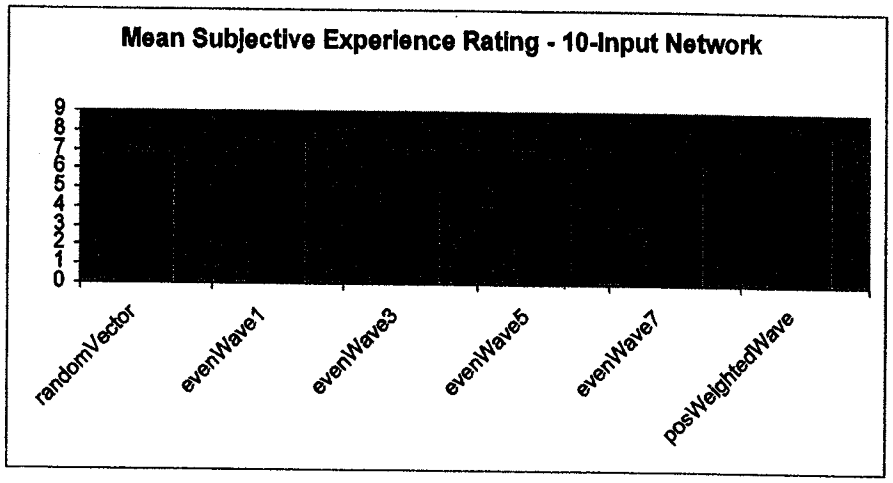


Figure 20. Mean subjective experience rating results for 10-input network

A Generic Multilayer Perceptron for Customizable Digital Neural Network Configurations in VLSI

Sam Evers
Yale University
New Haven, CT 06510

Abstract

A digital multilayer perceptron neuron is proposed and taken through design and layout using the Cadence CAD tool. The neuron is intended for use in a customizable hardware neural network package. This represents the first step towards a toolkit which will allow the end user to create a custom NN, comprised of multiple instances of the proposed neuron along with a custom designed controller and clock scheme. Only the generic neuron is designed; however, the other components are still discussed in detail.

1. INTRODUCTION

Artificial neural networks (ANN) used to exist primarily in the academic realm. Today, they have applications in countless fields. Anheuser Busch uses neural networks to predict the organic content in their competitors' beer vapors, financial institutions use them in trading models, and businesses are performing job placement with the help of neural network classifiers. Despite the relative proliferation of neural networks, small businesses and individual users often lack the resources necessary to either hire an expert or train themselves to implement them. In order to alleviate this problem I am exploring the hardware implementation of a generic, customizable neural network that can adapt to numerous uses.

Multilayer perceptron (MLP) networks have uses in classification, pattern recognition, and feature recognition. By adding more layers to the network, higher orders of information can be extracted from the input data. This means that MLPs are capable of solving non-linear problems with high degrees of complexity. Networks are often trained using the back-propagation algorithm, which updates the synaptic weights starting at the output layer and progressively working its way back towards the inputs. Each neuron is identical in its make up to all other neurons in the network. There is a slight difference, however, in the training algorithm for hidden and output neurons. This level of modularity lends itself to the design of a single generic neuron module. Multiple instances of the neuron can be placed in a fully connected feed-forward configuration to create a custom neural network of the user's choosing.

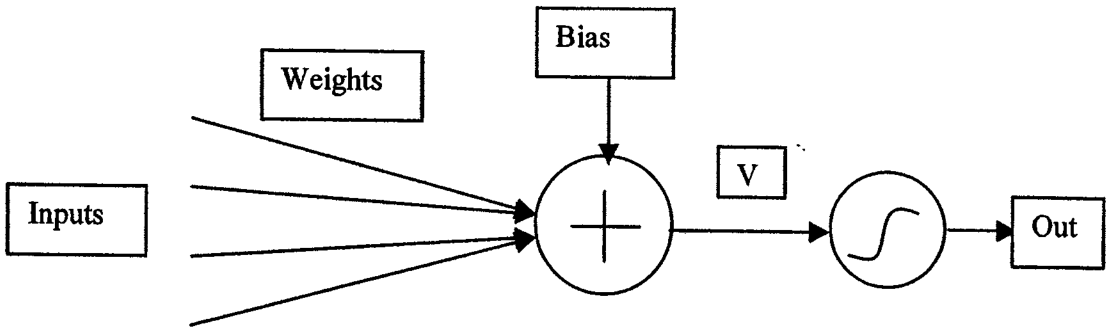
2. NEURON BASICS

Layout and design of the neuron is done using the Cadence VLSI software package. The layout is tailored to the AMI 0.5 μm process and can be fabricated in standard CMOS technology. This means that the transistors can be made with channel lengths as short as 0.5 μm. Each artificial neuron is implemented with four available inputs and acts as either a hidden or an output node based on the binary status signal from an on-chip control unit. The controller contains information describing the unique design of the network and sends out control signals to the individual neurons, telling them whether they are hidden or output nodes and whether they are in the forward or backward computation phase for training.

Aside from the small control unit and its associated memory bank, each neuron is self-contained. Since the back-propagation algorithm requires hidden nodes to have access to some of the calculations made in the previous layer, the controller's memory bank serves as a repository for the data that must be shared between neurons. The three major components of the neuron design are an arithmetic logic unit (ALU), a controller, and memory; meaning that each module is essentially a small microprocessor. The advantage of this design choice is that the size of each neuron can be kept relatively small. The drawback is a decrease in computational speed. A single ALU can only perform one computation at a time, which means that the inputs and synaptic weights cannot be multiplied in parallel. This problem could be solved by the inclusion an ALU for each of the neuron's four inputs, but the size of each module would become prohibitively large. The inherently parallel nature of neural networks is retained on a larger level though. Every node of a given layer can perform its computations simultaneously as long as conflicts are avoided when multiple modules attempt to access the shared memory simultaneously.

The architecture of an individual artificial MLP node will now be examined in more detail. Figure 1 contains a diagram of the neuron, with the equation for the sigmoid activation function below it (1). Inputs are multiplied by their associated synaptic weights and the products are added together to create an induced local field at the input to the neuron's processing element. The induced local field is evaluated by an activation function (i.e. sigmoid) to produce the neuron's output. Due to the nature of digital circuits, computing exp(-v) is not a straightforward task. One way to compute exponents is to use Newton's Method. For this project, a less complex second order approximation is used (2), where *v* is the input.

Figure 1. (Inputs are multiplied by the synaptic weights and summed, along with a bias, to create the input to a sigmoid function.)



$$Out = \frac{1}{1+e^{(-v)}} \quad (1)$$

$$Out = \begin{cases} 1-2^{-1}(1-|2^{-2}v|)^2 & 0<v<4 \\ 2^{-1}(1-|2^{-2}v|)^2 & -4<v<0 \\ 1 & v>4 \\ 0 & v<-4 \end{cases} \quad (2)$$

Before the network can perform its function, it must be trained using a training data set. This is a set of input vectors already associated with the desired output values of the network. The NN is stimulated with a set of inputs which propagate forward to the output. The output of the network is then compared with the desired output value, which the back-propagation algorithm uses to update the NN's synaptic weights. The back-propagation algorithm's equations are shown below.

$$e_j(n) = d_j(n) - o_j(n) \quad (3)$$

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)) & j = output \\ \varphi_j'(v_j^{(l)}(n))\sum_k \delta_k^{(l+1)}(n)w_{kj}^{(l+1)}(n) & j = hidden \end{cases} \quad (4)$$

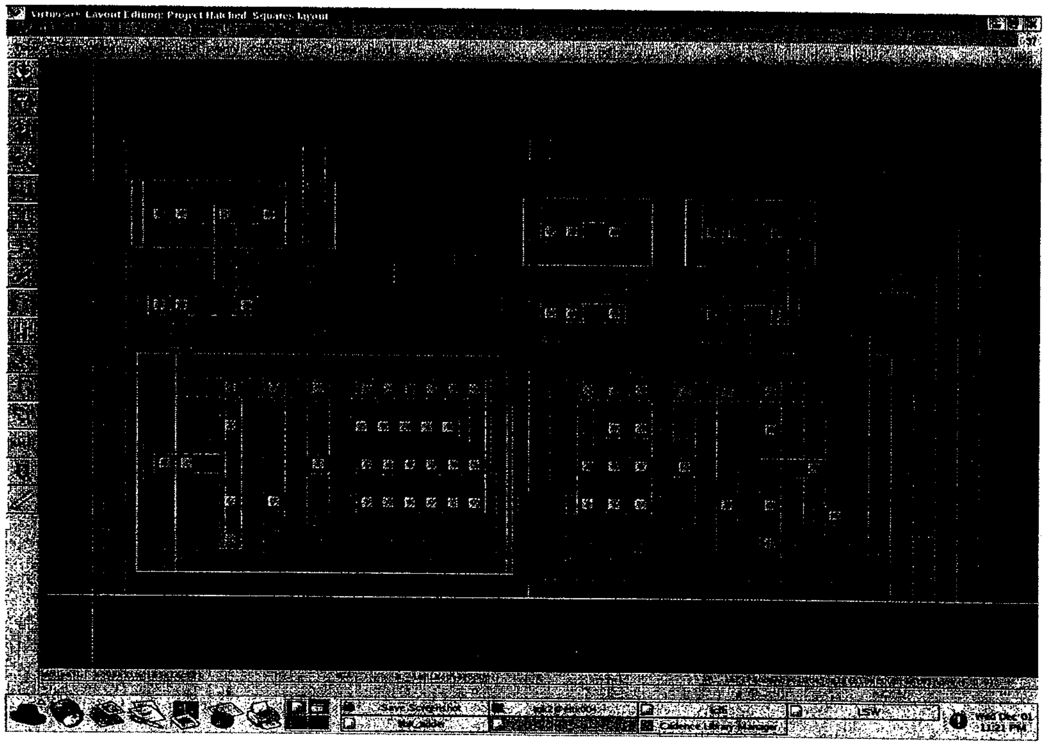
$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n) \quad (5)$$

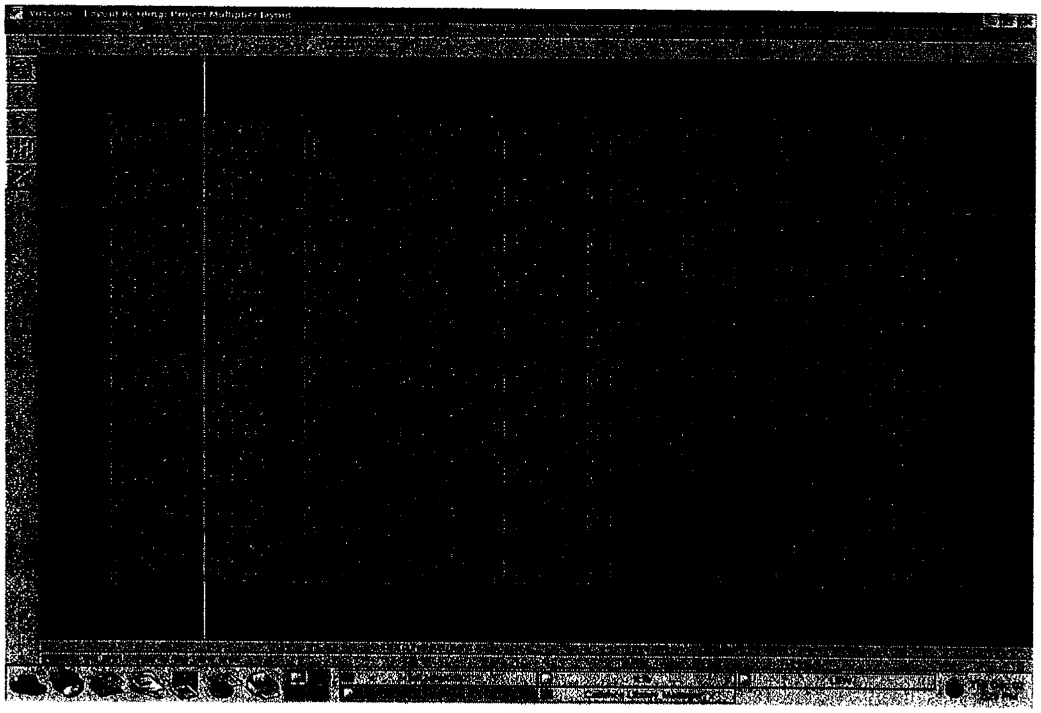
After the weights have been updated, a new training vector is applied and the process repeats. Given enough cycles through the entire training set (epochs), the synaptic weights will converge to their desired values and the NN will have 'learned' how to solve the desired problem.

3. ALU DESIGN

The ALU, which performs all of the neuron's computations, primarily consists of an adder and a multiplier. As in any digital design, decisions have to be made regarding how numbers will be represented. Multipliers can grow very large when required to compute numbers with long bit-lengths. In this case, inputs to the multiplier are restricted to nine bits. Since synaptic weights can theoretically grow to $\pm\infty$, a compromise is made that restricts the weights to nine bits in length while still allowing for high precision and magnitude. Using nine-bit 2's complement representations, the synaptic weights are constrained to ±15.9375 in size with a resolution of 0.0625. The use of 2's complement numbers allows for both positive and negative values to be represented without an increase in bit-length. The 9x9 *Baugh-Wooley* multiplier (Figure 2) is an array of specially connected carry-save adders that have been modified with simple NAND and AND logic gates.

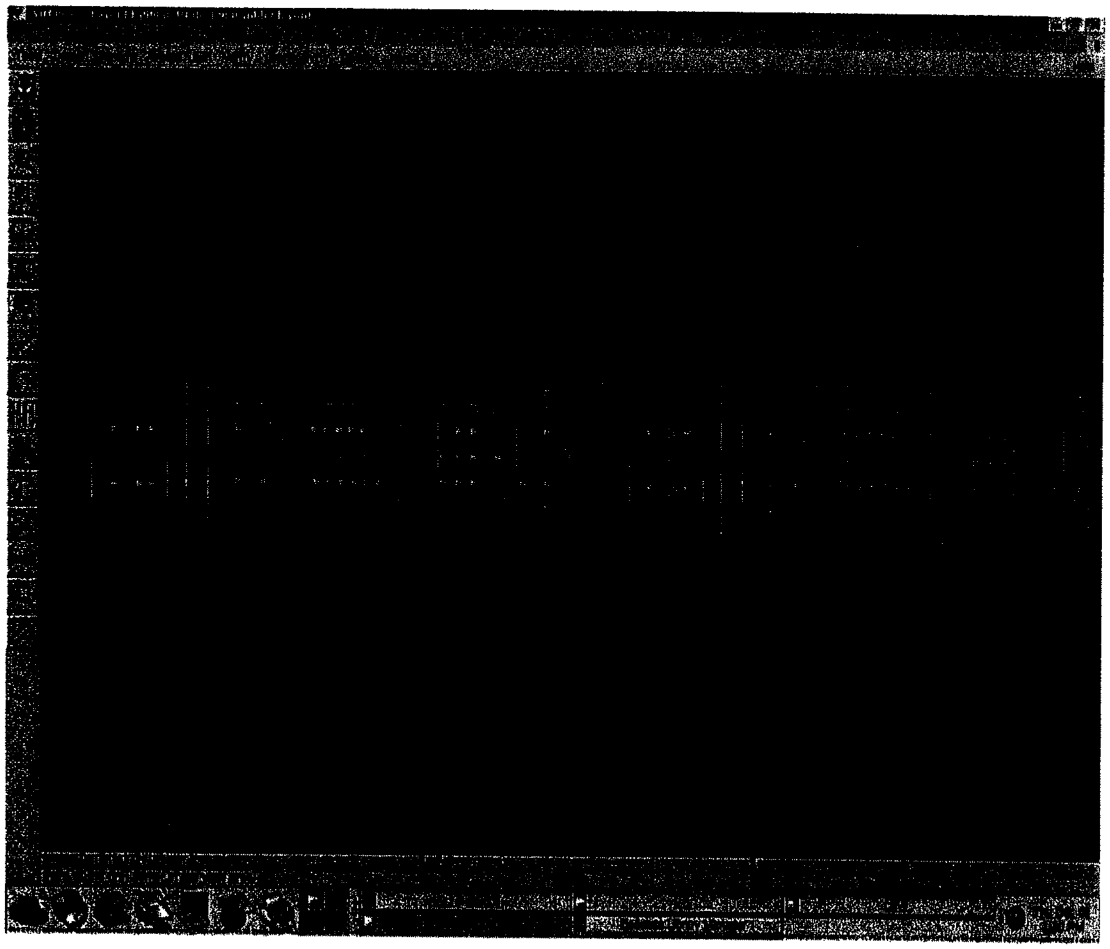
Figure 2. (**Top:** A single instance of the carry-save adder. **Bottom:** Instances are tiled to create the 9x9 multiplier.)





M x N Multiplication is performed by forming N partial products of M bits each, and then summing the appropriately shifted partial products to produce an M + N bit result. When two nine bit numbers are multiplied together they often produce outputs of more than nine bits. In a worst-case scenario the multiplier could produce an output as long as 17 bits (because only one of the sign bits is needed, not both). Because of this, the ALU requires a 17 bit ripple carry adder (Figure 3) to perform addition.

Figure 3. (A single bit-slice of the adder)



The back-propagation algorithm and sigmoid approximation require the ALU to perform subtraction and absolute value functions as well as addition and multiplication. These are implemented using an arithmetic extender, which is made up of just a few logic gates placed in front of the inputs to the ALU. Subtraction is done by adding the first number, A, to the inverse of the second number, B. In 2's complement, B is inverted by first inverting every bit and then adding 1. Absolute value is just a special case of subtraction. If the number is positive then nothing needs to happen. If it is negative, the number is just subtracted from zero. The arithmetic extender is described in more detail in Table 1.

Table 1. (S1 and S0 are select-bits determining the AE's functionality. The C(in) input allows for a carry bit during subtraction, and b(i) represents the ith bit of the B input.)

The AE operates on the B (the second input of the adder) based on the equation

- $b(i) = b(i)'*So+b(i)*So'$
- $C(in) = So$

S1 S0	Function	Operation	C(in)
0 0	Add	A+B	0
0 1	Subtract	A+B'+1	1
1 0	Multiply	X	X

4. CONTROLLER DESIGN

If the ALU is the computational heart of the neuron, then the controller is its brain. It tells the ALU which memory cells contain its inputs, what computation to perform, and where to store the output. Appendix 1 shows the controller described as a FSMD (finite state machine with a datapath). The FSMD is a flow chart which breaks down the forward and backward computation algorithms into individual steps. Each step represents a unique state of the controller. The controller moves from one state to another each clock cycle based on status signals from the main controller, as well as signals applied by the user to the chip's input pins and the satisfaction of conditional statements. In Appendix 2 the FSMD has been converted to a next-state table. Each state is given a binary encoding, and each of its possible next states is listed along with their associated status signals and conditional statements.

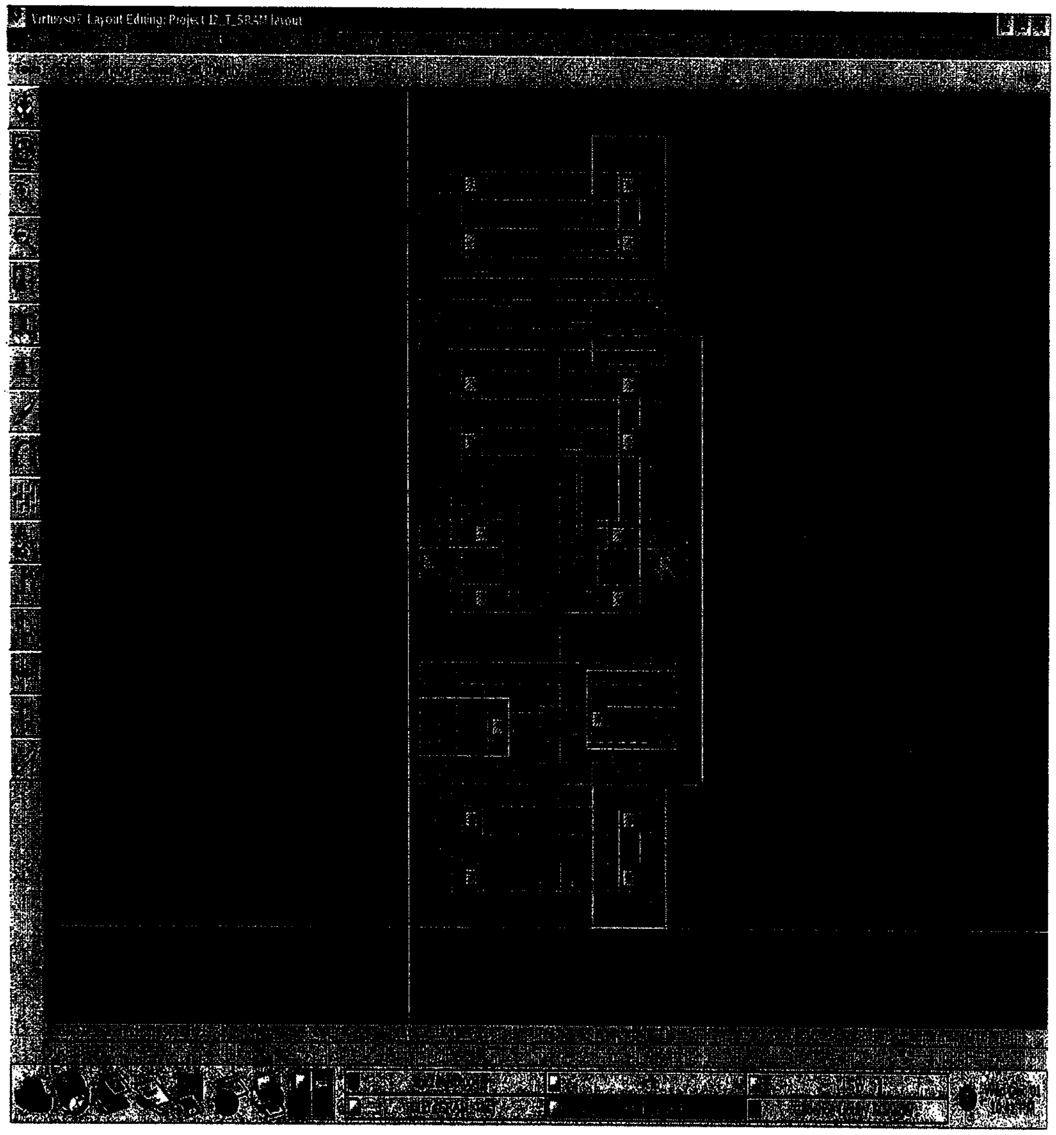
A string of D latches, one for each encoding bit, change on every clock cycle to produce the encoding for the next state. Control logic placed in front of the latches determines how they will change on each clock cycle. Output logic, which is placed behind the latches, uses the state encoding to produce status signals which control the reading and writing to memory as well as the functionality of the ALU. The control and output logic for a FSMD with 46 states is enormous, and is hardly ever calculated by hand. The beauty of a next-state table is that it can be input to a hardware description language such as Verilog or VHDL, which will generate a net list containing all of the necessary information for the control and output logic. By definition, an FSMD and a next-state table contain all of the information needed to synthesize the desired controller.

5. MEMORY DESIGN

The final element of interest in the neuron is memory. Two types of memory are used in this design. Constants are stored in a 5 word, 9 bit ROM. The rest of the memory is made up of single word, 9 bit SRAMs. The layout of a single 12 transistor SRAM is shown in figure 4. Each word consists of 9 such SRAM components, one for each bit. Smaller

SRAM designs are possible using as few as 6 transistors, but they have other drawbacks. The 6 transistor SRAM requires a pre-charger circuit placed on top and a sense amp on the bottom in order to function properly. The pre-charger supplies the power necessary to force the bit in memory to drop down towards the sense amp, which detects the arrival of the bit in order to output it. The savings in transistor count with the 6 transistor SRAM can actually lead to larger memory cells, especially when only a small amount of memory is needed.

Figure 4. (A single bit-slice of the 12 transistor SRAM)



6. OTHER REQUIRED COMPONENTS

As mentioned before, a neural network designed using the proposed generic neuron module requires a small controller to overlook the operation of the individual nodes. The chip will also require circuitry to regulate power, generate the clock signal, and distribute it across the chip with minimal clock skew. Problems can occur if clock skew, which arises when the clock signal arrives to different parts of the chip at different times due to delays caused by varying wire lengths, is not accounted for. Since every neuron is identical, each one in a given layer should finish its work at the same time. Problems with clock skew could potentially confuse the main controller, causing it to activate some neurons before they are ready.

All that is needed now to turn the generic neuron into a viable product is the design of the main controller and clock circuitry. A controller could be designed for any specific network configuration, but that would defeat the purpose of creating a customizable neural network application. Instead, it would be desirable to generate some sort of algorithm or computer application that could generate the controller design automatically, based on the user's input of the desired network configuration. Unfortunately that is beyond the scope of this project, as well as my programming abilities.

As for the clock, it too must be designed with regard to a specific neuron layout on the chip. This is the only way to ensure that clock skew does not become an issue. However, some of the clock's basic properties will remain constant throughout any choice of network design. Clock speed, which is not affected by the number of modules used in any given design, determines how fast the chip will operate. It is determined by the neuron's critical path, which represents the longest possible amount of time the neuron could take to perform an operation. The expected delays of the adder and multiplier (Figures 5 and 6) were determined through simulation. In this case the multiplier may exhibit a delay of up to 22 ns. Therefore, the critical path starts at the output logic, then goes to the SRAM, and finally ends with the multiplier. The period of the chip's clock must equal or exceed the critical path delay. Taking into account a few nanoseconds in the output logic and SRAM, as well as the 22 ns maximum delay for the multiplier, a conservative estimate (padded to ensure stability) of the clock's period would be 35 ns. This choice of clock period enables about 28.57 million computations per second, or a clock speed of about 28.57 MHz.

Figure 5. (Adder delay)

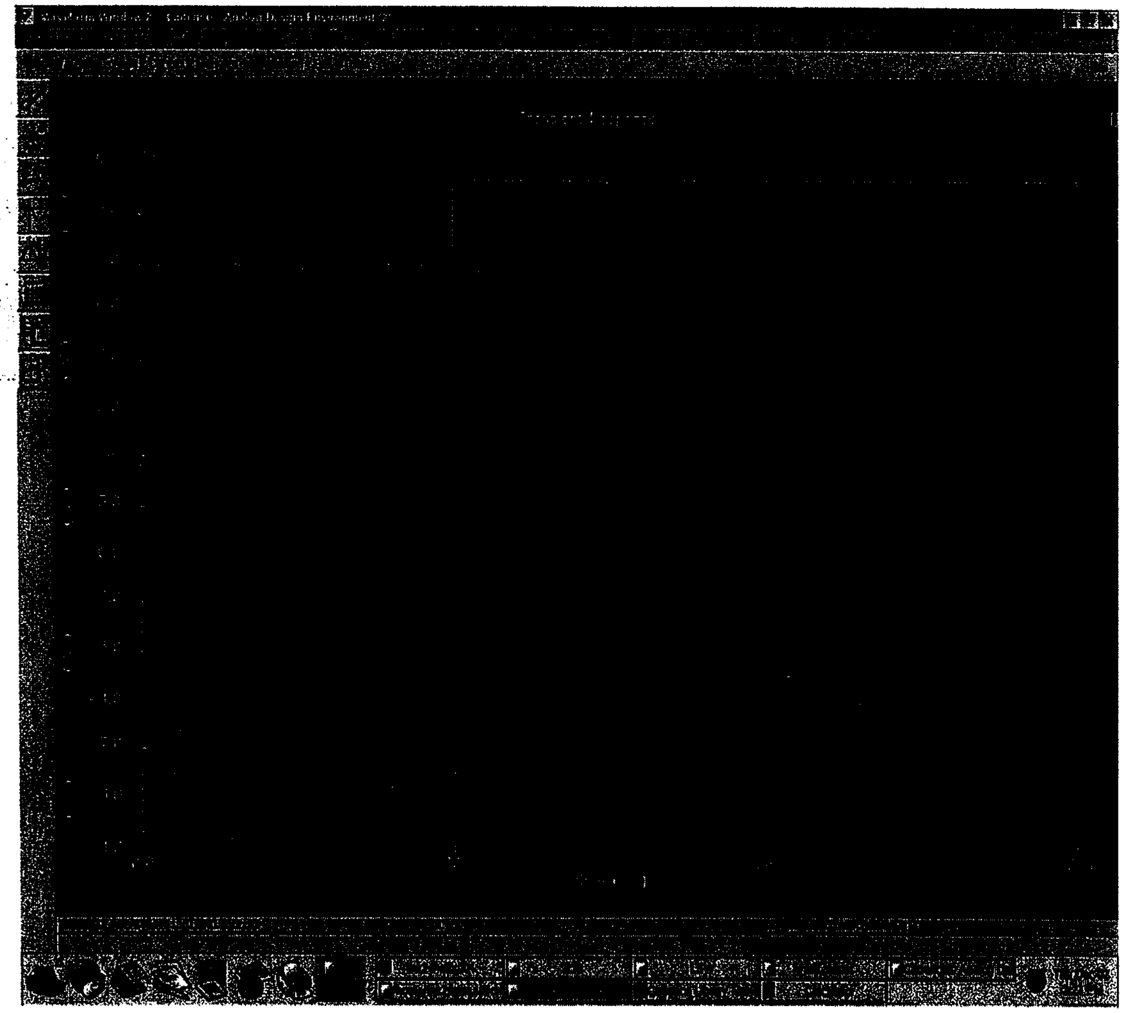
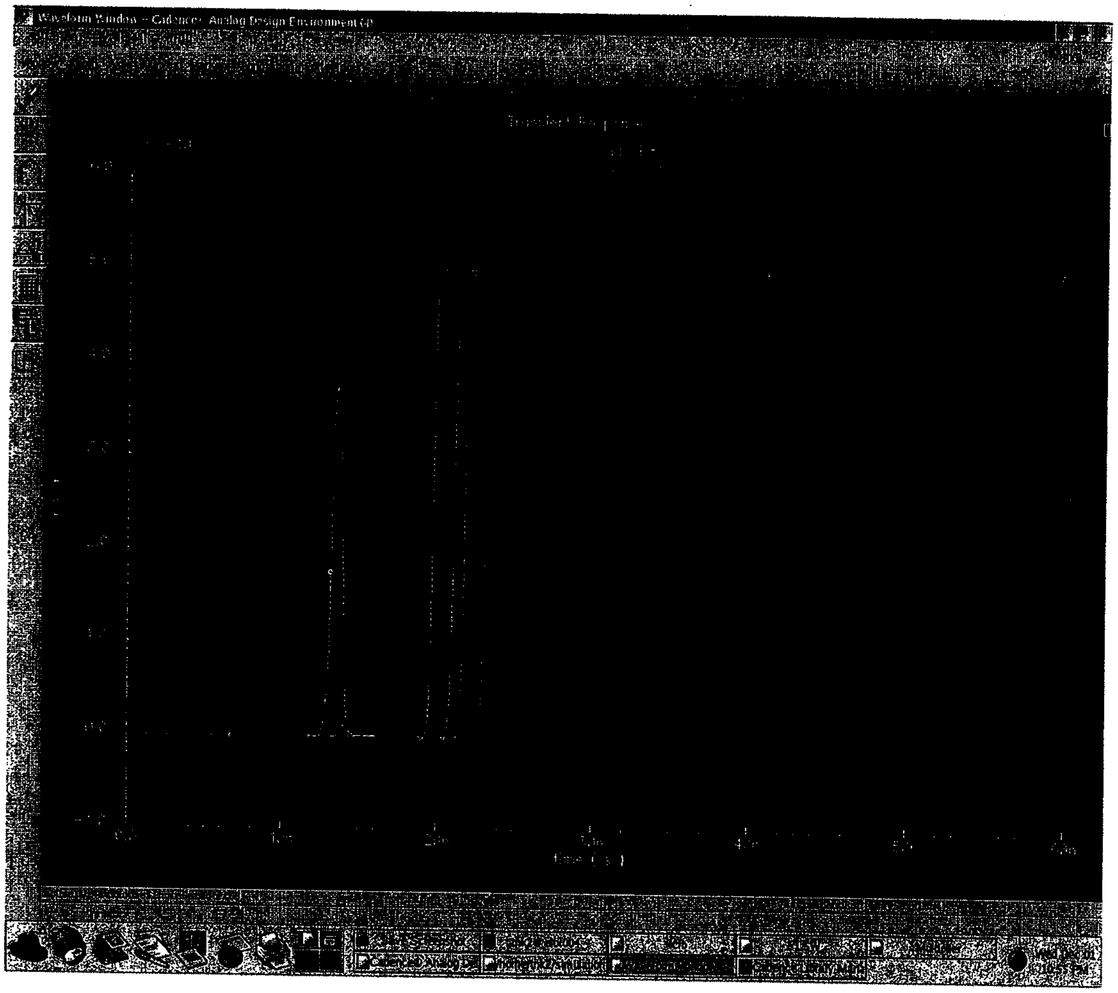


Figure 6. (Multiplier delay)



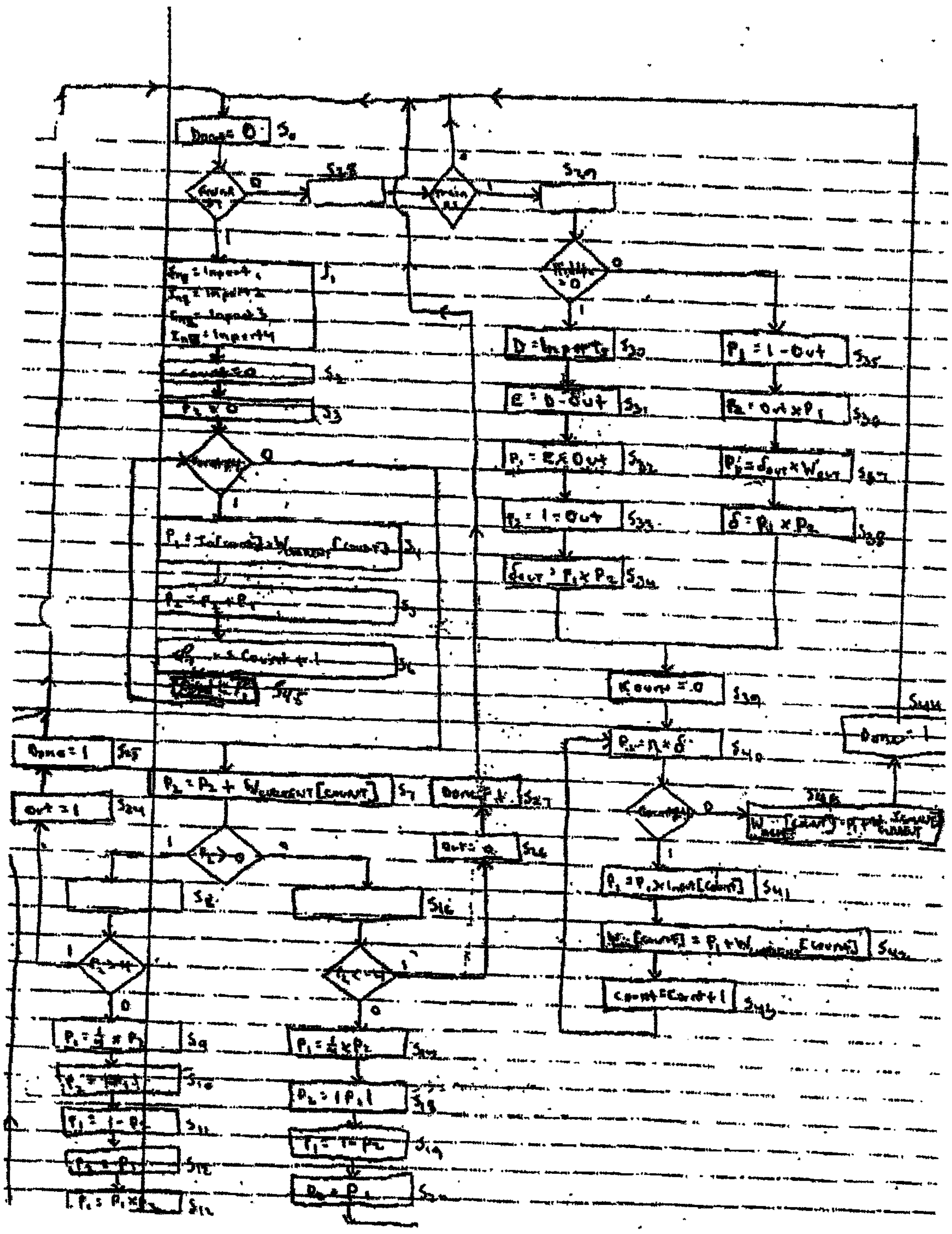
7. CONCLUSION

To put into perspective the capabilities of the proposed neuron its size, along with the size of a possible network layout, is estimated. A process parameter, _, is used in the size estimates. It represents half of the minimum channel length for a given process (0.25 μm in this case), and is the distance reference used in Cadence as well as most other VLSI CAD tools. The 9x9 multiplier is 1.4 M_^{2} and the 17-bit adder is 147 K_^{2}. The SRAM will take up 466 K_^{2}, and the control logic will represent about another 1 M_^{2}. The total estimated size of the neuron amounts to 3.013 M_^{2}, or 0.43 x 0.43 mm. A neural network consisting of 4 inputs, one hidden layer with 4 nodes, and a single output node would result in a chip size of about 1.5 mm^{2} (including clock, controller, and power). Networks of increasing complexity would obviously result in larger chip sizes.

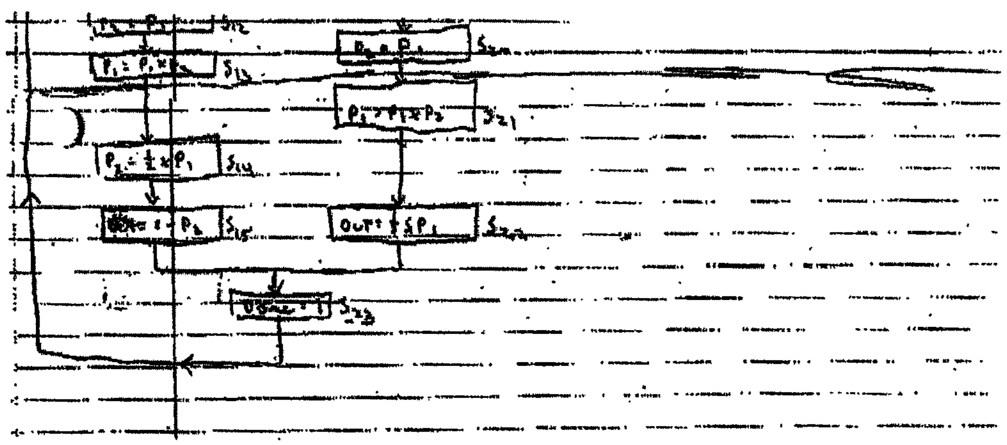
The final issue that must be addressed is the chip's pin requirements. Pins are required to apply input vectors to the network, as well as a few other user determined signals. During training, the user must supply the chip with a desired output value for each training vector. A single pin is also required for the binary status signal that tells the chip whether or not it is in training mode. Finally, pins are required to access the network's output, and to supply power to the chip. To access the network's output, 9 pins are needed (one for each bit). The same holds true for the pins where the desired output value is sent during training. The number of pins needed to apply the network's inputs can vary, depending on the size of the chip and the desires of the user. Inputs can be restricted to certain ranges of values (forcing the user to normalize/scale their inputs) in order to accommodate desired pin configurations. In total, pin requirements may range from as low as 40 to 60 or more.

The implementation of a generic MLP has been described through the design and layout of its functional parts, along with a discussion of the external components required to implement a custom neural network. The digital nature of the design limits both the speed of the neuron as well as the size. A mixed-signal (analog/digital) implementation, though beyond the scope of my abilities, would bring significant improvements to both issues. In analog VLSI design all calculations are done in current mode. Addition is performed merely by summing the currents coming into a wire junction, while multiplication requires only a few transistors. The weights and other stored values would still be held in digital memory, but the size of an individual neuron could be scaled down immensely. Thousands of mixed-signal neurons could fit on a chip capable of holding only 10 or 20 of their digital counterparts. In the future, the proposed digital design could be translated into the analog realm in order to accommodate increasingly larger and more complex neural network designs.

Appendix 1: Controller FSMD Flow Chart



(Continued on next page)



P₂ = P₁
S₁₂
P₁ = P₁ × P₂
S₁₃
P₂ = ½ × P₁
S₁₄
S₁₅
P₂ = P₁
S₂₀
P₁ = P₁ × P₂
S₂₁
S₂₂
S₂₃

Appendix 2: Controller FSMD Next-State Table

Present State		Next State		Datapath
$Q_5 Q_4 Q_3 Q_2 Q_1 Q_0$	St	Condition	St	
0 0 0 0 0 0	S_0	Forward=1 (y/n)	S_1/S_{28}	Done = 0
0 0 0 0 0 1	S_1		S_2	$In_{0,1,2,3}$ = Input$_{0,1,2,3}$
0 0 0 0 1 0	S_2		S_3	Count = 0
0 0 0 0 1 1	S_3	Count ≠ 4 (y/n)	S_4/S_7	$P_2 = 0$
0 0 0 1 0 0	S_4		S_5	P_1 = In[Count] × W_{input}[Count]
0 0 0 1 0 1	S_5		S_6	$P_2 = P_2 + P_1$
0 0 0 1 1 0	S_6		S_3	P_1 = Count + 1
0 0 0 1 1 1	S_7	$P_2 \geq 0$ (y/n)	S_8/S_{16}	$P_2 = P_2 + W_{hidden}$[count]
0 0 1 0 0 0	S_8	$P_2 \geq 4$ (y/n)	S_{24}/S_9	
0 0 1 0 0 1	S_9		S_{10}	$P_1 = \frac{1}{4} \times P_2$
0 0 1 0 1 0	S_{10}		S_{11}	$P_2 = P_1$
0 0 1 0 1 1	S_{11}		S_{12}	$P_1 = 1 - P_2$
0 0 1 1 0 0	S_{12}		S_{13}	$P_2 = P_1$
0 0 1 1 0 1	S_{13}		S_{14}	$P_1 = P_1 \times P_2$
0 0 1 1 1 0	S_{14}		S_{15}	$P_2 = \frac{1}{2} P_1$
0 0 1 1 1 1	S_{15}		S_{23}	Out $= 1 - P_2$
0 1 0 0 0 0	S_{16}	$P_2 \leq -4$ (y/n)	S_{26}/S_{17}	
0 1 0 0 0 1	S_{17}		S_{18}	$P_1 = \frac{1}{4} \times P_2$
0 1 0 0 1 0	S_{18}		S_{19}	$P_2 = \lvert P_1 \rvert \rightarrow$ 2's complement amp
0 1 0 0 1 1	S_{19}		S_{20}	$P_1 = 1 - P_2$
0 1 0 1 0 0	S_{20}		S_{21}	$P_2 = P_1$
0 1 0 1 0 1	S_{21}		S_{22}	$P_1 = P_1 \times P_2$
0 1 0 1 1 0	S_{22}		S_{23}	Out $= \frac{1}{2} \times P_1$
0 1 0 1 1 1	S_{23}		S_0	Done = 1
0 1 1 0 0 0	S_{24}		S_{25}	Out = 1
0 1 1 0 0 1	S_{25}		S_0	Done = 1
0 1 1 0 1 0	S_{26}		S_{27}	Out = 0
0 1 1 0 1 1	S_{27}		S_0	Done = 1
0 1 1 1 0 0	S_{28}	Train = 1 (y/n)	S_{29}/S_0	
0 1 1 1 0 1	S_{29}	Hidden = 0 (y/n)	S_{30}/S_{35}	
0 1 1 1 1 0	S_{30}		S_{31}	D = Input$_5$
0 1 1 1 1 1	S_{31}		S_{32}	E = D − Out
1 0 0 0 0 0	S_{32}		S_{33}	P_1 = E × Out
1 0 0 0 0 1	S_{33}		S_{34}	P_2 = 1 − Out
1 0 0 0 1 0	S_{34}		S_{35}	$S_{out} = P_1 \times P_2$
1 0 0 0 1 1	S_{35}		S_{36}	P_1 = 1 − Out

(Continued on next page)

1	00100	S_{36}		S_{37}	$P_2 = O_{ut} \times P_1$
1	00101	S_{37}		S_{38}	$P_1 = \delta_{out} \times W'_{out}$
1	00110	S_{38}		S_{39}	$\delta = P_1 \times P_2$
1	00111	S_{39}		S_{40}	Count = 0
1	01000	S_{40}	count ≠ 4 (y/n)	S_{41}/S_{46}	$P_1 = \eta \times \delta$
1	01001	S_{41}		S_{42}	$P_2 = P_1 \times$ Input[count]
1	01010	S_{42}		S_{43}	W_{new}[count] = $P_1 + W_{current}$[cou
1	01011	S_{43}		S_{44}	count = count+1
1	01100	S_{44}		S_0	Done = 1
1	01101	S_{45}	count ≠ 4 (y/n)	S_4/S_7	count = P_1
1	01110	S_{46}		S_{49}	W_{new}[count] = $P_1 + W_{current}$[co

}

NeuroEvolution of a Pole Balancing Vision and Control System

Reuben Grinberg
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract

I attempt to use NeuroEvolution of Augmenting Topologies (NEAT), a genetic algorithm that operates on neural networks, to evolve a solution to an augmented version of the well-studied pole balancing problem. Instead of using state variables, the evolved solution must use video from a camera that faces the cart and pole. I verify that NEAT can solve the original pole balancing task. Memory constraints prevented evolution of a network that takes visual input. Future work should try to reduce the memory requirements of the problem by optimizing program code or by using a "roving eye" that looks at small portions of the input at a time.

1. INTRODUCTION

1.1. Vision

Hand-built solutions to vision problems often label the world and act on those labels. A system that does edge detection using a Sobel matrix and then Hough transforms to find circles is an example of such a system. These approaches may miss subtlety in the domain that makes a simple and robust solution possible.

Evolution may be one way to take advantage of this subtlety. Evolutionary techniques may one day be powerful enough to find robust, accurate solutions to vision problems more quickly than an engineer. Some examples of difficult problems that might benefit from an evolutionary approach are general face recognition, eye gaze tracking, and hand gesture recognition.

To see if an evolutionary approach to these difficult problems is feasible, I attempt to use evolution to solve a simpler vision problem: pole balancing.

1.2. Pole Balancing

Pole balancing or inverted pendulum balancing is a well-studied problem and is often used to assess new learning algorithms. A rolling cart is attached to a one-dimensional rail and a pole is attached to the cart so that it may fall only parallel to the rail. The task is to move the cart in such a way that the pole stays balanced. The available variables are the cart's position, x, the cart's velocity, x', the pole's angle from the vertex, ϕ, and the angular velocity of the pole, ϕ'. The output from a control system is the force to apply to the cart. If the cart hits the barriers of the rail or if the pole falls past a certain angle, the control system has failed. (see figure 1)

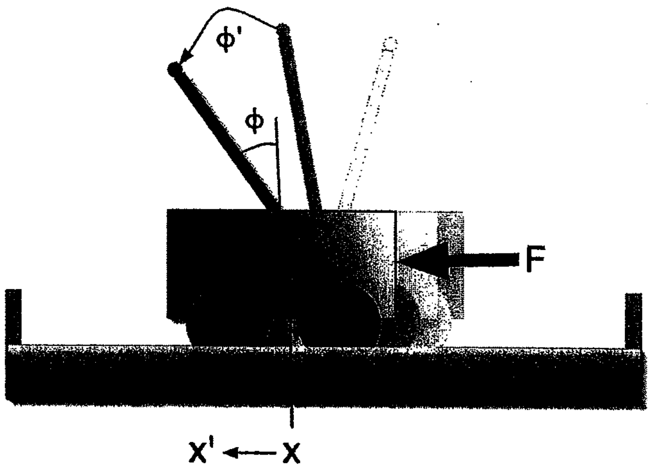


Figure 1. A cart with wheels is attached to a rail and cannot roll past the barriers on the left and right. F is the force applied to the cart by a program, x is the position of the cart, x' is the velocity of the cart, ϕ is the pole's angle, and ϕ' is the pole's angular velocity.

There exist many different solutions to the pole balancing problem; neural networks that take the state variables as input solve it adequately (Stanley and Miikkulainen, 2002)

The new task is to balance the pole using video feed of the cart and pole instead of the state variables. The video feed is from a camera that faces the cart perpendicular to the rail (see figure 2).

A non-evolutionary approach to this new task is to do image transformations to separate the foreground and background, find where the cart and the pole are in the image, and then use an approach like a Hough transform to find the cart position and the pole angle. After obtaining the state variables one of the proven methods can be used to balance the pole.

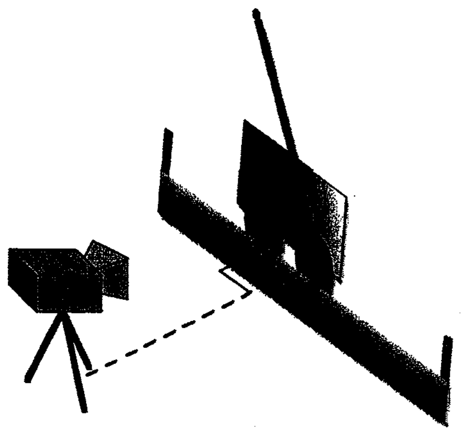


Figure 2. The video feed is from a camera facing the cart and is perpendicular to the rail.

Here are two possible evolutionary approaches to the new task. One is to use evolution to create a system that extracts the state variables from the video feed and then feeds these variables to a proven balancing method. Another is to evolve the whole solution in one comprehensive network.

1.3. NeuroEvolution

Using genetic algorithms to evolve neural network controllers has been shown in several different papers to be a fast and efficient way to solve the pole balancing problem (Moriarty, 1997; Moriarty and Miikkulainen, 1997).

There are two different approaches to evolving neural networks or NeuroEvolution (NE): evolving only the network weights and evolving the topology along with the network weights. The first approach is just like other algorithms that train network weights such as Hebb's rule or backprop. The second, however, is much more general. Evolved topologies may include excitatory, inhibitory, and recurrent links. Since recurrent neural networks are Turing complete (Hyötyniemi, 1996) and NE of topologies has the potential to evolve any recurrent network, it is probable that NE can solve any computable problem.

1.3.1. NEAT

NEAT (NeuroEvolution of Augmenting Topologies; Stanley and Miikkulainen, 2002), one of the methods shown to solve pole-balancing efficiently is a method that evolves both weights and topologies of neural networks. Stanley and Miikkulainen (2002) has a good discussion of the merits of the approach, comparisons to some other popular NeuroEvolution techniques such as ESP and SANE, and examples of how it can be used.

Every time a NEAT neural network is activated, the inputs for every neuron are summed and fed into the activation function, the sigmoidal transfer function $\varphi(x)=\frac{1}{1+e^{-4.9x}}$, and the output is relayed to connected post-synaptic neurons. That is, for every time step, information travels one layer. If there are 5 neurons in between the inputs and the outputs, the output will not be affected by a specific input value until the network has been activated 5 times. Because of this, a network can encode memories in recurrent links.

2. EXPERIMENTS

Practical considerations require us to simulate the cart and the camera. Using the simulated state variables, a program creates a matrix representing the view from the camera. Every pixel that is the cart or pole is set to one; the rest of the matrix is zero. For this experiment, the length of the rail was arbitrarily set to 5.2; the barriers are at –2.4 and 2.4. The cart must keep the pole between -12 and 12 . See figure 3 for examples of these matrices.

```
000000110000000000000000000000000000000000000000000000000000
000000110000000000000000000000000000000000000000000000000000
000000110000000000000000000000000000000000000000000000000000
000000110000000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000001100000000000000000000000000000000000000000000000000
000000000011000000000000000000000000000000000000000000000000
000000000011000000000000000000000000000000000000000000000000
000000000011000000000000000000000000000000000000000000000000
000000111111110000000000000000000000000000000000000000000000
000000111111110000000000000000000000000000000000000000000000
```

```
000000000000000000000000000000000000000000001100000000000000
000000000000000000000000000000000000000000001100000000000000
000000000000000000000000000000000000000000001100000000000000
000000000000000000000000000000000000000000000011000000000000
000000000000000000000000000000000000000000000011000000000000
000000000000000000000000000000000000000000000011000000000000
000000000000000000000000000000000000000000000011000000000000
000000000000000000000000000000000000000000000000110000000000
000000000000000000000000000000000000000000000000110000000000
000000000000000000000000000000000000000000000000110000000000
000000000000000000000000000000000000000000000000110000000000
000000000000000000000000000000000000000000000000001100000000
000000000000000000000000000000000000000000000000001100000000
000000000000000000000000000000000000000000000011111111000000
000000000000000000000000000000000000000000000011111111000000
```

```
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000000011000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000000001100000000000000000000000000000000000000
000000000000000011111111000000000000000000000000000000000000
000000000000000011111111000000000000000000000000000000000000
```

```
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000001100000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000000000110000000000000000000
000000000000000000000000000000000001111111100000000000000000
000000000000000000000000000000000001111111100000000000000000
```

Figure 3. These binary matrices were used to represent frames from a video camera facing the cart and pole. The background is set to 0 and the pole and cart are 1.

2.1. Experiment 1

Because I was using Java ports[1] of the original NEAT software that do not include pole-balancing experiments, I wanted to verify that these ports could solve the pole-balancing problem as well as the original version of the software.

2.1.1. Results

[1] I ported Stanley's single pole experiment in experiments.cpp from C++ to Java for use in JNeat, a Java version of NEAT written by Ugo Vierucci (http://www.cs.utexas.edu/users/nn/downloads/software/JNEAT.zip). I had to modify gui.Generation.evaluate to allow evaluation using my pole balancing code.

I verified Stanley's and Miikkulainen's (2002) result that NEAT successfully solves the pole-balancing problem.

NEAT very quickly evolved a solution that could balance the pole for more than 100,000 time steps. (see figure 4).

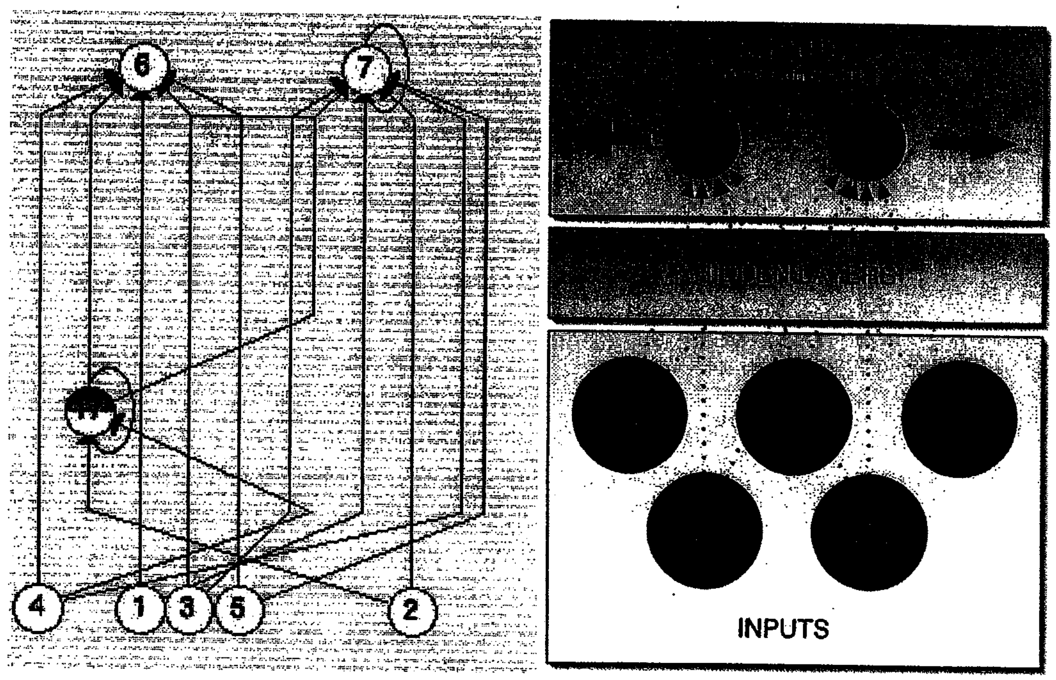


Figure 4. On the left is a solution to the single-pole balancing problem evolved using NEAT. 1 is Bias, 2 is x, 3 is x', 4 is ϕ, and 5 is ϕ'. Blue lines signify excitatory connections and red lines signify inhibitory connections. Node 17 is a hidden node that was added by evolution. The recurrent links on nodes 7 and 17 are also products of evolution. As illustrated on the right, if the first output neuron is more excited than the second then force is applied to the left. Similarly, if the second neuron is more excited than the first, then force is applied to the cart to the right.

2.2. Experiment 2

Attempts to evolve a comprehensive solution that uses video feed as input failed because of technical issues. Because of the high number of possible links, the two implementations of NEAT that I used (ANJI[2] and JNeat) both failed because of lack of memory, even though I had assigned two gigabytes of memory to the java runtime[3].

[2] http://anji.sourceforge.net/
[3] I increased the memory available to java with the flag '-Xmx2000m'

2.3. Experiment 3

Because of the memory problem, I had no hope of evolving a network that could handle the image. I decided to use a backprop trained feedforward neural network to extract the cart position and pole angle from the images of the cart and pole.

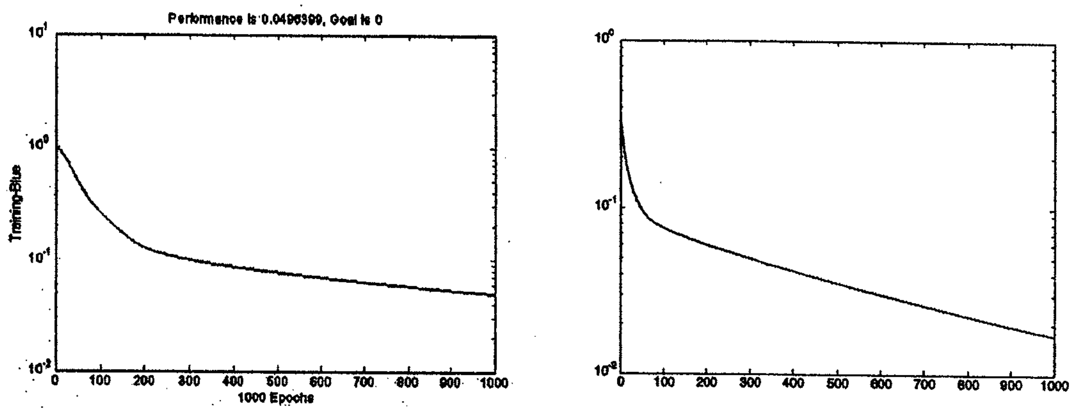
2.3.1. Methods

I made two sets of 120 test matrices (like the ones in figure 2) with the cart position at {-2.4, -1.88, -1.36, -0.84, -0.32, -0.01, 0.51, 1.03, 1.55, 2.07} and with the pole angle at {-12 , 9.6 , -7.2 , -4.8 , -2.4 , 0 , 2.4 , 4.8 , 7.2 , 9.6 }[4]. The first set had the dimensions 30 by 15 pixels, and the second set had the dimensions 100 by 50 pixels. Using Matlab's neural network toolbox, I created a neural network with one input layer and one output layer using tansig as the evaluation function. The number of neurons in the input layer was 15 x 30 = 450 for the first set and 100 x 50 = 5000 for the second set. The output layer had two neurons – one for cart position and another for pole angle. To create the targets for training, I normalized the position and angle variables so that they were between 0 and 1. For example, the target for the network that looks at the image created with the cart position equal to 2.4 and pole angle equal to -12 is (0,0); the target for network that looks at an image created with cart position equal to –0.84 and pole angle equal to 0 is (.5, .5).

2.3.2. Results

After training for 1000 epochs, the results were satisfactory; they are summarized in figure 5.

[4] Angles outside of [12 ,-12] are not needed because the run is considered failed in that region.



30 x 15 grid (450 inputs)	*Mean*	*Min*	*Max*
x	0.09%	~0	1.6%
ϕ	4.89%	~0	20.02%
100 x 50 grid (5000 inputs)			
x	1.89%	~0	16.87%
ϕ	3.20%	~0	27.14%

Figure 5. The result of training a feedforward network on 121 (30 x 15) and (100 x 50) grids. On the top left are the results of a network that has 15 x 30 = 450 input nodes, no hidden layers, and two output nodes. On the top right are the results of a network that has (100 x 50) = 5000 input nodes, no hidden layers, and two output nodes.

This neural network could serve as the first part of the first evolutionary approach mentioned in the introduction. The second part requires a network that can balance the pole using only the cart position and pole angle.

2.3. Experiment 4

2.3.1. Methods

To utilize the backprop network constructed in experiment 3 a network that can keep the pole balanced using only the cart position and pole angle is required. It is impossible to get velocities by looking at a single frame as the backprop net is doing.

2.3.2. Results

Unfortunately, evolving a network that could solve the pole-balancing task using only cart position, pole angle, and bias did not work. For both of the approaches illustrated in figure 6, the best results achieved were around 250 – 300 time steps (100,000 steps is a successful balancing). These results were obtained around the 150th generation for both networks and had not improved at generation 450.

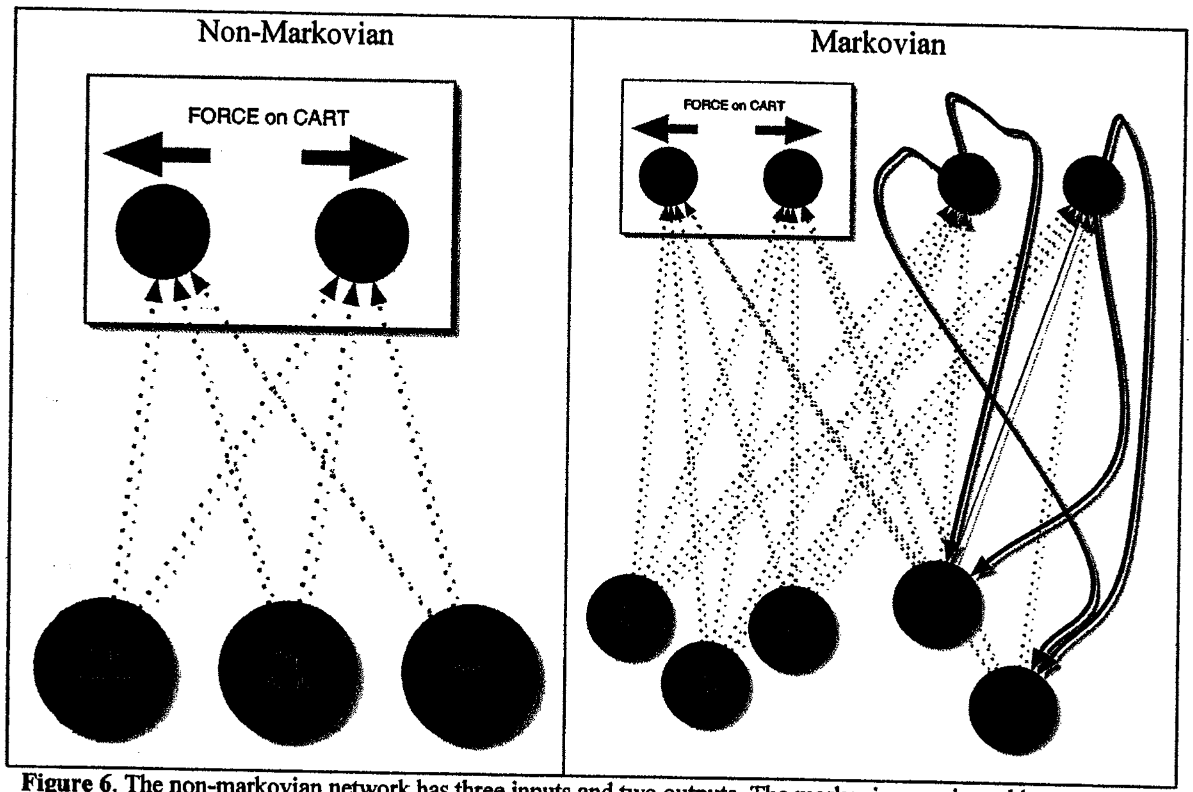


Figure 6. The non-markovian network has three inputs and two outputs. The markovian version adds two output nodes with fixed links to two input nodes. The markovian network is a reachable network from the non-markovian via evolution, but reduces the search space. My reasoning was that the fixed recurrent links could be used to derivate x' and ϕ'.

3. CONCLUSION

I verified that NEAT could solve the simple pole-balancing task, but memory issues prevented successful solution of the visual pole-balancing task. Since the goal was to show the feasibility of using evolution techniques to solve vision problems, the successful backprop training of a neural network to extract pole angle and cart position from images is irrelevant. NEAT failure to evolve a network that can solve the pole-balancing problem without velocity information is troubling. Perhaps this task is much more difficult than it seems.

It is possible that a bug in the code prevented proper evolution for these tasks. Poor choices for the NEAT constants may have also prevented successful evolution of a pole-balancer that does not use velocity input.

Evolutionary computation implementations often have numerous constants that strongly affect the speed of search and whether a solution is found at all. Some of NEAT's constants are: mutation rate, survival threshold, age significance, mate only probability, population size, dropoff age, and others for a total of 34 constants. Successful alchemy with these constants may allow us to succeed where current efforts have not. It would be useful to have an NE implementation that automatically and adaptively optimized its constants during evolution.

A potential pole-balancing system using both of the components successfully created here is illustrated in Figure 7. The plan proposed in Figure 7 requires a perfect image. If we could obtain a perfectly segmented image from nature in a robust way we would probably never need evolutionary computation. Furthermore, it is probable that perturbing the "perfect image" slightly (by translating it by 1 pixel, for example) will destroy the feed forward network's ability to extract the pole angle and cart position. It is not clear what is gained by implementing the hodge-podge and somewhat arbitrary plan in figure 7.

Future work should fix the memory issue so that high dimensional input networks can be evolved. There is already a suggestion on Ken Stanley's homepage[5] where to look for one memory hogging procedure.

One possible solution to the memory issue is to use a roving eye. Stanley (2002) describes evolving a small network that can play the game GO on large boards. Instead of using the whole board state as input to the network, Stanley constructed a roving eye that

[5] http://www.cs.utexas.edu/users/kstanley/neat.html

could "see" a limited part of the board and could move around the board. To solve the visual version of the pole-balancing problem, a roving eye could be used that can look at limited portions of the current frame and move itself about.

It may make sense to look at some problem domain that is even simpler than pole balancing

4. ACKNOWLEDGEMENTS

Ken Stanley and members of the NEAT user's group[6] were very helpful in answering my questions. Professor Willard Miranker provided useful criticisms and comments on drafts of this work.

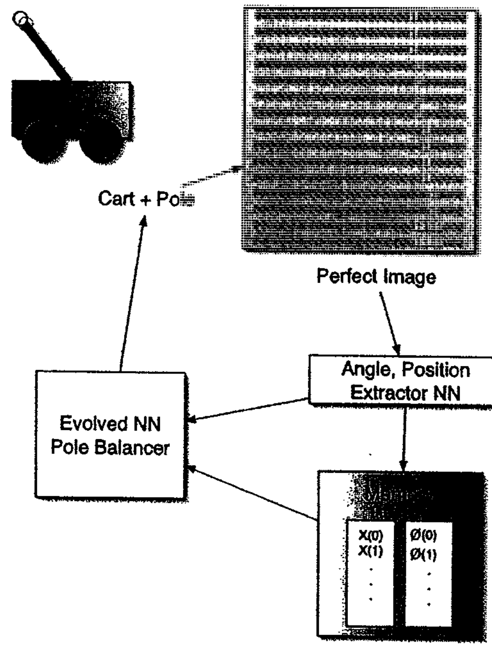


Figure 7. A possible plan of action to control the cart using the modules constructed. The memory module simply remembers x and ϕ for t-1 and returns $\frac{x(t)-x(t-1)}{\tau}$ to approximate x' (and the same for ϕ) where τ is the time period over which the simulation is discretized. This module, together with the angle extractor, should give the NEAT-evolved pole-balancer the variables it needs to work successfully.

5. BIBLIOGRAPHY

[6] http://groups.yahoo.com/group/neat/

Hyötyniemi, H. (1996) Turing Machines are Recurrent Neural Networks. Publications of the Finnish Artificial Intelligence Society, pp. 13-24. (http://www.uwasa.fi/stes/step96/step96/hyotyniemi1/)

Moriarty, D. E. (1997). Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report. UT-AI97-257.

Moriarty, D. E. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive co-evolution Evolutionary Computation, 5(4):373–399.

Stanley, K.O and Miikkulainen, R. (2002) Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation 10(2):99-127. (http://www.cs.utexas.edu/users/nn/pub-view.php?RECORD_KEY(Pubs)=PubID&PubID(Pubs)=116)

Image Classification Using A Self-Organizing Tree Algorithm Neural Network

David S. Hughes and Wesley C. Maness
Department of Computer Science, Yale University, New Haven, CT 06511

Abstract
Self-Organizing Tree Algorithm (SOTA) Neural Networks were developed for topological distribution analysis of protein sequences and other complex amino-acid structures. We present the application of SOTA networks to image classification. By building a 'genetic' sequence for each image through image processing, we can use a SOTA network to classify a series of images into arbitrarily detailed clusters. These classifications can then be used in searching a set of images for similar images or in comparing images based on color and textural features at different resolutions.
Keywords - neural networks, feature extraction, self-organizing feature maps, image classification

1. INTRODUCTION

As the amount of image and multimedia content on the web as well as in private databases increases, new methods are needed to search for salient needles in this haystack of content. While text-based filename and metadata searching techniques have been traditionally used, it is still unclear how to search images and other multimedia based on content. There are many applications in which it would be interesting to find similar images to a given image, particularly in finding work that suits a user's tastes with a minimal amount of effort. 'Find me the paintings that are most similar to this one' is a reasonable question that is still difficult to answer with traditional search techniques. Neural networks provide a way of classifying data in ways that will permit finding similar images based on a 'phylogenetic' tree structure. This method is explored in detail in Section 2.

To build sequences representing spatial and color characteristics of images, we use image processing and image resampling to create sequences of values representing information about each pixel. These values may be as simple as the amount of red color per pixel, or may be more complex values like the magnitude of the gradient, discussed in Section 3.2.3, at each pixel. Once these sequences are created, we can sort them and sample deterministically at a desired density, creating a sampled sequence that reflects the distribution of the relevant values in the image. This process is discussed in detail in Section 3.2.

We will use unsupervised neural networks, and in particular self-organizing maps in our image classification system. These types of networks provide a robust framework for clustering of image data. Neural networks are suitable for image classification since they work well with data sets in which there are outliers and poorly defined clusters.

SOFM (Self-Organizing Feature Maps) have proven to be easily scalable to large data sets. The SOFM is generally a two-dimensional grid with a rectangular or hexagonal topology, where the number of nodes is fixed before execution. The weights of nodes are randomly initialized, and the training process changes these weights to capture the distribution of the data set. At the end of the training process, clusters are assigned to the nodes of the SOFM grid. The distance between trained nodes is representative of the distance between cluster centers in N-dimensional space, where N is the size of the input. This reduction of the input space is a salient property when dealing with large data sets. However, SOFM is a topology preserving neural network, meaning that the number of clusters is arbitrarily fixed from the beginning. Thus, the clustering that results may not be representative of the number of logical clusters in the analyzed data. Another limitation of SOFM is the absence of a hierarchical structure, which makes it impossible to detect higher order relationships between data clusters.

Hierarchical clustering allows higher order relationships between clusters to be detected. This is a simple distance-based clustering method that has the desirable property of growing its topology dynamically. The clustering procedure is as follows:

- Assign a cluster to each item (so that if there are N items, there are N initial clusters).
- For the closest pair of clusters, merge them into one cluster, reducing the total number of clusters by one.
- Recompute distances between the new merged cluster and the other clusters.
- Repeat steps 2 and 3 until all items are clustered into one cluster of size N.

This is an agglomerative clustering method, meaning that it starts with many clusters or elements and joins them together. The neural network structure described in the next paragraph is divisive, meaning that it starts with the entire input set and divides it into more and more clusters until its growth is complete. Unfortunately, the height of the trees produced with hierarchical clustering cannot be controlled, leaving us again with a method whose results may not be representative of the logical clustering in the input data.

Since SOFMs do not provide a distribution preserving output and are unable to detect higher order relationships, and we cannot control the growth of clusters in hierarchical clustering, we will use the Self-Organizing Tree Algorithm (SOTA), presented in [1]. SOTA is a self-organizing neural network with an adaptive binary tree topology that provides divisive clustering to arbitrary depths. This network clusters based on a level of variability allowed within each cluster, dividing a cluster when its heterogeneity reaches a threshold. Although this is a divisive clustering method, its hierarchical structure preserves higher level 'phylogeny', so we can also agglomerate results upward to find relationships between low-level clusters. We propose here an application of the SOTA neural network to image classification.

In Section 2 we discuss in detail the SOTA neural network. In Section 3 we discuss how we implemented our image classification application. Finally, our conclusions and a discussion of other possible applications and future work are given in Section 4.

2. THE SELF-ORGANIZING TREE ALGORITHM (SOTA)

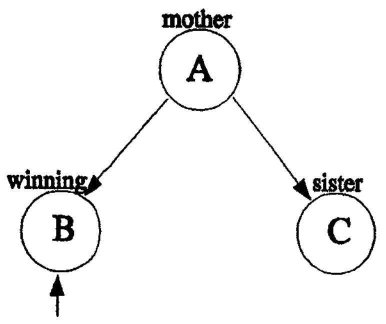
SOTA is based on the SOFM [2]. The SOFM algorithm proposed by Kohonen generates a mapping from a complex input space to a simpler output space. The input space is defined by the experimental input data, whereas the output space consists of a set of nodes arranged according to certain topologies, usually two-dimensional grids. The application of the algorithm maps the input space onto the smaller output space, producing a reduction in the complexity of the analyzed data set. In the case of SOTA, the output is a binary tree. In this algorithm a series of nodes, arranged in a binary tree, is adapted to intrinsic characteristics of the input data set. Development of this tree can be stopped at any taxonomic level or when all data is processed.

Distances, which are Euclidean, are obtained from the pair-wise comparison of image data for each image. If we have two images with their corresponding image data: $image1(i_{11}, i_{12}, .. i_{1n})$ and $image2(i_{21}, i_{22}, .. i_{2n})$, the distance between the two image nodes is $d_{1,2}$ where:

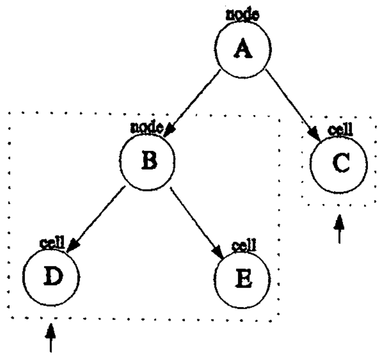
$$d_{1,2} = \sqrt{\sum_{t=1}^{n}(i_{1t} - i_{2t})^2} \qquad (1)$$

It is important to note that the $image1$ and $image2$ vectors do not represent actual image data (pixel values), but a 'genetic sequence' of features extracted from two images. Details of this feature extraction are presented in Section 3.

The initial system, shown in Fig.1(a), is composed of two external elements, denoted as cells, connected by an internal element, that we will call a node. Each cell (or node) is a vector with the same size as



(a) Starting point of the network



(b) Topological neighborhood for different possible winning cells

Fig. 1. Shown in Fig.1(a) is a system composed of two cells, B, and C, connected by a means of an ancestral node. Inputs are presented to nodes B and C. Let's assume B is chosen for a particular input, as shown by the arrow, B is updated accordingly. In the event of cell D, as shown in Fig.1(b), being the winning cell, the neighborhood extends to itself and their mother and sister cells B and E, respectively. Nevertheless, in the case of cell C, the neighborhood includes only itself. The reason for this is that if cell A were updated through C but not through B, it would receive an asymmetrical updating; and in this case cell A would not be a good representation of their descendants B and C.

the image data. Each component in the vector corresponds to a column in the data set, that is, to one of the conditions under which the image data has been measured. In the beginning, the entries of the two cells and the node are initialized with the mean value of the corresponding column of the data set. The algorithm proceeds by expanding the output topology, shown in Fig.1(b), starting from the cell having the most heterogeneous population of associated input data. Two new descendants are generated from this heterogeneous cell, which then changes its state from cell to node. The series of operations performed until a cell generates two descendants is called a cycle. During a cycle, all image genes are input sequentially into the SOTA neural network and each is assigned to the closest cell. Thus, the network is trained only through its terminal neurons. This process of successive cycles of generation of descendant cells can last until each cell has one single input image data assigned (or several, identical profiles), producing a complete classification of all the image data.

Adaptation in each cycle is carried out during a series of epochs. Each epoch consists of the presentation of all the image data to the network. A presentation implies two steps: first, finding the best matching cell (winning cell) for each image data profile, that is, the cell with the lowest distance cell-profile and second, to update this cell and its neighborhood. Cells are updated by means of the following formula:

$$C_i(\tau+1) = C_i(\tau) + \eta(P_j - C_i(\tau)) \tag{2}$$

where η is a factor that accounts for the magnitude of the updating of the i^{th} cell depending on its proximity to the winning cell within the neighborhood, $C_i(t)$ is the i^{th} cell vector at the presentation t, and P_j is the j^{th} image data profile vector. The topological neighborhood of the winning cell is very restrictive. Two different types of neighborhood are used. If the sister cell of the winning cell has no descendants (both sister cells are the only descendants of the ancestor node), the neighborhood includes the winning cell, the ancestor node and the sister cell, otherwise it includes only the winning cell itself.

A heterogeneity value for each cell is computed by its resource, R. This value will be used to direct the growth of the network by replicating, at the end of each cycle, the cell with the largest resource value. The resource is defined as the mean value of the distances among a cell and the image data profiles associated to it:

$$R_i = \frac{\sum_{k=1}^{K} d_{P_k C_i}}{K} \tag{3}$$

where the summation is over the K profiles associated to the cell i.

The criterion used for monitoring the convergence of the network is the total error, ϵ, which is a measure of how close the image data profiles are to their corresponding winning cell after an epoch. The error is defined as the summation of the resource values of all the cells that are being presented at the epoch t:

$$\epsilon_t = \sum_i R_i \tag{4}$$

Thus, a cycle finishes when the relative increase of the error falls below a given threshold:

$$\left| \frac{\epsilon_t - \epsilon_{t-1}}{\epsilon_{t-1}} \right| < E \tag{5}$$

The network follows its growing process by replicating the cell with the highest resource value. This cell gives rise to two new descendant cells and becomes a node. The values of the two new cells are identical to the node that generated them. The growing process ends when the heterogeneity of the system falls below a threshold. In our case we allow for the tree to fully expand by setting the threshold equal to zero.

3. ARTISTIC STYLE CLASSIFICATION AND AUTHORSHIP IDENTIFICATION

3.1 Motivation

Artists typically produce art in a series of movements or periods. Indeed, a common linkage exists not only between pieces in a specific artist's catalogue, but between pieces from an entire artistic period. Impressionism, cubism, and renaissance are examples of these periods, in which many works followed a particular set of rules or cues. There may also be different periods within a single artist's catalogue, like Pablo Picasso's Blue Period. Analysis of these catalogues can highlight themes in an artist's work over his or her lifetime, showing correlations between works that are produced decades apart.

Since art is subjective, the impressions that it induces often vary from person to person. This complicates the question of classification, because it increases the importance of choosing the right attributes for classification. We have chosen a set of simple and reasonable criteria for determining classification, including textural and color-based attributes. While it is difficult or impossible to classify based on actual painting subjects (since this would involve identifying what objects the image depicted), we can institute criteria that capture an overall feel of a painting or image. We hope to make it easier for people to find art that they like without sifting through thousands of paintings. Additionally, our classification system will allow us to study correlations between artists and artistic periods.

Using SOTA, neural networks allow us to classify large sets of images (pieces of art) into groups based on a set of extracted features or, roughly, themes. Each leaf node of the resulting output dendogram represents a single piece of art or a group of pieces, and each junction in the dendogram represents a common thematic 'ancestor' for all the images descending from that junction.

3.2 Feature Extraction and Sequence Building

Since the SOTA neural network was originally built for processing genetic sequences, we've used a genetic analogy in our image processing. The genetic sequences in this case are actually samplings of features extracted from each image. The goal is to build sequences that represent a number of low-level image features and then compare those sequences with the SOTA neural network to produce the overall image classification. We sample the overall image colors in red, green, blue, and monochrome (obtained from the standard RGB to luminance conversion, luminance = .3R + .59G + .11B), and also use a series of neighborhood texture comparisons as the features to build the image 'gene'. Additionally, color and monochrome gradients are sampled and used as features. All features can be scaled in importance by varying the number of samples included. Figure 2 illustrates the sampling procedure, which works as follows:

- Image processing techniques like those in Sections 3.2.1 - 3.2.3 are applied to the original image.
- A new buffer of the same size as the image is created and used to store the results of the image processing.
- When the new buffer is filled, it is sorted.
- The sorted buffer is sampled at regular intervals to extract a sequence that captures the distribution of values in the image.

The sampling method creates a non-decreasing discrete function of the image's pixel values (or a function of these values), and then samples this function to characterize the image.

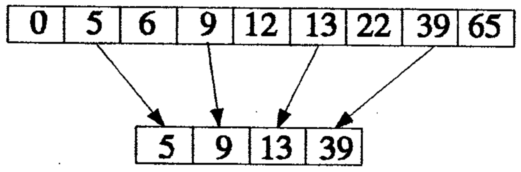


Fig. 2. This diagram demonstrates our sampling method. Samples are chosen at fixed intervals from a sorted list of pixel values and used to create a new sequence that approximates the original data. This particular diagram shows an array sampled with a sampling interval of 2.

3.2.1 Color Sampling

The color features of an image are extracted by doing regular sampling of a sorted array of the image's pixel values. Pixel values are either the image's red, green, or blue channels, or an extracted "luminance" channel. This gives a general simplified color distribution function for the image, and these functions can be compared element by element by their Euclidean distances in the SOTA.

3.2.2 Texture Sampling

Texture sampling is done in a similar way to color sampling, but the resultant elements instead represent the value of the difference between the processed pixel and the average value of its horizontal and vertical neighbors. For example, images with signifigant low level detail or rapid color change between pixels such as a picture of TV static would have a high value of textural complexity, whereas large regions of flat, slowly changing colors skew the distribution downward. An example is shown in Figure 4. This sampling compares fine grained texture complexity, and can be extended to coarser and coarser graining by resampling the image to lower resolutions, which is discussed in Section 3.2.4.

3.2.3 Gradient Sampling

The gradient sampling technique used is based on the Sobel operator, a set of convolution masks commonly used for edge detection. Our program detects gradients in the horizontal and vertical directions separately as well as the overall (non-axis-aligned) gradient. Sampling the sorted gradient magnitudes for data to add to our gene allows the classification scheme to reflect the distributions of low-level change in test images. For example, an image that is composed of a single flat color will have a constant (at zero) gradient function, an image with high gradients throughout will have a consistently large gradient function, and an image with large variations in its gradients will have a gradient function that represents a spectrum of gradients from small to large. These functions are compared as part of the overall image sequence in the SOTA neural network. Gradient sampling can be repeated at lower resolutions to produce feature sequences that focus on larger scale variations. A nice property of the Sobel operator is that its convolution masks reduce the effects of image noise, making gradient sampling in particular more resistant to noisy images.

3.2.4 Image Resampling

We have decided to do texture sampling on a number of different image resolutions in order to identify patterns within larger neighborhoods. Single pixel neighborhoods are not very representative of overall textures, since they consider such a small surrounding area at each pixel. We use a simple averaging scheme to resample the images by averaging a 2×2 square of the original image to obtain each resampled pixel. This halves the resolution of the image, as demonstrated in Figure 6. This allows us to process the images at a lower resolution, taking larger scale features into account.

3.2.5 Feature Scaling

The features that we extract are all added to the gene for each image and the genes are input into the SOTA for classification. In order to scale features in importance, the features can be sampled with a higher or lower density. For example, sampling the color at higher rates will make the color distribution more

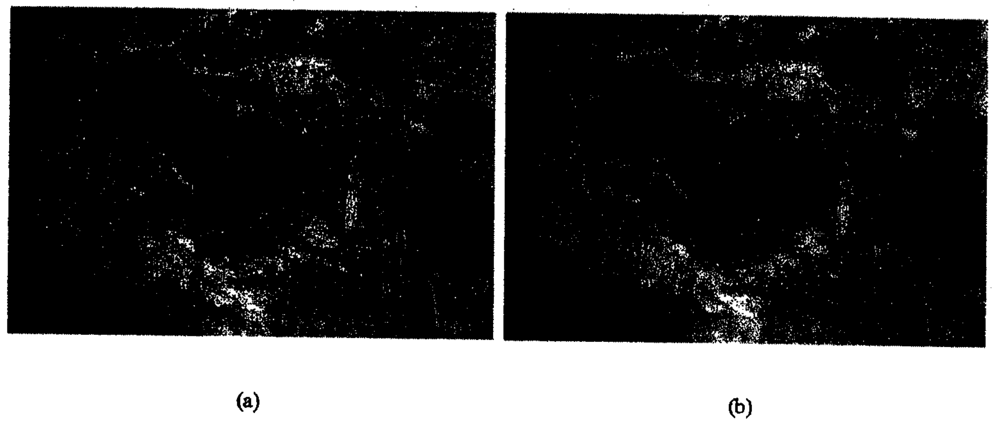


Fig. 3. Illustrates the conversion of an image to monochrome using standard luminance calculations. (a) shows the original image (*Old Town II* by Wassily Kandinsky) and (b) shows the image after conversion to monochrome.

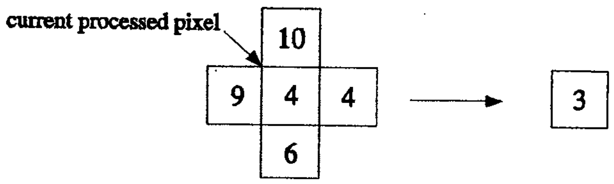


Fig. 4. Demonstrates how processed values are obtained in the texture sampling procedure. The processed pixel is compared with the average values of its neighbors, and the magnitude of the difference rounded to the nearest integer is used as the processed value. In this example, $result = abs(4 - ((10 + 9 + 6 + 4)/4))$, rounded to the nearest integer.

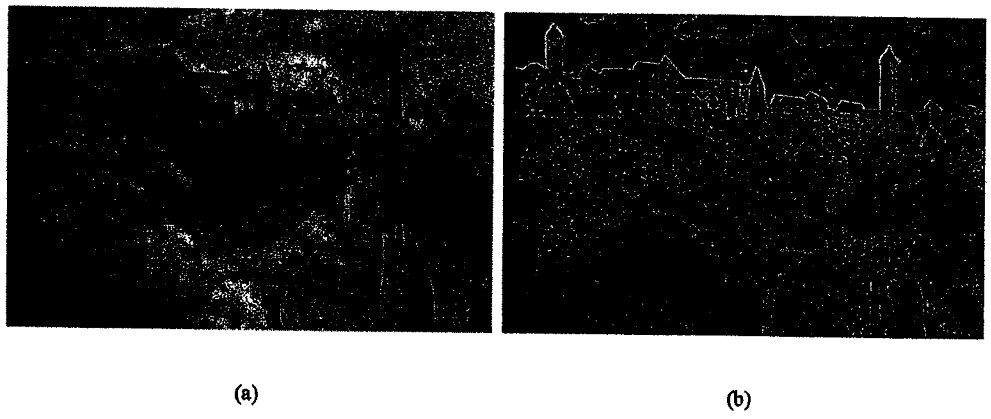


Fig. 5. Illustrates the conversion of an image to grayscale gradient image. The lightness of each pixel corresponds to the magnitude of the gradient at that pixel given by the Sobel operator. (a) shows the original image and (b) shows the gradient image after conversion.

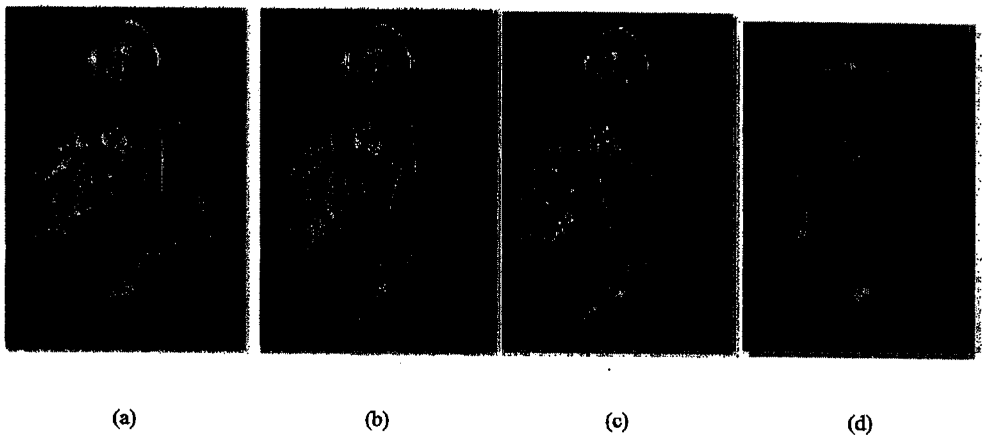


Fig. 6. Illustrates the usage of resampling an image multiple times, halving the resolution each time. (a) shows the original image (*Il Salvatore* by El Greco, (b) shows the original image resampled to half the original resolution, (c) shows the original image resampled to one quarter the original resolution, and (d) shows the original image resampled to one eighth the original resolution.

important in determining distances between two images. Figure 7 shows two sequences that scale feature importance differently. Another method of feature scaling is applying non-linear scaling to the individual values of the extracted feature sequences. This can be done in different ways for different features, and enhances or reduces differences between selected sequences in Euclidean space.

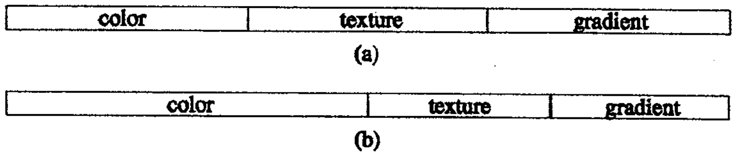


Fig. 7. Shows two differently scaled composite sequences to be input into the SOTA neural network. (a) gives equal weight to color, texture, and gradient properties, while (b) emphasizes color and gives texture and gradient less influence.

3.3 Data

The input images we used are selected from the works of Wassily Kandinsky. These paintings represent a large number of different color and texture combinations, as well as different artistic periods. These characteristics make the data useful for testing our classification system.

3.4 Experimentation and Results

The following experiments show and describe classification trees produced from image feature sequences built with different features and scalings of those features. A major difficulty in experimenting with our system was the difficulty in displaying the resulting trees. The figures showing our results were time-consuming to build, and this was a limitation in testing large image sets. Another difficulty we had was in comparing our results to what we "should" have seen. Our evaluation criteria were subjective since we didn't have a clear metric to measure our results against, but the classifications in our results should also appeal to the reader's intuitive reasoning about how images should be grouped.

3.4.1 First Experiment: Texture Sampling

For our first experiment we presented SOTA with a series of sequences generated by texture-sampling a series of randomly chosen Kandinksy paintings. A partial tree of results for this experiment is shown in Figure 8. The results are those that we were hoping for, with the tree showing distinct categories that break down based on the amount of high-frequency texture change in the corresponding images.

3.4.2 Second Experiment: Color Sampling

The second experiment was to present SOTA with the same series of images as in the first experiment, sampling color rather than texture. Each image was sampled 100 times based on its red channel only. Results for this experiment are shown in Figure 9. This experiment also showed promising results, dividing the images into visibly different color categories.

3.4.3 Third Experiment: Color and Texture Sampling

In the third experiment we presented SOTA with image sequences consisting of a new set of color and texture sampled Kandinsky paintings chosen at random. Each image was sampled 100 times based on its blue channel and its texture. Results for this experiment are shown in Figure 10. The results show that the images are now partitioned based on intuitive visual differences in both color and texture complexity.

3.4.4 Fourth Experiment: Gradient and Texture Sampling

This experiment uses the same set of images presented in the previous experiment, but the sequences are built by sampling image gradients and texture 100 times each. The results were similar to those in the previous experiment, but with a key difference. No direct color sampling was used in the fourth experiment, but the results show classifications that appear to be based on color differences. Although this could be a coincidence, we reason that it is more likely due to overlap between features. Since the gradient and texture features use monochrome color, and this is dependent upon the images three color channels, these features contain some of the information that is provided by straightforward color sampling. In the case of this experiment, this information was enough to preserve many of the classifications produced by experiment three.

4 CONCLUSIONS AND FUTURE WORK

As can be seen in Figures 8 through 12, our modified SOTA neural network was relatively successful in classifying images into groupings of similar images. However, one drawback of using a SOTA neural network to perform image classification is that there isn't a clearly defined metric for measuring the correctness of a resulting classification. "Correctness" is based primarily upon the user's opinions of the classification results, even though it can be shown that our classifications are correct based on the features that were extracted. A major goal of future work will be to reconcile user preferences with the set of features to be extracted by the image processor. Although our work at this point reflects only simple, low-level features, it may become necessary in the future to use more complex feature extraction techniques. As it is, we have provided a content-based image classification system with a framework for scaling features to fit the tastes of individual users.

In addition to the experiments that were run, future tests should use a large range of images and explore different feature scalings. The difficulties that we had in displaying the SOTA output (which involved manually manipulating images, nodes, and connections) translated into difficulties in analyzing results and making improvements. In the future, it will be necessary to look into tools for quickly and automatically displaying the trees described in the program output, in order to speed up experimentation and analysis.

One interesting area of research is reoccurring themes over time in an artist's body of work. Figure 11, which was generated by color and texture sampling, shows two images, assigned to neuron 4, created 23 years apart from each other. *Picture With White Border* painted in 1913 and *Dominant Curve* painted in 1936, both by Wassily Kandinsky, have been grouped together due to their similar textural and color characteristics. This observation can allow us to draw further conclusions that the artist, in 1936, revisited a theme that existed in his work around 1913. By generating SOTA neural networks, themes and style patterns can be extracted allowing users the ability to organize an artist's body of work, not chronologically, but by style and theme.

REFERENCES

[1] J. D. A. J. M. Carazo. Phylogenetic reconstruction using an unsupervised growing neural network that adopts the topology of a phylogenetic tree. *Journal of Molecular Evolution*, 44:226–233, 1997.

[2] T. Kohonen. The self-organizing map. In *Proc. IEEE '78*, pages 1464–1480, 1990.

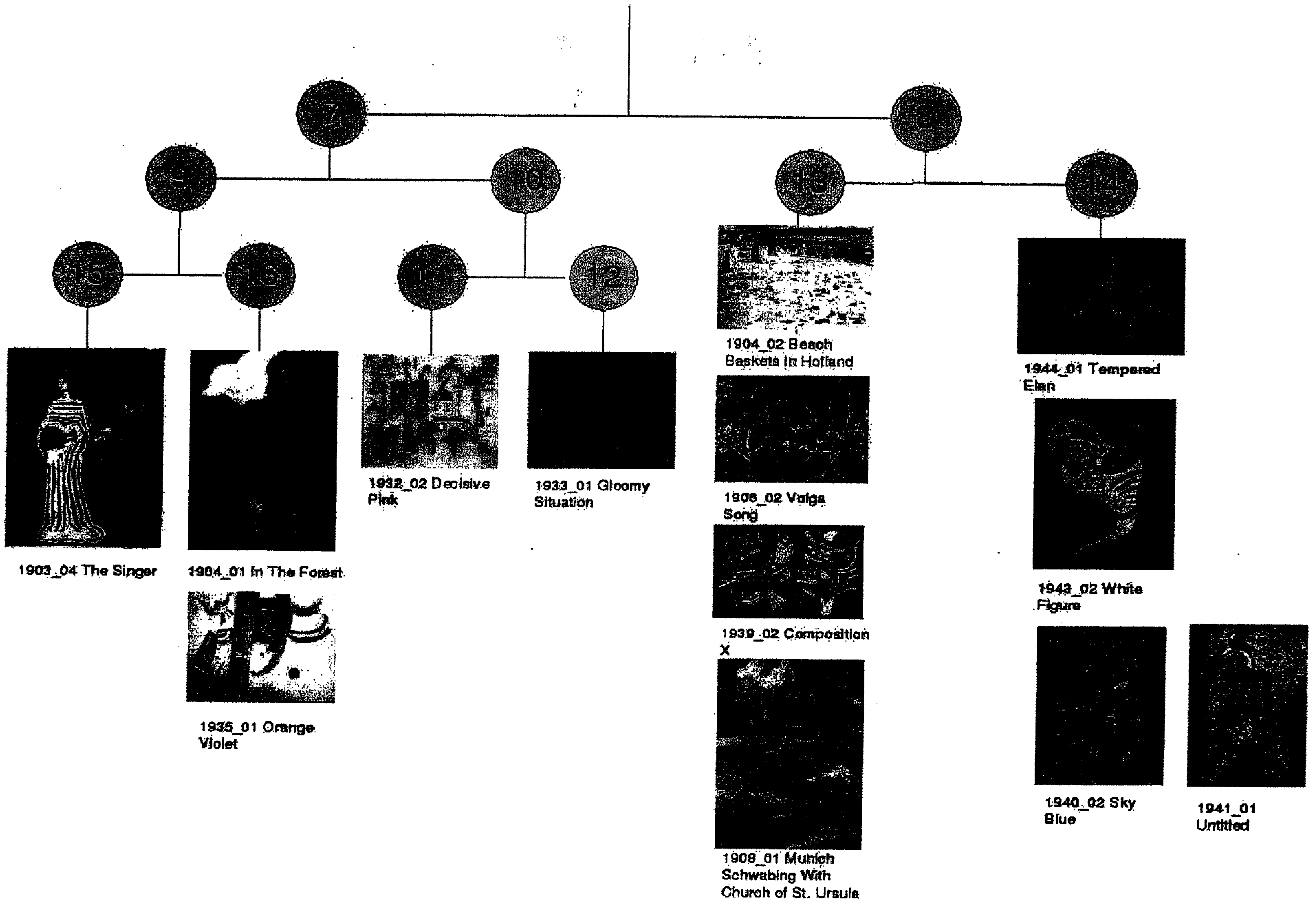


Fig. 8. Shown here is the resulting SOTA neural network using texture sampling with a variability threshold of 20 and a resource of 0. Each image was sampled 100 times. Note that the images *1940_02 Sky Blue* and *1941_01 Untitled* in neuron 14, which have been placed together, both share similar textual qualities of small figures in a single color background. All paintings are by Wassily Kandinsky.

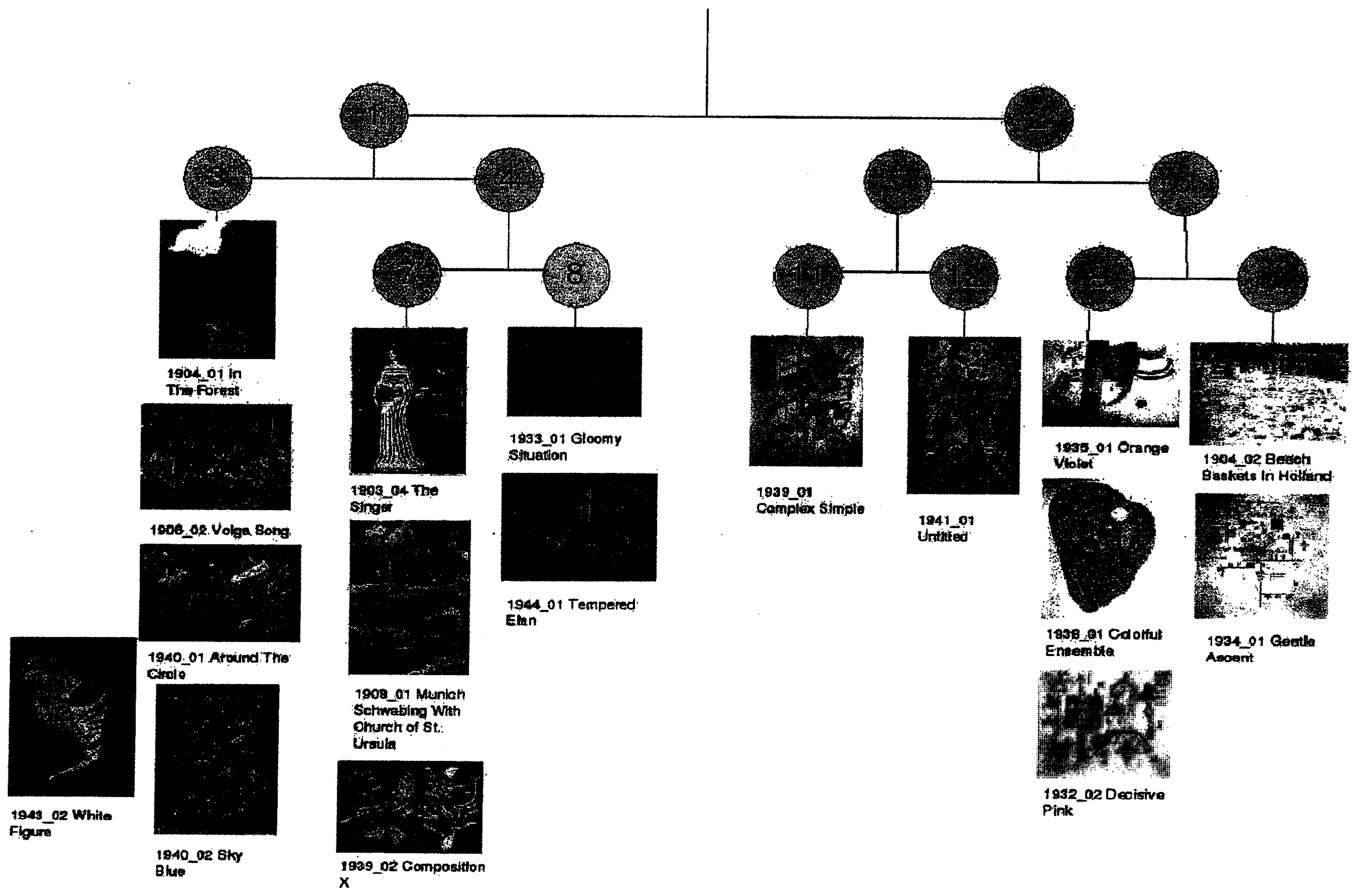


Fig. 9. Shown here is the resulting SOTA neural network using red color sampling with a variability threshold of 200 and a resource of 150. Each image was sampled 100 times. Note that images on the left of the dendrogram (neuron 1) all have a higher color saturation and images on the right of the dendogram (neuron 2) all have a lower color saturation. All paintings are by Wassily Kandinsky.

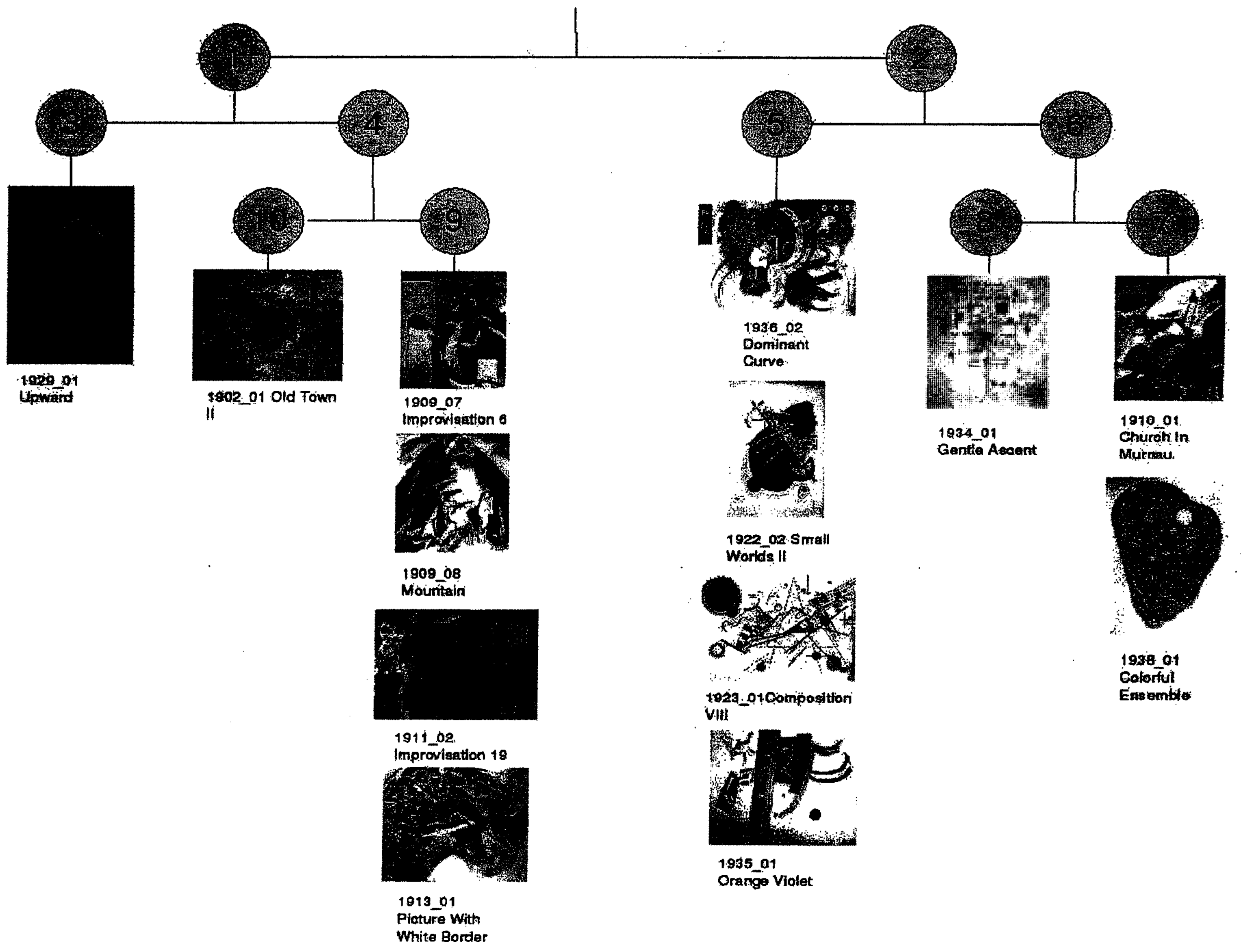


Fig. 10. Shown here is the resulting SOTA neural network using blue color sampling combined with texture sampling with a variability threshold of 200 and a resource of 150. Each image was sampled 100 times per feature. Note that images here, as in Figure 9, have been sorted to the left and right of the dendogram corresponding to their color saturation. Note also that images have been placed based on textual similarities. For example, the images *1909_07 Improvisation 6* and *1909_08 Mountain* in neuron 9 are grouped together because they have

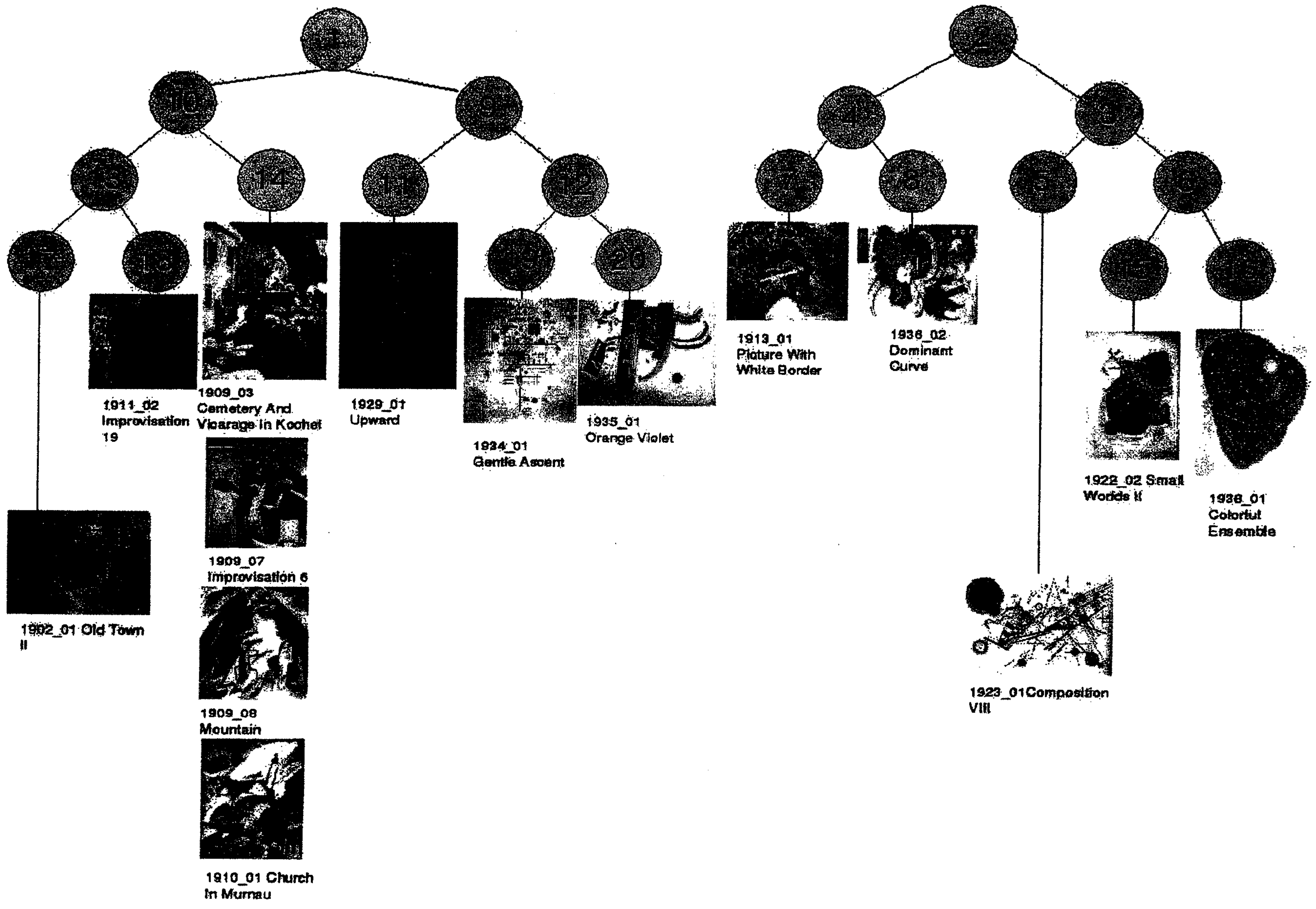


Fig. 11. Shown here is the resulting SOTA neural network using gradient sampling combined with texture sampling with a variability threshold of 0 and a resource of 150. Each image was sampled 100 times per feature. This network is very interesting because the image *1910_01 Church In Murnau* has been assigned to neuron 14, along with the images *1909_07 Improvisation 6*, *1909_08 Mountain* and *1909_03 Cemetery And Vicarage In Kochel*, according to its gradient and texture but not color. However, if you refer to the network in Figure 10, which is generated

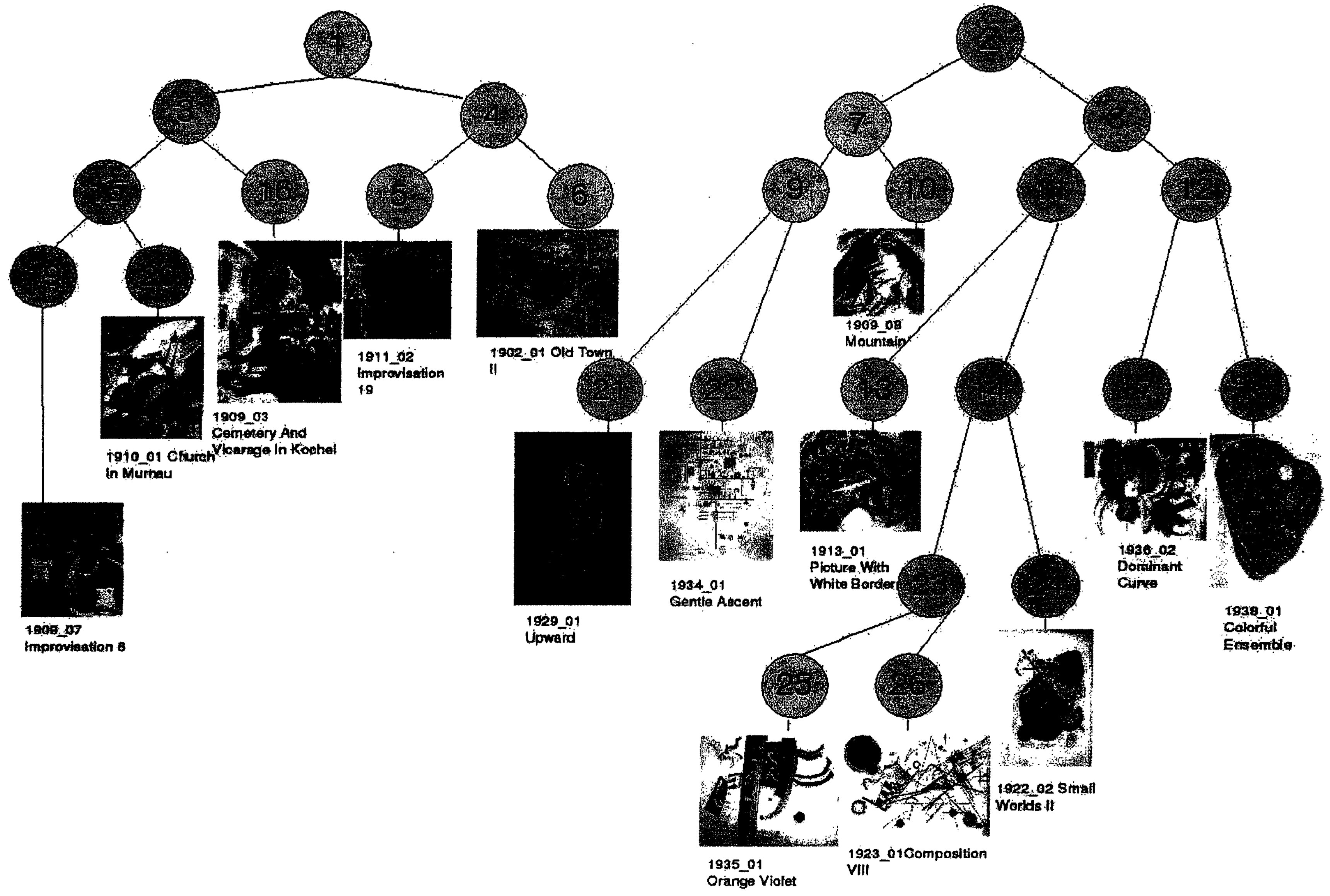


Fig. 12. Shown here is the resulting SOTA neural network using monochrome texture sampling at three levels of resolution with a variability threshold of 100 and a resource of 150. Each image was sampled 100 times per level of resolution. All paintings are by Wassily Kandinsky.

Modeling Memory Consolidation to study the Effects of Trauma-induced Amnesia and to simulate Categorical Learning

Bhavna Kapoor
Department of Computer Science
Yale University, New Haven,
Connecticut 06511

Abstract

Any neural network modeling of the intricate process of memory consolidation involving the neocortex and the hippocampus must account for the fact that learning in the brain is an on-line, incremental, adaptive and life long process. The Adaptive Resonance Theory (ART) class of models lend themselves well to these requirements. We use this neural network paradigm to model a rudimentary memory consolidation system in the form of the ART1-Memory Consolidation (ART1-MC) model. Further, we simulate traumatic brain injury and study the response of the model to trauma and compare it with the observed, clinical effects of retrograde and anterograde amnesia. We also enhance the model to support categorical learning from a general to a specific domain.

Keywords - adaptive resonance theory, memory consolidation, trauma, anterograde amnesia, retrograde amnesia, general to specific learning

1. INTRODUCTION

The hypothesis that new memories in the brain consolidate slowly over time was proposed 100 years ago, and continues to guide memory research. In modern consolidation theory, it is assumed that new memories are initially 'labile' and sensitive to disruption before undergoing a series of processes (e.g., glutamate release, protein synthesis, neural growth and rearrangement) that render the memory representations progressively more stable (Haist et. al., 2001). It is these processes that are generally referred to as "consolidation". The principal agent in memory consolidation research, in terms of brain regions, is the hippocampus. The hippocampus is involved in the consolidation of contextual memories into various parts of the neocortex, and is part of a region called the medial temporal lobe (MTL), that also includes the perirhinal, parahippocampal, and entorhinal cortices. Lesions in the medial temporal lobe typically produce amnesia characterized by significant loss of recently acquired memories as compared to normal memory. This has been interpreted as evidence for a memory consolidation process ([15]).

Understanding the complex dynamics of human memory consolidation will be informed by extensive and accurate computer simulations of brain process models. Research into the development of such neural network models is motivated by the fact that it is not always possible to obtain experimental data with regards to human brain functioning, and that neural network models could provide important insight that might benefit our understanding of memory disorders such as Alzheimer's disease.

Trauma-induced amnesia models have active use in building computerized medical decision support systems for traumatic brain injury to support surgeons in making difficult medical decisions; for instance, recommending a risky open-skull surgery to a patient (Li Y.C. et. al., 2002). Using neural network models to simulate categorical learning can help in evolving advanced, unsupervised learning systems crucial to research in artificial intelligence, machine learning and robotics.

In this report, we study and enhance an Adaptive Resonance Theory (ART) based neural network model called the Adaptive Resonance Theory-1 Memory Consolidation model (ART1-MC). Section 2 introduces the architecture and working of the ART model as proposed in (Carpenter G. & Grossberg S., 1986) . Section 3 discusses the details of modeling trauma on ART1-MC model and the results obtained. In Section 4, we enhance the ART1-MC model to the ART1- Memory Consolidation with Categorical Learning (ART1-MCCL) model and discuss the working and results of this model. Finally, in Section 5, we summarise the results obtained in the report.

2. MODELING OF MEMORY CONSOLIDATION

Any neural network strategy that attempts to model the intricate and partially understood process of memory consolidation involving the neocortex and the hippocampus must account for the fact that learning in the brain is an on-line, incremental, adaptive and life long process. The Adaptive Resonance Theory (ART) class of models lend themselves well to these requirements.

2.1 Adaptive Resonance Theory (ART)

Adaptive Resonance Theory (ART) was developed by Gail Carpenter and Stephen Grossberg during their studies of the behavior of models of networks of neurons. The ART paradigm can be described as a type of incremental clustering. It has the ability to learn without supervised training and is consistent with cognitive and behavioral models. It is an unsupervised clustering paradigm based on competitive learning that is capable of automatically finding categories and creating new ones when they are needed.

ART was developed to solve the learning instability problem suffered by standard feed-forward networks. The weights, which have captured some knowledge in the past, continue to change as new knowledge comes in. There is therefore a danger of degrading, even losing the old knowledge with time. The weights have to be flexible enough to accommodate the new knowledge but not so much so as to lose the old. This is called the *stability-plasticity dilemma* and it has been one of the main concerns in the development of artificial neural network paradigms.

ART architecture models can self-organize in real time producing stable recognition ability while getting input patterns beyond those originally stored. ART is a family of different neural architectures. The first and most basic of these is ART1 , which can learn and recognize binary patterns. ART2 is a class of architectures categorizing arbitrary sequences of analog input patterns. ART3 and ARTMAP are generally more complex while employed in specific problem domains such as invariant visual pattern recognition, where biological equivalence is discussed.

The ART architectures use a combination of feedback, higher-level control, and nonlinearities to form regions in state space that correspond to concepts, embodied in the statistical structure of the input. The earlier ART models developed grandmother cells at an output layer. New patterns to be classified were judged either as new or as new examples of old concepts. If they were sufficiently far away from previously classified patterns in the state space, new grandmother cells were formed. On the other hand, if they were within a certain distance from a previously classified pattern, they would be interpreted as prototypes of known concepts. As such, ART models employ a combination of bottom up (input-output) competitive learning with top-down (output-input) feedback learning.

An ART system consists of two components: an attentional subsystem and an orienting subsystem. The stabilization of learning and activation occurs in the attentional subsystem by matching bottom-up input activation and top-down expectation values. The orienting subsystem controls the attentional subsystem when a mismatch occurs (in the attentional subsystem). In other words, the orienting subsystem works like a novelty detector. This structure leads to four basic properties of ART models:

1) Self-scaling computational units

The attentional subsystem is based on competitive learning, enhancing pattern features but suppressing noise.

2) Self-adjusting memory search

The system can search memory in parallel and change its search order adaptively.

3) Direct category access

Already learned patterns directly access their corresponding category.

4) Adaptive modulation of attentional vigilance

The system can adaptively modulate attentional vigilance using the environment as a teacher.

2.2 Overview of the ART1-Memory Consolidation (ART1-MC) model

2.2.1 Architecture

As in figure 1, the ART1-MC model consists of 2 processing levels F1 and F2. Level F1 contains a network of nodes, each of which represents a particular combination of sensory features. Level F2 contains a network of nodes that represent recognition codes or categories that are selectively activated by the activation patterns across F1. The attentional subsystem consists of gain control parameters that enable F1 to

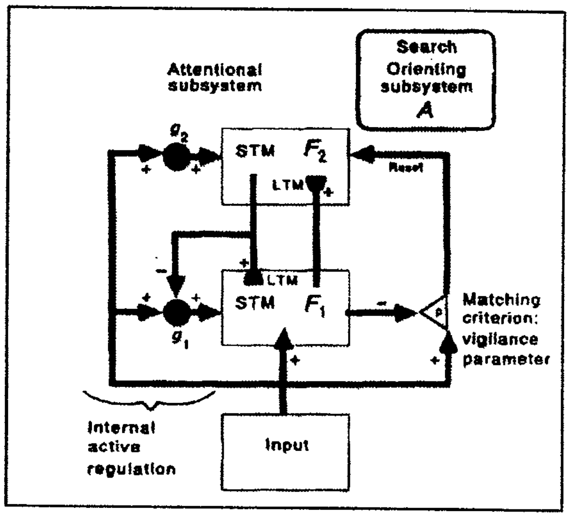

Attentional subsystem
Search Orienting subsystem A
g_2
STM F_2
LTM
Reset
LTM
STM F_1
g_1
Matching criterion: vigilance parameter
Internal active regulation
Input

Fig 1. Block diagram of ART1-MC model

distinguish between bottom-up input patterns and top-down priming or template patterns. The orienting subsystem is responsible for generating a reset wave to F2 when there is a mismatch of bottom-up and top-down patterns at F1. It is controlled by the vigilance parameter.

2.2.2 Working

The interactions between the attentional and orienting subsystems of ART1-MC selfstabilize learning, without an external teacher as the network familiarizes itself with an environment by categorizing the information within it in a way that leads to stabilizing the performance of the network over time.

Level F1 encodes a distributed representation of an event by a short-term memory (STM) activation pattern across a network of feature detectors A,B,C,D. Level F2 encodes the event using a compressed STM representation of the F1 pattern. Learning of these recognition codes takes place at the long-term memory (LTM) traces within the bottom-up and top-down pathways between levels F1 and F2. The top-down pathways read out learned expectations whose prototypes are matched against bottom-up input patterns at F1. The size of mismatches in response to novel events are evaluated relative to the vigilance parameter rho of the orienting subsystem A. A large enough mismatch resets the recognition code that is active in STM at F2 and initiates a memory search for a more appropriate recognition code. Output of subsystem A can also trigger an orienting response. If the expectation is close enough to the input exemplar, a state of resonance develops as the attentional focus takes hold. This resonant state, rather than the bottom-up activation, is what drives the learning process. The resonant state persists long enough, at a high enough activity level to activate the slower learning process. The system learns prototypes, rather than individual exemplars.

2.2.3 Implementation Details

As the ART1-MC model belongs to the ART1 category, it is capable of processing only binary inputs. The input set is a sequence of patterns of random binary values. The length of each pattern is 5. The length of the input sequence ranges from 25 to 200. The output of the system is an input-output mapping from a given input pattern to a class. The system has a fixed number of classes. This number is parameterized and ranges between 10 to 35. The system does not dynamically create new classes to account for novel inputs. If it encounters a new input pattern outside the categories it knows, that pattern simply triggers a "novel input found" response from the system. The vigilance parameter Rho is set to 0.9. The ART1-MC system has been implemented using ANSI C on the the TurboC compiler.

3. SIMULATING TRAUMA-INDUCED AMNESIA

Possible causes of amnesia in humans include dementia, alzheimer's disease, infection, brain tumor, brain injury and certain metabolic disorders. Our focus however, is to specifically and restrictively study the effects of amnesia caused by sudden, localized injury (henceforth referred to as trauma) to the parts of the brain believed to be the major players in the task of memory consolidation, i.e., the hippocampal formation. and the neocortex (Cohen et. al., 1993).

Clinical symptoms of amnesia in humans include memory loss of events prior to trauma (retrograde amnesia), inability to lay down new memory of events after trauma (anterograde amnesia), failure of consolidation, tendency to learn the first event in a series and abnormal reactions to novelty (Grossberg & Merrill, 1994).

3.1. Modeling Trauma

Recall that in the ART1-MC model of declarative memory consolidation discussed in Section 2, the orienting subsystem plays the role of the hippocampal system while the attentional subsystem represents aspects of the inferotemporal cortex.

To model traumatic injury in the ART1-MC model, we develop 2 functions that cause localized random disruption in either the orienting or the attentional subsystems. As trauma can occur at any point in the learning cycle, a measure of randomness is attached to when the trauma function is triggered in the model during the training cycle. The nature of the input to the system remains the same as discussed in Section 2. Two different and independent input sets have been used and each is tested under normal as well as trauma conditions. The input-output pairs from pre and post trauma test cycles are compared to draw inferences about the change in behaviour of the system.

3.1.1 Trauma to the Orienting Subsystem

The orienting subsystem, by means of the vigilance parameter (rho) calibrates how much novelty the system can tolerate before activating the orienting subsystem to reset mismatched categories and to select better F2 representations with which to learn novel events at F1. This, without risking unselective forgetting of previous knowledge. Trauma to the orienting subsystem randomly resets the vigilance parameter, the value of which determines the extent of amnesiac damage.

3.1.2 Trauma to the Attentional Subsystem

The attentional subsystem is characterized by the properties of layer F2. In the ART1-MC model, trauma to the attentional subsystem is achieved by randomly disrupting the weights, i.e., the STMs associated with F2, and also by resetting the L parameter of the net, since this is involved in the initialization and the weight adjustment functions for F2.

3.2 Experimental Results

3.2.1 Experiment 1

Aim : To study the effect of trauma on the Orienting Subsystem.

Training set : 3 sets of fixed-size input sequences of lengths 25, 50 and 100. The input sequences consisted of 30 random combinations of binary values. The size of each pattern was 5. The number of classes was fixed to 30. The vigilance parameter was set to 0.9.

Output before trauma : Each input pattern was classified by the network into one of the 30 classes. The output class for each unique input pattern was recorded.

Trauma : The same input sequences were used to train the network but the network was traumatized (i.e. the vigilance parameter was reset to a random value) at roughly the mid-point of each input sequence (i.e. 12, 25, 50). The new value of the vigilance parameter after trauma was recorded.

Output after trauma : The output class for each unique input pattern was recorded. The outputs before and after trauma were compared for each input pattern and the number of misclassified patterns was recorded for each of the 3 input sequences of lengths 25, 50 and 100. The percentage error in classification was calculated and plotted against the corresponding value of the vigilance parameter after trauma to give the plot in figure 2.

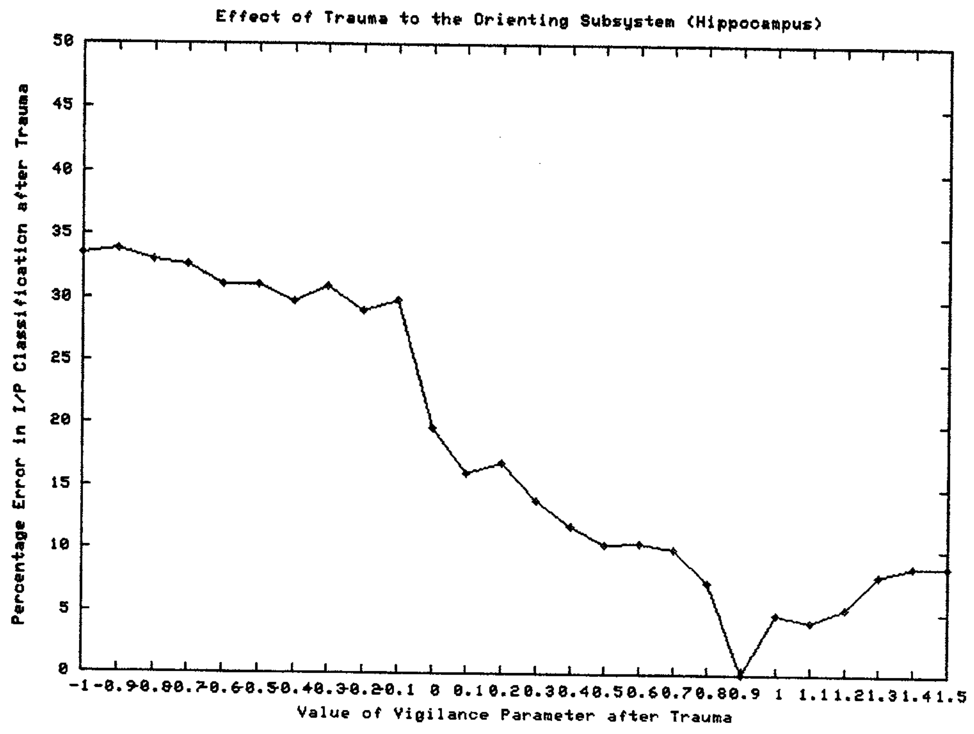


Fig. 2 Experiment 1

3.2.2 Experiment 2

Aim : To study the effect of trauma on the Attentioning Subsystem.

Training set : 1 set of fixed-size input sequence of length 50. The input sequence consisted of 30 random combinations of binary values. The size of each pattern was 5. The number of classes was fixed to 15. The vigilance parameter was set to 0.9. The number of weights in level F2 was fixed to 15. The value of the L parameter was set to 1.

Output before trauma : Each input pattern was classified by the network into one of the 30 classes. The output class for each unique input pattern was recorded.

Trauma : The same input sequence were used to train the network but the network was traumatized (i.e. n (0 <= n <= 15) weights in level F2 were randomly reset to a value between 0 and 1) at roughly the mid-point of each input sequence (i.e. 25). The number of weights distorted was recorded.

Output after trauma : The output class for each unique input pattern was recorded. The outputs before and after trauma were compared for each input pattern and the number of misclassified patterns was recorded. The number of misclassified input patterns after trauma was plotted against the number of F2 level weights distorted as shown in figure 3.

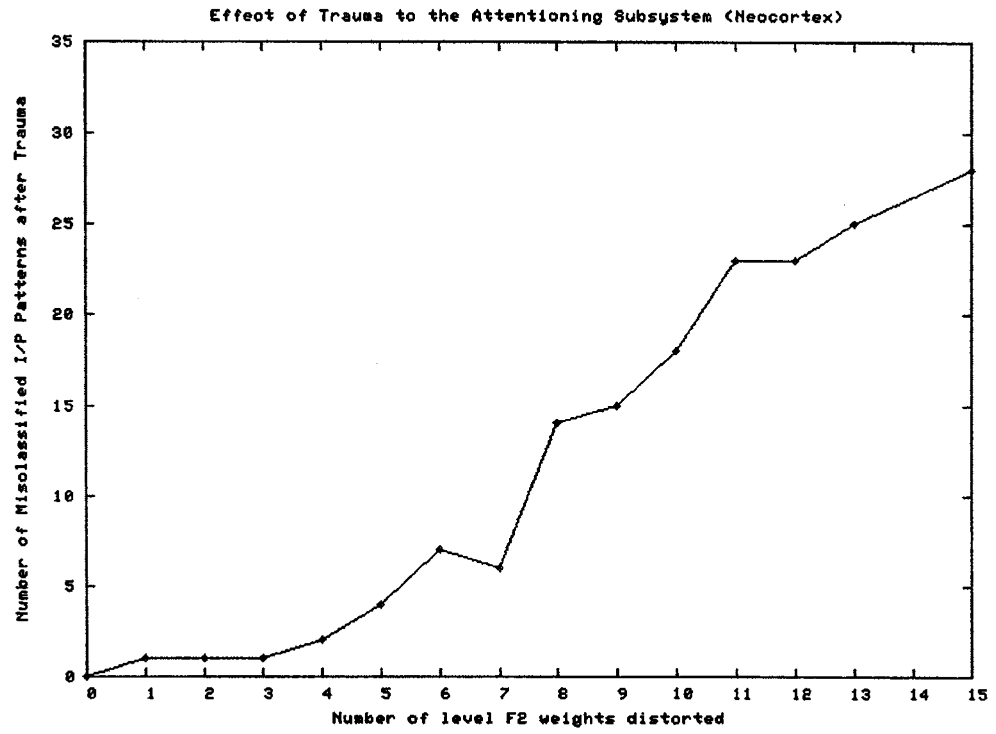


Fig. 3 Experiment 2

3.3 Effects of Trauma on the ART1-MC and their Implications

The following observations have been made based on the experiments in Section 3.2.

3.3.1 Effects of Trauma on the Orienting Subsystem

When the effective value of the vigilance parameter rho after trauma is within the bounds 0 and 1, *mild amnesia* occurs. A limited form of retrograde amnesia and a slightly more pronounced anterograde amnesia is observed. The reason for the limited nature of the former is that once consolidation has taken place, the search process for familiar events is progressively decoupled from the orienting subsystem, and thereby the disrupted value of the vigilance parameter has little effect on the recall of well consolidated events. This corroborates the widely held hypothesis in neuropsychology that the hippocampus mediates the gradual formation of neocortical memory representations.

The unlimited anterograde amnesia is seen because with a corrupted vigilance parameter, the network cannot correctly carry out the memory search to learn a new recognition code.

Another relevant observation is that anterograde amnesia in this case is only temporary. With further input samples, the orienting subsystem is able to re-stabilize itself relatively quickly. However, its behaviour may not be the same as it was before trauma. More specifically, if the post-trauma value of the vigilance parameter is small i.e. between 0 and 0.4, the underlying recognition system loses the ability to be specific and makes broad generalizations and abstract prototypes. Likewise, if the resultant value of the vigilance parameter is large, typically between 0.75 and 1, the recognition system becomes more prone to making narrow generalizations and to prototypes that represent fewer input exemplars, even a single exemplar. This behaviour of the system is consistent with clinical effects of mild amnesia wherein the subject is able to generally classify and identify events but has trouble dealing with details or vice-versa and remains in a state of confusion for some time after the trauma.

It is also seen that if the vigilance parameter becomes greater than 1, *severe amnesia* (both retrograde and anterograde) sets in. Also, the network takes longer to re-stabilize and resume normal memory function.

3.3.2. Effects of Trauma on the Attentional Subsystem

Trauma to the F2 layer parameters results in the inability of the system to be able to consistently classify input patterns. As the degree of distortion of the weights in F2 increases, the system is unable to recognize patterns it had seen before the trauma and classifies these as novel inputs. Changes in the L parameter of the network however do not degrade the performance of the system significantly unless the value of L crosses a threshold value 5.

The observations on the response of the attentional subsystem to trauma show that it closely resembles the symptoms of clinical retrograde amnesia caused by traumatic injury to the neocortical region i.e. the site of long term memory.

4. SIMULATING CATEGORICAL LEARNING

The perception and identification of various categories is an important aspect of learning processes. The ability to divide the world into structured categories emerges early in infancy and continues to be refined by experience and learning throughout human life. For example, infants as young as four months old can classify various domestic cats into a single category – correctly excluding horses and tigers – although they can make the mistake of including female lions. By seven months, with the development of finer-scale visuo-perceptual categorization skills, the category 'cat' is well enough differentiated to exclude female lions. (Hasegawa et. al., 2002) Behavioral and neurophysiological evidence shows that the neural mechanisms of categorization learning may be embedded in the inferior temporal cortex, a part of the neocortex.

4.1Modeling Categorical Learning

The ART1-MC model can be easily adapted to simulate categorical learning. We call the new model that we develop to do so the ART1-Memory Consolidation with Categorical Learning model (ART1-MCCL).

4.1.1 Architecture

The general architecture of this model is the same as ART1-MC. To specialize the model for categorization support, the attentional subsystem in ART1-MCCL is used to represent the inferior temporal (IT) cortex, an area of the neocortex previously implicated in representing visual objects and subject to flexible adaptation in different learning paradigms. As shown in figure 4, the model has, in addition to the attentional and orienting subsystems, a categorization subsystem which is controlled by a gain function g_3.

4.1.2 Working

During the learning and consolidation process, the categorization subsystem maintains a sparse associative map of the input patterns and their corresponding feature map categories. Every feature map category in the categorization subsystem corresponds to a unique input pattern. If the input A to the F1 level is similar (but not the same) as a pattern B which the network has already seen and classified, both

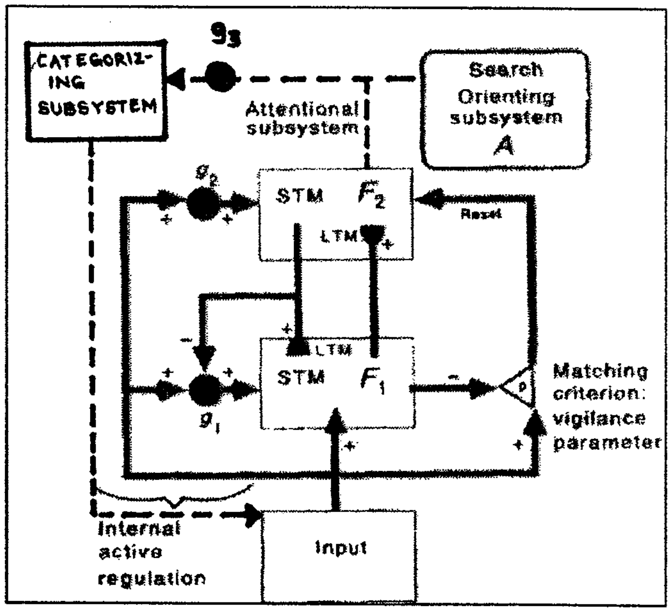

CATEGORIZ-ING SUBSYSTEM
g_3
Attentional subsystem
Search Orienting subsystem A
g_2
STM F_2
LTM
Reset
LTM
STM F_1
g_1
Matching criterion: vigilance parameter
Internal active regulation
Input

Fig. 4 Block diagram of ART1-MCCL model

inputs essentially map to the same class and a clash occurs. This prompts the categorization subsystem to explicitly reclassify the previous input A through an explicit feedback input loop. In the process, the attentional subsystem which now has knowledge of pattern B discovers prominent distinguishing features between A and B and creates subcategories for the 2 inputs under the same general class heading.

4.2 Experiment

To show that the ART1-MCCL model is capable of categorical learning, it was tested on randomly generated sequences of fixed-length patterns. The vigilance parameter rho was set to 0.9. Following is a sample test case of size 35 (patterns named A-M and represented as red and yellow boxes in figure 5) and the corresponding classes and subclasses (1-11), represented as ovals in figure 5 :

Input Sequence :	***o*	A	ooooo	J
	**o*o	B	o****	H
	****o	C	*o***	K
	**o*o	B	**o**	L
	****o	C	***o*	A
	**o*o	B	****o	C
	****o	C	**o*o	B
	*oo*o	D	*oo*o	D
	*oo**	E	*oo**	E
	*oo*o	D	ooo**	F
	*oo**	E	oo***	E
	ooo**	F	oooo*	I
	oo***	G	ooooo	J
	o****	H	oooo*	I
	oo***	G	o****	H
	ooo**	F	ooooo	J
	oooo*	I	oooo*	I
			**ooo	M

Result : Figure 5 shows the output generated for the above input sequence. The oval nodes show the class number and the rectangular nodes are the individual input patterns. Note that for this particular example, the ART1-MCCL model has been able to successfully categorize and subcategorize all but 2 of the input patterns (M and J). This can be attributed to the fact that these 2 patterns occur towards the end of the training sequence. So the network has not seen enough samples of M and J to be able to deal with them.

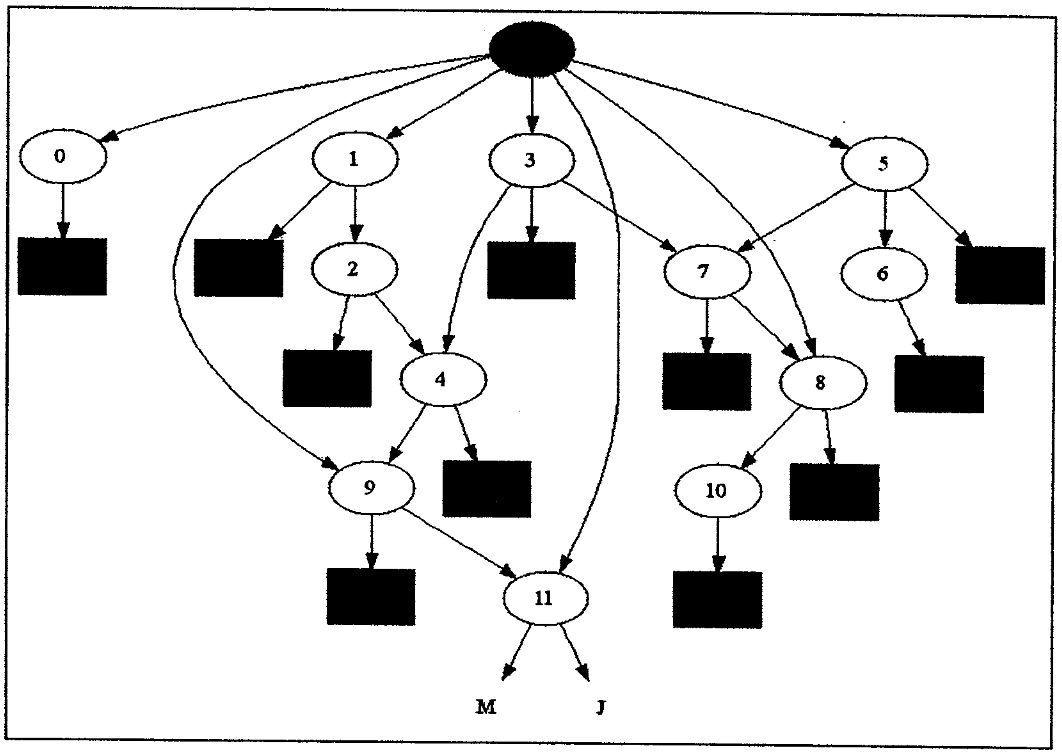


Fig. 5 Result of Training with Subcategorization

5. CONCLUSION

The ART1-MC model was chosen over it's counterparts such as backpropagation and multi-layer feedforward networks to model memory consolidation because it supports incremental learning and thereby circumvents the stability-plasticity dilemma. This model when subjected to traumatic injury to the attentional or the orienting subsystems has shown behavior which is consistent with the observed and experimental effects of trauma-induced amnesia. Not only is this further indication of the ART1-MC model's ability to correctly model brain functions, but it is also a clue that Adaptive Resonance Theory models are effective for neurobiological modeling. The study of trauma-induced amnesia shows that memory consolidation over a period of time does take place and it also gives an insight into how the hippocampus and the neocortical regions might be interacting with each other to achieve this. Likewise, the ART1-MCCL model provides a reasonable simulation of categorical learning and lends further credence to the behavioral and neurophysiological evidence that the IT cortex plays an important role in categorization tasks.

5. REFERENCES

1. Alvarez R. & Squire L.R. (1994). *Memory consolidation and the medial temporal lobe : a simple neural network model.* Proceedings of National Academy of Scienes (USA), V.91, PP. 7041-7045.

2. Carpenter G.A. & Grossberg S. (1986). *A massively parallel architecture for a self-organizing neural pattern recognition machine.* Computer vision, graphics and image processing (1987), v.37, pp. 54-115.

4. Cohen, Neal J., and Howard Eichenbaum. *Memory, Amnesia, and the Hippocampal System.* Cambridge, Massachusetts: MIT Press, 1993.

5. Clark R.E., Broadbent N.J., Squire L.R. & Stuart M.Z. (2001). *Anterograde amnesia and temporally graded retrograde amnesia for a nonspatial memory task after lesions of hippocampus and subiculum.* Journal of neuroscience (2002), v.22(11), pp. 4663-4669.

6. Grossberg S. & Merrill J.W.L. (1994). *The hippocampus and cerebellum in adaptively timed learning, recognition, and movement.* Journal of cognitive neuroscience (1996), v.8, pp. 257-277.

7. Haist, F., Gore, J.B. & Mao, H. 2001. Consolidation of human memory over

decades revealed by functional magnetic resonance imaging. *Nature neuroscience*, 4 (11), 1139 - 1145.

8. Hasegawa I. & Miyashita Y., *Categorizing the world : expert neurons look into key features*, Nature Neuroscience (2002), v.5.2, pp. 90-91.

9. Hasselmo M.E. & McClelland J.L., *Neural models of memory*. Current opinion in neurobiology (1999), v.9, pp.184-188.

10. Li Y.C, Liu L, Chiu W.T, Jian W.S. *Neural network modeling for surgical decisions on traumatic brain injury patients*. International Journal of Medical Informatics (2002), Volume 57, Issue 1, pp. 1-9.

11. Meeter M., *Control of consolidation in neural networks: avoiding runaway effects*. Connection science (2003), v.15(1), pp. 45-61.

12. Pandya. A. & Macy R., *Pattern Recognition with Neural Networks in C++*, CRC Press (1996).

13. Sigala N. & Logothetis N.K., Nature (2002) , v.415, pp. 318-320.

Web Resources

13. CELEST Technology website, Boston University
http://cns.bu.edu/techlab

14. Karsten Kutza. *Neural Networks at your Finger Tips*
http://www.neural-networks-at-your-fingertips.com/

15. About Memory : Learning about memory for permanent memory improvement
http://www.memory-key.com/NatureofMemory/consolidation.html

Learning The Distances Between Pairs of Sensors on the Retina

Edo Liberty
Department of Computer Science, Yale University

Abstract
This paper describes an experiment done with a neural net. The architecture of which, is based on the model of the retina and the visual cortex. The net used was a hybrid of a feed-forward net (first layer) and a Hebbian net (second layer). The first layer models the sensors on the retina by calculating one value for each pixel in the image, using the brightness of the pixel and it's neighboring pixels. Each value is thought of as the output of one independent sensor. The location of the sensor is defined to be the location of its central pixel (see Figure 2). The sensor outputs are then fed, with out regard of their original position in the image, to the second layer, which learns the map between the sensor outputs and their original position in the image, using Hebbian learning.

1. INTRODUCTION

How does the brain map the information received in the visual cortex to a specific physical locations on the retina? It is obvious that this process is performed, since we can easily tell a straight line from a crooked one and can easily say that the title of this page is "inside" the frame of the page although the black ink bears no resemblance to the white paper. This is done by using the knowledge that the sensors receiving the image of the black ink are located "inside" the region of sensors that received the white of the paper.

This paper shows how the first stage of such a process was achieved, namely, the creation of a mapping between physical distances of sensors on the retina and "neural distances" in the brain, the latter expressed in terms of synaptic weights. We proceed by segmenting the retina into small segments. Each such segment is processed by a small feed forward neural net to deduce some local property while ignoring the segments position in the visual field. The local property will be the value of a specified function of the intensity values of the pixels belonging to the segment. Every small net will act as an "independent sensor". These sensor outputs are then given as inputs to a second net that uses Hebbian learning to deduce the original positions of the individual sensors, as shown in Figure 1. This is done under an assumption that the distance between them (the sensors) is inversely related to the correlation between their outputs.

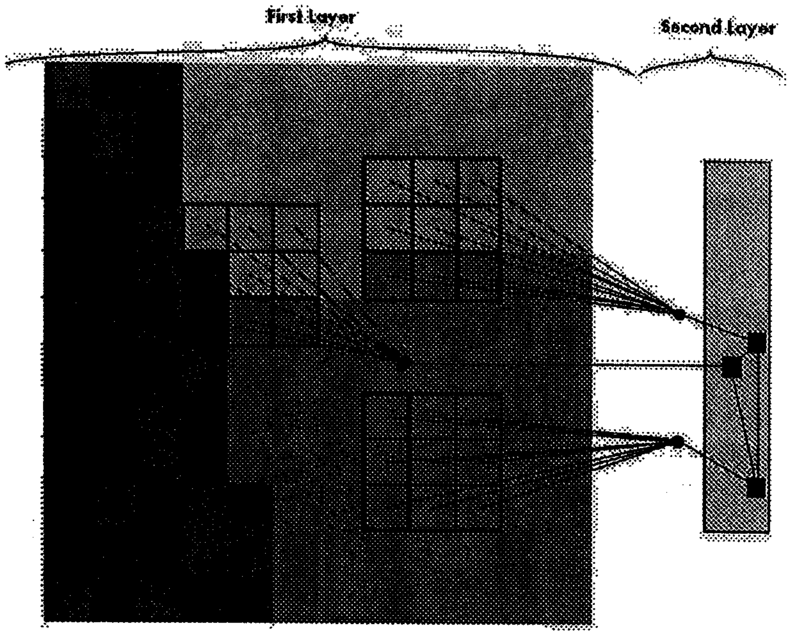


Figure 1. On the left hand side, three retinal segments and their corresponding first layer nets. On the right hand side the gray rectangle represents the Hebbian net, referred to as the Second Layer.

We will show that, during the learning process, every synaptic weight between two neurons in the Hebbian net (denoted with small squares on the right hand side of Figure 1) becomes proportional to the Euclidian distance between the corresponding segments.

In section 2 we discuss the specific implementation details of the first layer. We include a description of the specific local function chosen, the training set, and the training details. The second section deals with the second layer. We describe how image inputs were created and how the Hebbian network was implemented. The third section contains the results of the experiment, given both graphically and numerically. The results show that the distances between sensors, learned by the net, are correct up to an error of 10% of the image size. The fourth section includes result analysis. In this section we also describe some variants of the experiment performed and mention some results but without going into any detail.

Our eventual goal is to accurately position each sensor on the XY plane and thereby reconstructing the retinal spatial structure. The distance map created in this experiment is therefor only the first of two stages. After the distance map is obtained, each sensor actual position can be approximated using a relaxation process similar to a self organizing map due to Kohonen (1982). This paper will not describe the implementation of this second process.

2. FIRST LAYER

The first stage will be to create the small nets that perform the local calculation. The calculation chosen is a weighted average of the brightness of the central pixel and it's neighbors. A pixels neighbors are the eight pixels surrounding it (as shown in figure 2).

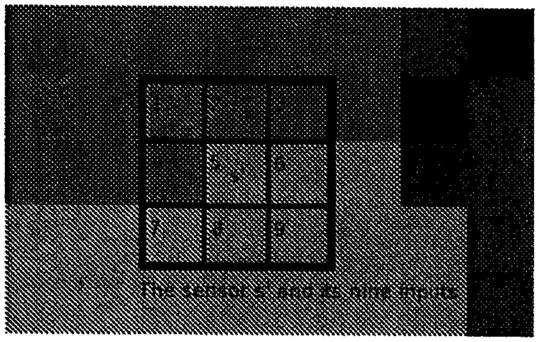


Figure 2. Sensor s' and the nine inputs it receives.

The average function for sensor (pixel) s' in figure 2 is

$$F(s') = (5x_5 + 3(x_2 + x_4 + x_6 + x_8) + 2(x_1 + x_3 + x_7 + x_9))/25$$

Here x_i denotes the brightness of pixel i. This simple linear function can be approximated by a single neuron with 9 inputs. Each training epoch contained a 1000 randomly chosen exemplars. Each such exemplar consisted of a random vector with 9 components and the weighted average of these components. Random inputs were chosen because training the net with inputs taken from actual images resulted in very slow convergence rates. This is due to the fact that visual data extracted from 3x3 pixel matrices is usually not sufficiently diverse. In other words, the vast majority of 3x3 pixel matrices contained roughly the same color and therefor contained less information then a random such matrix.

Technical Details

Net configuration:

- 1 output neuron
- 9 inputs
- Training algorithm: gradient decent
- Transfer function: linear
- Learning rate: 0.01
- Max number of epochs: 1000
- Error goal: 0.1

Training:

- Size of training set: 1000
- Epochs needed for convergence: about 150
- Training convergence graph shown in figure 3.

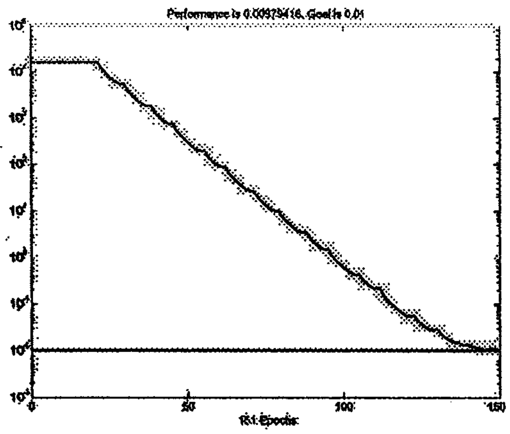


Figure 3. The performance graph while learning the average function.

3. SECOND LAYER

Having trained the basic calculating blocks, we now turn to describe the implementation of the second layer. In this section we describe how input images were created and how the Hebbian network was implemented. For reasons of technical convenience I chose to work with images consisting of 12x12 pixels [1]. Since the average function is not defined for pixels on the edge of the image we do not create sensors for them. This means that from a 12x12 pixel image we actually get only a 100 sensor segments, instead of the expected 144. In order to supply the images needed to train the second layer, a random generator for images was created. The random images created are not literally random. The generator randomly selects two chords through the image. The chords divide the image into four parts. To each of the above parts the generator assigns a random color [2]. To make the model more realistic the intensities were summed with noise. Each pixel in the image was augmented by a random brightness value between 0 and 20. The same experiment was run without noise, the results of which are given in section 4. Sample input images are shown in figure 4.

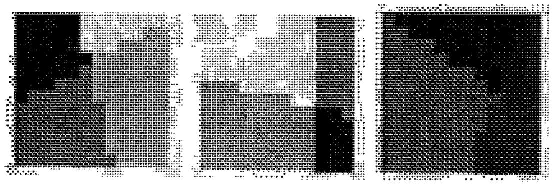


Figure 4. Three typical images created by the random image generator.

An adjacency matrix M of size 100x100 is initialized to zero. M will hold the learned distance between each pair of sensors. The distance between $Sensor(x_1, y_1)$ and $Sensor(x_2, y_2)$ is the Euclidian distance $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. In the adjacency matrix this distance will be given in $M(i, j) = M(10x_i + y_i, 10x_j + y_j)$.

At each iteration, after an input image is supplied the 100 outputs of the basic nets are calculated. These outputs are then fed as inputs to the second layer. For each pair of such outputs we update the learned adjacency matrix as follows,

$$M_{10x_1+y_1,10x_2+y_2}(n+1) = M_{10x_1+y_1,10x_2+y_2}(n)+\eta|output(Sensor(x_1, y_1))-output(Sensor(x_2, y_2))|$$

This weight updating formula serves as the implementation of the Hebbian learning. Each entry in the adjacency matrix, corresponding to two sensors, holds the weight of the con-

[1] The number of basic nets needed is roughly the number of pixels in the analyzed image. Larger pictures will be computationally intense since the size of the Hebbian network connecting all the sensors is quadratic in their number.

[2] A color in this case was just a number between 0-255

nection between them. Notice that a large difference between sensor outputs causes a large increase to their corresponding weight. This gives us the desired effect. Sensors that are far apart are likely to have different outputs, therefore the corresponding weight between them in the Hebbian net would often increase. Whereas sensors which are very close to one another will usually exhibit similar outputs and the corresponding weight between them will remain relatively low.

The main claim is that, if a sufficient number of images are presented to the net, the resulting learned adjacency matrix is proportional to the actual distance matrix D (see figure 8) [3] The proportionality factor between the two matrices can be found in two ways. One way, is to calculate the factor C that minimizes The total error, $min||CM - D||$. This method was not chosen. Alternatively, notice that the largest entry in the actual distance matrix D is of size $10\sqrt{2}$ corresponding to pairs of sensors in opposite corners. If the two matrices are proportional to each other, then the scaling factor must be equal to the ratio between the largest entry of M (the learned adjacency matrix) and $10\sqrt{2}$. Although the scaling factor achieved by the first method clearly reduces the average error, we chose to use the second method. The reason was that the second method does not require the use of D (the actual distance matrix) [4].

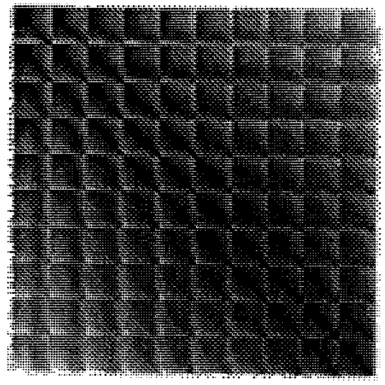
[3]The matrix D holds the actual distance between all pair of sensors. The distance between $sensor(x_i, y_i)$ and $sensor(x_j, y_j)$ is given in the distance matrix D in position

$$D(i,j) = D(10x_i + y_i, 10x_j + y_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

[4]For every NxN image the scaling factor is $N\sqrt{2}$. There is no need to find the largest entry in the actual distance matrix.

4. RESULTS

For visualization purposes the learned adjacency weight matrix was rescaled to the range 0-255 and presented as a gray scale image. Given in the figures below are the learned adjacency weight matrices after 10, 100, and 1000 images ware presented to the network.



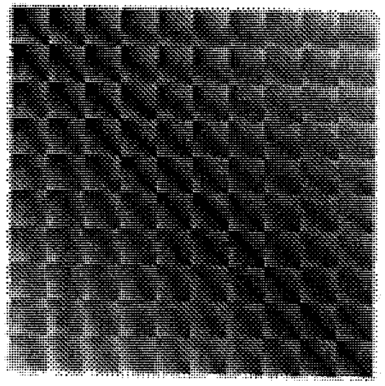
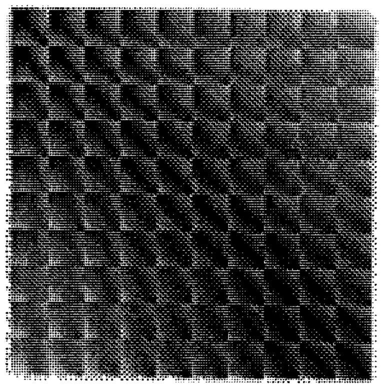


Figure 5,6. Learned adjacency matrix for 10 and 100 images.

For visual comparison, I present in figures 7 and 8, M the learned adjacency matrix and D the matrix created by calculating the Euclidian distance between each pair of locations on the original image.



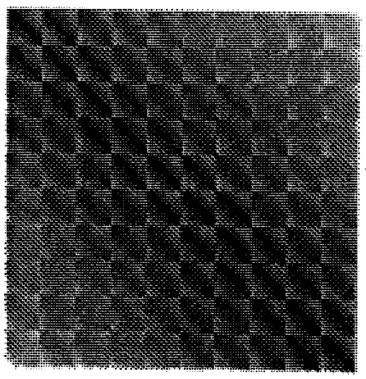


Figure 7,8. Learned adjacency matrix for 1000 images and the actual distance matrix D.

Numerical Results

In the table below all errors given are in pixel length units. Both D and M are of size NxN, N=100

number of images	$Max\|M_{i,j} - D_{i,j}\|$	$\frac{1}{N^2}\sum_{i,j}\|M_{i,j} - D_{i,j}\|$	$\frac{1}{N^2}\sqrt{\sum_{i,j}(M_{i,j} - D_{i,j})^2}$
10^0	11.890	4.0696	4.7607
10^1	7.3089	2.0748	2.6034
10^2	4.9418	1.1954	1.4874
10^3	4.7736	1.2661	1.6169
10^4	4.7642	1.2158	1.5554

5. DISCUSSION

I will argue later that the errors displayed in the table are sufficiently small. Nevertheless after processing a large number of images the largest error in distance is still about 5 pixels. The exact reason for which is not known. However the errors are distance errors between each pair of sensors. This means that statistically each sensor is displaced by only half of that. Moreover the sensors close to the borders of the picture suffer from border effects. Thus the near border sensors are responsible for most of the average error. In a second experiment I measured the errors corresponding to centrally located sensors. The errors found were typically less then one pixel.

Now consider the average error, the second colum in the table. Notice that it is less the 1.3 pixels in length. That means that the weights in the Hebbian net give an almost correct distance estimate to each pair of sensors [5]. These results are satisfactory. Using these distances we can initiate a self organizing relaxation process that achieves an accurate map of the actual sensor locations on the retina. This process, although not complicated, is out of the scope of this paper and will be dealt with in a subsequent article.

One surprising result was that using a very basic local function like a weighted average over a nine dimension input vector (or a 3x3 matrix) was sufficient to learn the correct sensor

[5] In our case we have a 100 sensors, which means that we obtained for each sensor 99 almost correct distances to other sensors

distances, i.e to learn the matrix D. Moreover I managed to get very similar results while running the learning process using a wide variety of local functions. These results are not shown in this paper. It seems that the choice of the local function is almost irrelevant. The corelation between neighboring regions was always sufficient to build a full and correct distance map.

Not included in this paper are experiments which I made with different characteristics for the generated images. One set of those images were divided to more then 4 colors and with more noise. A feature of these experiments was much slower convergence. Moreover much faster convergence was achieved by using a different set of images. The later were divided by only one chord and assigned only two colors (instead of four), they were also not added with noise.

6. SUMMARY

Overall, the results achieved, are accurate enough to demonstrate that global knowledge regarding the relative distances between sensors on the retina can be achieved while processing their outputs, and the corelation between them, i.e no description of their actual geometric physical setup is needed. Although this is not shown here, the data learned by the Hebbian net is perfectly sufficient to restore the original location of each of the sensors. Moreover the same task can be performed by the Hebbian net regardless of the local functions calculated by the first layer.

ACKNOWLEDGMENTS

I thank professor Willard Miranker for his help with writing this paper.

A Game Theoretic and Genetic Algorithmic Approach to Modeling the Emergence of Mind in Early Child Development with Exposure to Semi-Developed Parents

James Logan
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract

We use a genetic algorithm to study a child's development of mind in a game theoretic environment modeling the interaction with parental figures. The development of mind is investigated when the child is exposed to both a developed parent and a semi-developed parent, one whose game strategy is optimal or not optimal respectively. This provides a model for how a child's interaction with both a parent and an older sibling can affect their development. As the quality of the sibling's play decreased, the child's development was slowed. Also, when the exposure time to the sibling was increased, relative to exposure to the parent, the child's development was slowed.

Keywords: genetic algorithm, game theory, development of mind

1. INTRODUCTION

Early in life, children develop an awareness that their mind is disjoint from the environment. Understanding how this development matures and comes about is an active topic of psychological research. This maturation of awareness is affected by the interaction with parents during early development[1]. This parental interaction can be generalized to two types of responses to a child, "mentalizing" and "non-mentalizing". A mentalizing response would be one in which the parent responds to the child in such a way that acknowledges the child's state of mind. This could be by relating the child's actions to how they are feeling, thus linking the action to a mental state. A non-mentalizing response would be one in which the parent is responding directly to the child's actions without linkage to how the child is thinking or feeling[1]. However, a child is rarely exposed to only one parent during this development.

We will model the interaction of a child with a fully developed parent and a semi-developed parent, which we will refer to as a "sibling" herein. The real life basis for this interaction would be a child who interacts with his parent for a period of time followed by interaction with an older sibling for a period of time. This interaction should affect the child's development as the strategy he encounters will alternate. Hypothetically, this might slow the child's development as it will make the child more likely to misinterpret the strategy as a whole.

To model the development and interaction with a parent, we use a game theoretic approach adumbrated by Miranker and Mayes. The model was originally implemented by Vladimir Barash and is extended to include multiple parent interactions and investigation of that arrangement. This model uses a variant of the game Prisoner's Dilemma in a recurrent form to model the interaction between parent and child. A genetic algorithm is utilized to model the development of the child's emergence of mind through fitness from the game derived from increasingly good memory selection.

In section 2 we discuss the model set up and how it aligns with the interaction with a parent and sibling. Section 3 will discuss the experiments performed and their results. Section 4 will provide conclusions drawn from these results. Overall, the results of the experiments are that with increasing exposure to the sibling, the child's development is slowed and when the sibling's development is reduced the child's development is also slowed.

2. THE MODEL

2.1 Game Theoretic Environment

To model the interaction between the parent and child, a modified version of iterated Prisoner's Dilemma will be used. The game consists of iterative steps that result in a changes to the fitness level of the child. The parent's strategy is not based on a payoff system; the parent's strategy is fixed during the experiment. The moves are Intuition and Mind for the child and Attention and Ignore for the parent. The child wants to get Attention from the parent, and also wants to use his Intuition instead of building his Mind. This models that there is a cost involved with using his Mind, so he prefers to utilize his Intuition. The parent wants the child to develop his Mind, but prefers Ignoring the child to paying it Attention. This models that the parent has a cost associated with paying Attention to the child, while Ignoring the child and having him develop would be ideal to the parent. The payoff matrix is shown in the following table (payoffs are listed as parent, child):

	Attend	Ignore
Mind	R,R	S,T
Intuition	T,S	P,P

Figure 1: Payoff Table

This model becomes a Prisoner's Dilemma when T>R>P>S. This models the desired interactions as described above.

2.2 Genetic Algorithm Mind Model

The strategies of the parents in this model are held constant in order to isolate the child's development. It is assumed that the parent is at a constant stage of development, and thus the parent's strategy should not be fluid during the interaction. The strategy of a fully developed parent will be to play a "tit for tat" strategy. This strategy plays Attend if the child last played his Mind or plays Ignore if the child last used his Intuition.

The child's play consists of two options, a random play or a memory based play. At the beginning of play, the child's choice of play is 50/50, because the child has no development of memory at this point and so plays randomly; the child simply chooses random plays. The child's fitness is increased by an amount equal to the payoff. The child then determines whether or not to memorize this play. This determination is done by checking against a random variable and a memory is created, as follows.

The child's memory is structured as a list of sequences of plays. Each sequence contains a list of tags, which are a set of 2 moves (1 for the parent and 1 for the child). Each sequence is also assigned a weight. This weight represents how beneficial the sequence is for the child to follow, and is given a relatively small value when first specified, based on the overall increase of the child's fitness based on the series of plays. (which initially will be low due to the short length of the sequence)

If the child chooses to use his memory, then the child searches his memory for tags that agree with the last sequence of plays beginning with the last play of the parent. If one or more sequences is found, the child orders the tags based on their weight and plays the first move in the sequence with the greatest weight. If the parent's subsequent move is the same as the next tag in the sequence, the sequence is continued, with the child playing the corresponding move in the next tag. The child continues to follow out the sequence until either he reaches the end of the sequence or the parent deviates from their move listed in the sequence. Once the parent deviates from the sequence's predicted moves or the child's tag sequence runs out, the child makes a random play. If the payoff from this play is is greater than a specified constant then the payoff is added to the child's fitness and the weight of the tag is increased, otherwise it is subtracted from the fitness and the tag's weight is decreased. If the child's sequence is at an end, the random move is added to the end of the sequence. If the parent has deviated from the moves in the sequence, the beginning of the sequence (the part that has been traversed) is used to create a new sequence with the deviating parental move and the child's move added on. Once the sequence has been completed, the child then begins again and either plays randomly or searches his memory for another sequence. For an example of memory generation, see Appendix A.

2.3 Exposure to Multiple Parents

The extension to this model is the exposure to the child of parents with different strategies. The first parent's strategy will be a "tit for tat", as described above. A second, less developed parent, or sibling, will be modeled with the same strategy, but with a random deviation from that strategy. The sibling's deviation will occur if a random variable, constrained between 0 and 1, overcomes constant threshold, A. The value of A can range from 1 (being a fully developed parent) to .5 (being random in choice selection).

We will model the child playing with the parent for a period of time and then playing with the sibling, with repetitive exposure. To do this, the iterations of the Prisoner's Dilemma game will alternate between the parent and the sibling, with exposure to each spanning multiple turns. For example, the developed parent would be exposed to the child for 10 iterations, or turns, followed by 10 turns of exposure to the sibling. The repetition represents multiple exposure to both participants. A single 'run', or a single instance of the game, will consist of 100 iterations. The model for the child's memory will be reset for each run by removing all sequences and setting the child's fitness back to zero. The goal will be to determine what the effect of the exposure to multiple parents will have on the development of the child's development.

3. EXPERIMENT

3.1 Overview

First, we ran the model with no interaction with a sibling in order to produce a baseline to compare with. Second, we ran the model several times with both the fully developed parent and the sibling, altering the development of that sibling. Third, we ran the model with a fixed development of the sibling, or A value, while changing the frequency of parent-sibling interaction.

The information we are trying to extract from the model is at what point the child develops a strategy that allows him to play the game effectively. That is, the child has a development of mind at this point that allows him to play successfully. This effectively results in the child's fitness exploding. We want to see the effect of exposure to multiple caregivers on the time for the child to develop this strategy. To measure this, we ran the game (for 100 child plays) 40 times and noted the number of child plays, or turns, until the child's fitness rose above a specified threshold, sufficiently high to indicate when the fitness has exploded. For each trial of 40 runs, there are several that do not ever exceed the threshold. This means that the child's fitness has never exploded, herein referred to as the run "maturing". These runs are not included in the figures but their number is noted in the text. For a more detailed description of how this threshold is used and an example run, see Appendix B.

3.2 Exclusive Parent Exposure

These runs included no interactions with a sibling and provide a basis of comparison (see Figure 2). During this trial, in which 5 runs did not mature, we can see the the majority (22) of the runs matured before 55 turns, with the remaining maturation trailing off toward 100 plays. The mean and standard deviation of time to mature are 53.57 turns and 17.51, respectively.

3.3 Exclusive Sibling Exposure

For this trial, the child is exposed only to a sibling with an A value of .75 (see Figure 3). We see that the maturation of the runs is delayed, with most occurring between 40 and 80. This trial also exhibit a distribution that more closely resembles a normal

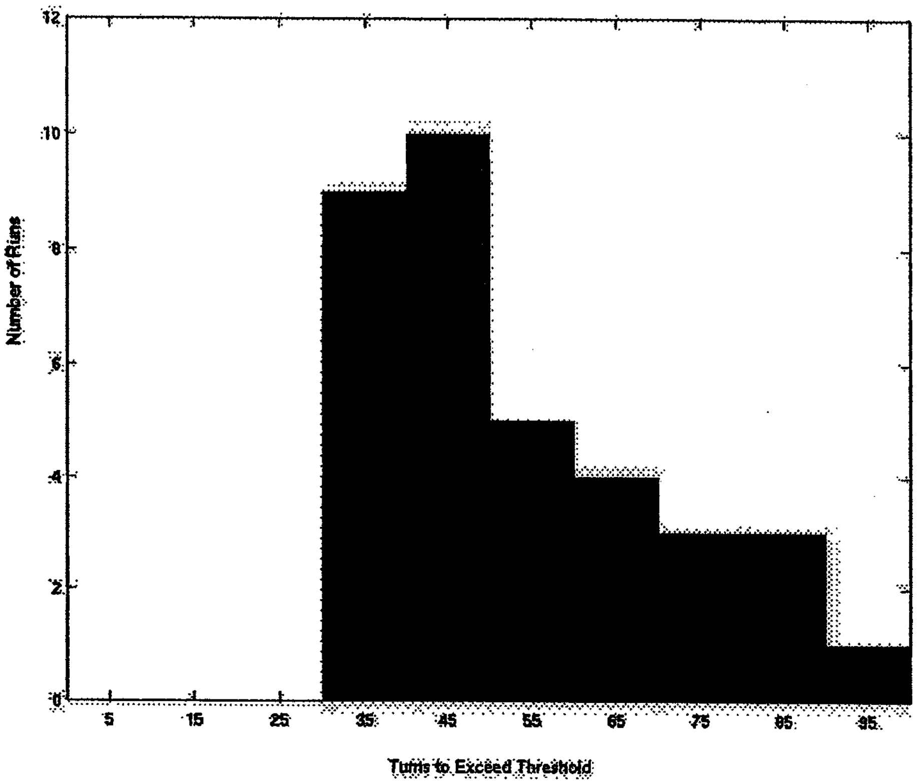


Figure 2: Exclusive Parent Trial

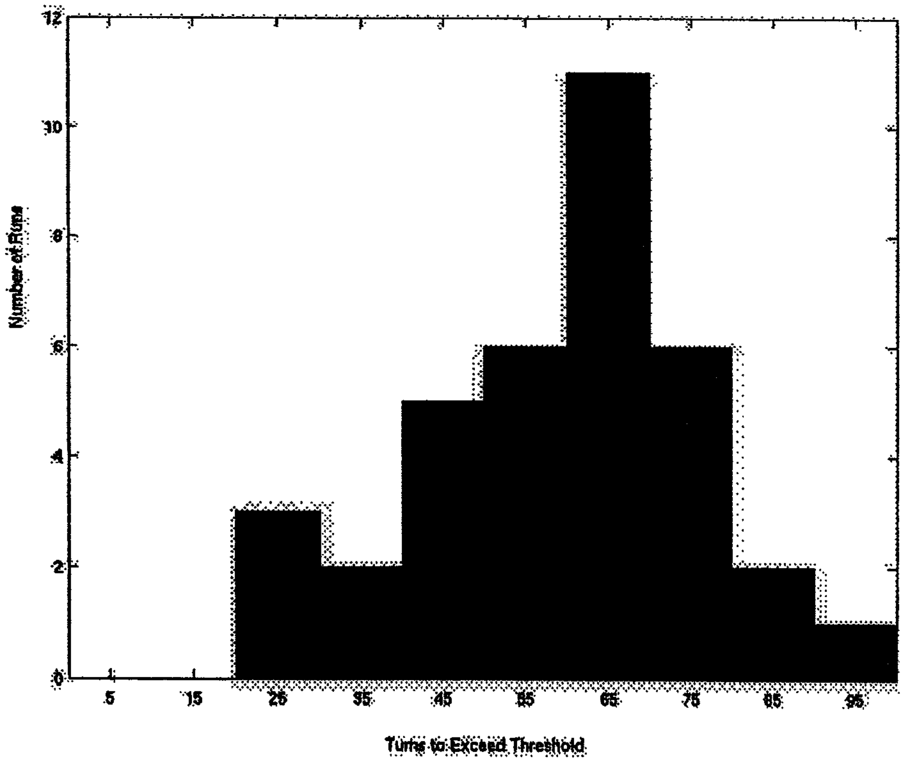


Figure 3: Exclusive Sibling Trial

distribution. During this trial, 4 runs did not mature. The mean time to mature was 59.17 turns and the standard deviation was 17.13.

3.4 Comparison of Different Levels of Sibling Development

For this series of trials, the child is exposed to the fully developed parent for 10 turns followed by the sibling for 10 turns, repeated 5 times. The sibling's 'A' value, which determines the randomness of their plays, is tested at several values and shown for the values of .65, .75, and .85.

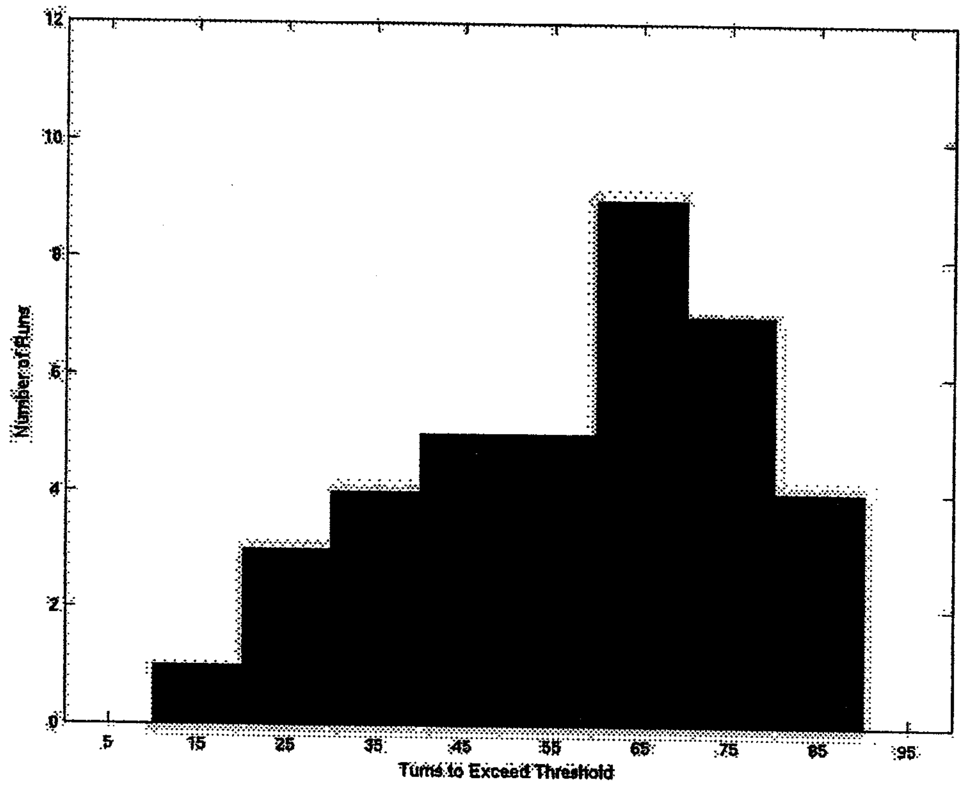


Figure 4: Sibling with A=.65

For the run with A=.65 (see Figure 4), it is obvious that exposure to the sibling is affecting the child's development. The explosive growth of these runs is occurring much later than in the exclusive parent trial, as shown in Figure 4. During this trial, there are 6 runs that fail to mature. The mean time to mature is 57.36 turns with a standard deviation of 19.09.

When the A value is increased to .75, as shown in Figure 5, the bulk of the exponential growth occurs between 40 and 70 turns, with 2 runs failing to mature. As expected, the increase in A value has decreased the overall time to converge for most of

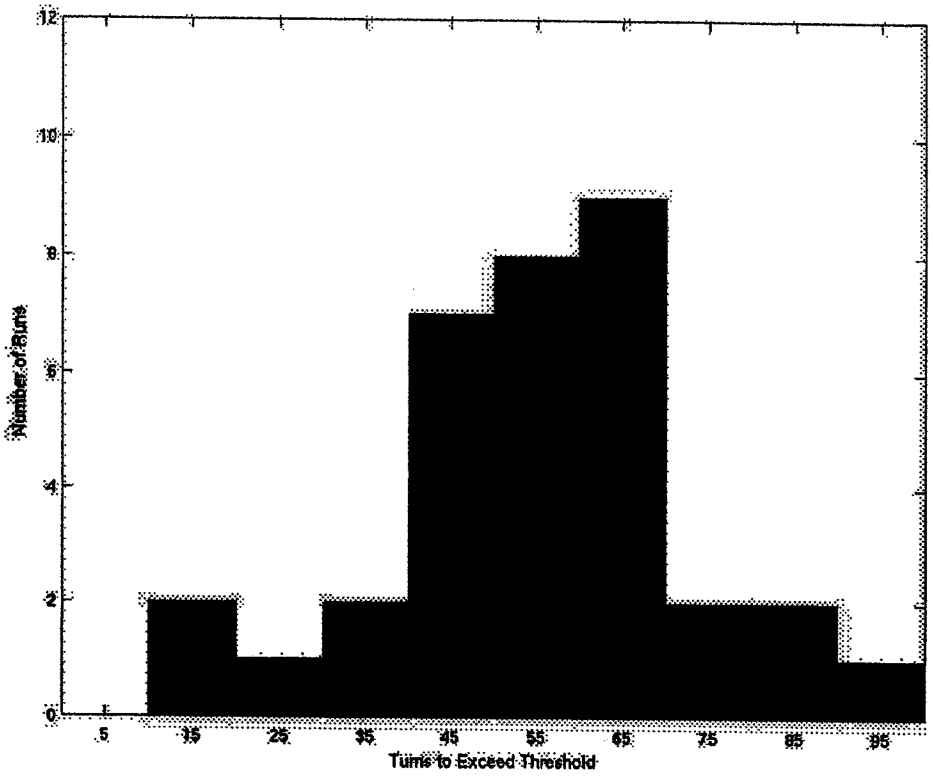


Figure 5: Sibling with A=.75

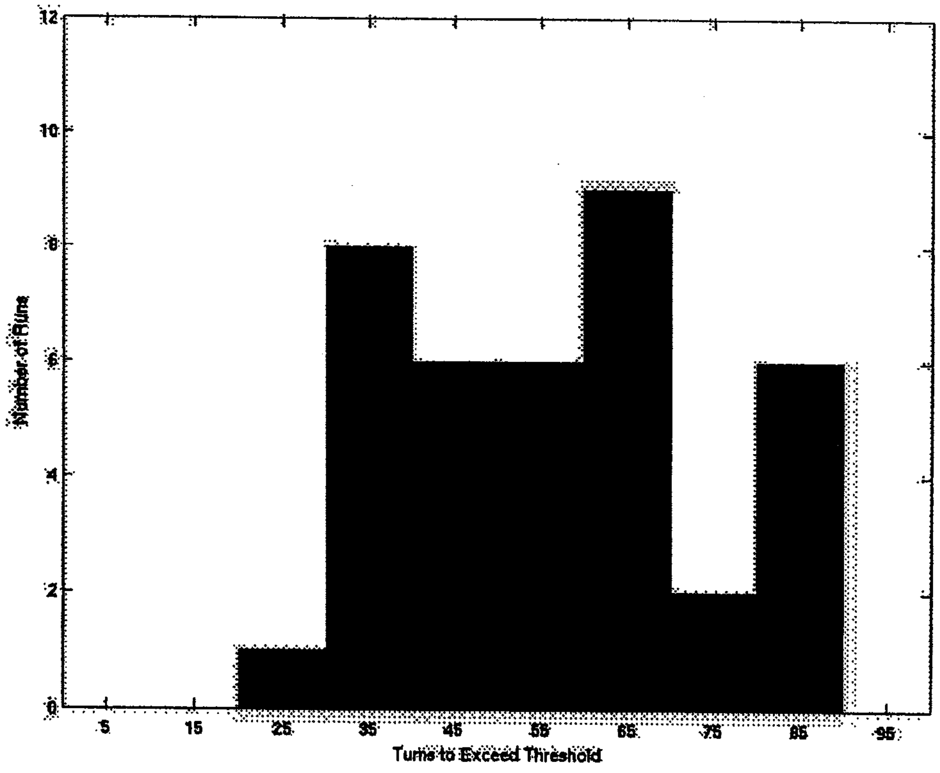


Figure 6: Sibling with A=.85

the runs. The mean maturation time was 55.29 turns with a standard deviation of 18.01. On the third trial, with A=.85 (see Figure 6), the time to mature decreases overall, although less pronounced than the previous change, to a mean of 54.92 turns with a standard deviation of 17.72 turns. Two runs failed to mature.

A Value	Mean	Stand Dev.	Non-Maturing
0.65	57.36	19.39	6
0.70	56.54	18.38	5
0.75	55.48	18.31	2
0.80	55.16	16.66	4
0.85	54.92	17.72	2
0.90	55.08	15.91	2
0.95	52.21	15.39	1

Table 7: Statistics of Trials vs. Changing A Value

Overall these runs show that as the sibling's development is decreased from that with no sibling exposure, the effect on the child is a slow down of maturation of fitness. The mean time to mature and the standard deviation both decreased as the A value of the sibling decreased. The number of non-maturing runs showed a trend of decreasing overall, with a single outlier. See Table 7 and Figure 8 for results.

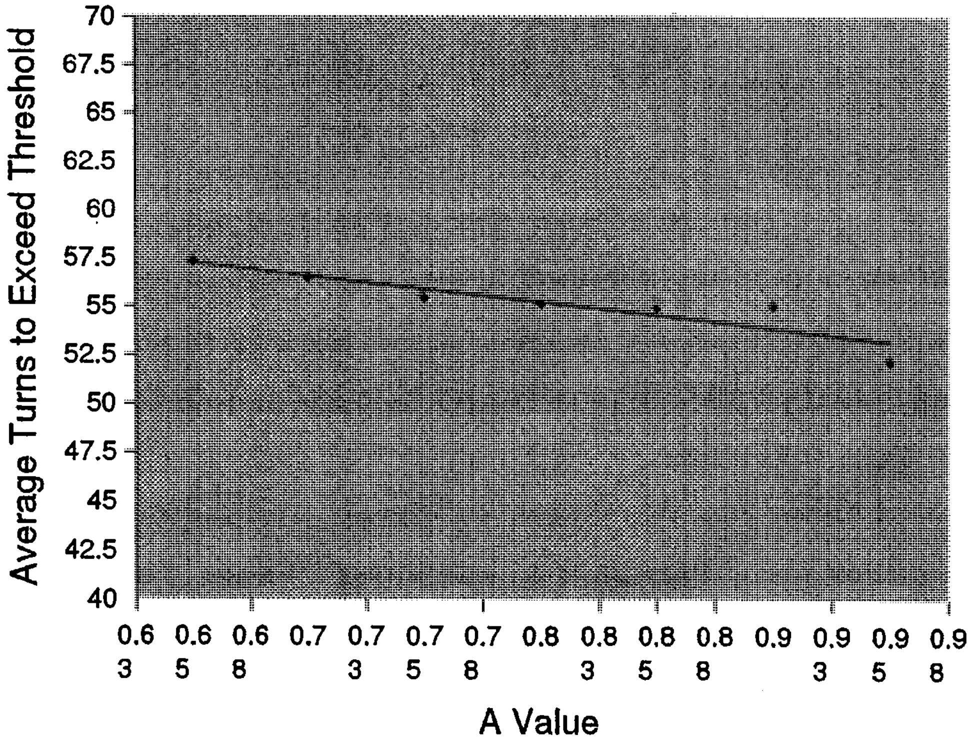


Figure 8: Changing A Value vs. Turns to Exceed Threshold

3.5 Comparison of Different Levels of Sibling Exposure

Next we examine the effects of how siblings affect how the child's mind develops, we adjusted the amount of time that a child interacts with the a parent and a sibling. The base comparison run, that with a sibling with an A value of .75, is shown in Figure 5 above. The first trial exposes the child to the

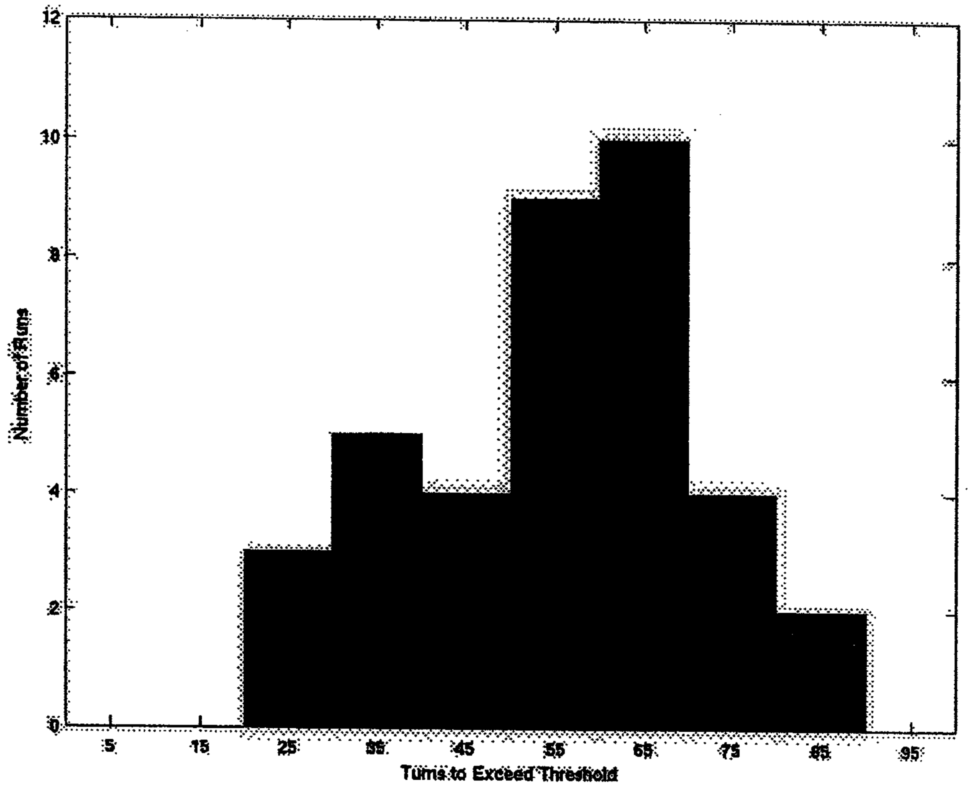


Figure 9: Exposure Skewed to Developed Parent

developed parent for 15 turns, followed by 5 turns with the sibling, repeated 5 times. The next trial flips the exposure, with the child seeing the sibling for 15 turns and the fully developed parent for 5 turns. We would expect that longer exposure to the sibling would delay the child's development.

In the first trial, shown in Figure 9, the maturation time was shorter than in the non-skewed case as expected, with 3 runs not maturing. The mean time to mature was 55.16 turns with a standard deviation of 16.57. This aligns with our prediction as the child is being exposed to the parent for the bulk of the time of the experiment, and thus the retardation introduced by the sibling does not have as much effect on the results.

When the child is exposed to the sibling for the longer period, the time to mature is increased from the base case, but still only has 3 runs not maturing. (see Figure 10) The mean time to mature for this trial was 57.86 turns with a standard deviation of 19.65.

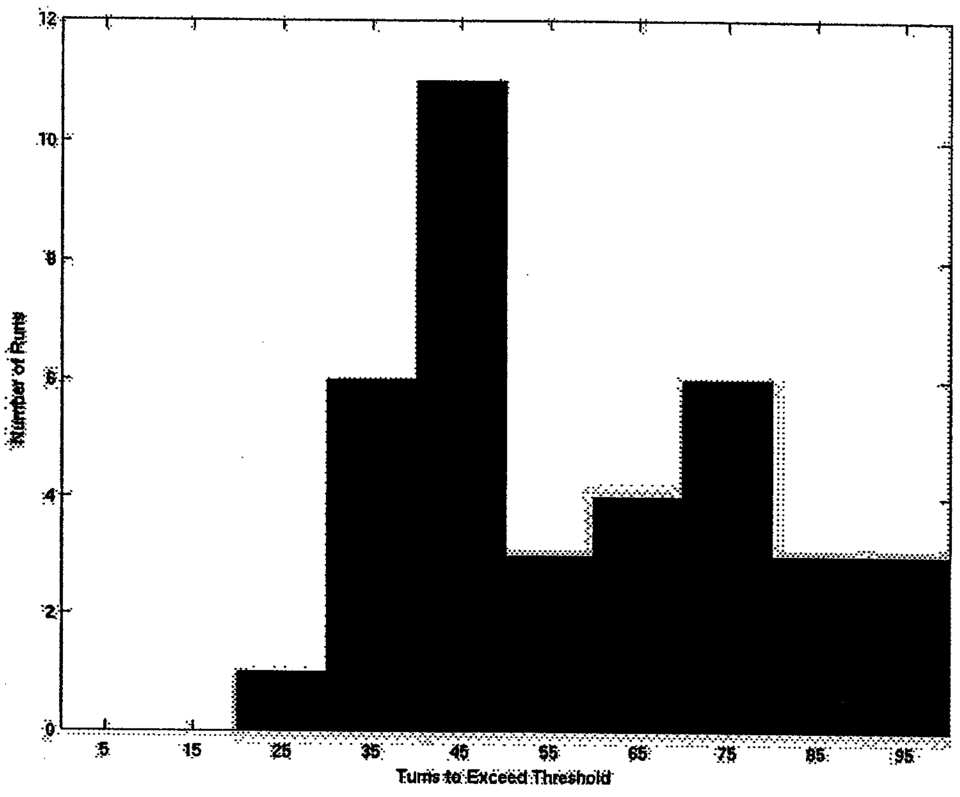


Figure 10: Exposure Skewed to Sibling

This trial shows that the randomness of the sibling affects the ability of the child to achieve it's convergence on a strategy, which is what was expected. The longer the sibling was exposed to the child, the longer it took the child's mind to mature, as seen in Figure 10. Also, the increasing exposure to the sibling spread the distribution of the maturation times. However, it seems that the number of runs that didn't mature were not affected greatly by the change in time exposed to the sibling. These observations are summarized in Table 11.

Sibling Exposure	Mean	Stand Dev.	Non-Maturing
0.25	55.16	16.57	3
0.5	55.48	18.31	2
0.75	57.86	19.65	3

Table 11: Statistics of Trials vs. Sibling Exposure

4. CONCLUSION

The results suggest that as the child is exposed to a sub-optimal strategy it becomes harder for him to converge on a sequence of plays that dominates the playing ability of the parent's optimal strategy. As the A value of the sibling is decreased, and thus the sibling's development decreased, the child's development is slowed. The child's memory development is retarded due to the inconsistent strategies that the sibling exposes him to. This A value can be linked to the a physical attribute of the parent's condition. For example, a decreasing A value could represent an increase in depression on the part of the parent or a decrease in the parent's ability to focus on raising the child. The exposure to a sub-optimal strategy does slow the child's development, but the effect that is more pronounced is the increase in unpredictability in the maturation time of the child's development. This result could be interpreted as showing that although a child is overall hindered by exposure sub-optimal parenting, the child's development would also be less likely to be in synchronization to other children. In other words, the child may mature faster or slower than his peers as the unpredictability of his development has increased.

5. REFERENCES AND ACKNOWLEDGEMENTS

1. Mayes and Miranker, A Game Theoretic and Genetic Algorithmic Approach to Modeling the Emergence of Mind in Early Child Development

I would like to acknowledge Willard Miranker for his supervision and guidance on this project and also Linda Mayes for her perspectives on the psychological aspects of the experiments.

6. APPENDICES

Appendix A

To illustrate how the memory is formed, we show the memory generation on a half run without sibling interaction,. The symbols used are 'I' – Intuition, 'M' – Mind, 'A' – Attention, and 'G' – Ignore. Following the sequence it the weight assigned to the tag.

Memory after 5 moves
[M,A], 0.050
[M,G,M,A], 0.060

As you can see, the initial sequences formed are relatively short and have low values for their weights due to the low payoff that playing these sequences generates. Note that the longer sequence, although

Memory after 10 moves
[M,A], 0.050
[M,G,M,G,I,G], 0.106

Memory after 15 moves
[M,A,M,G], 0.086
[M,G,M,G,I,G,M,A], 0.124

Memory after 20 moves
[M,A,M,G,I,G,I,A,I,A], 0.149
[M,G,M,G,I,G,M,A], 0.124

Memory after 25 moves
[M,A,M,G,I,G,I,A,M,G], 0.383
[M,G,M,G,I,G,M,A,I,G,I,G], 0.279
[M,A,M,A], 0.050

Memory after 30 moves
[M,A,M,G,I,G,I,A,I,A,I,G], 0.450
[M,G,M,G,I,G,M,A,I,G,I,G], 0.279
[M,A,M,A], 0.050
[M,A,M,G,I,G,I,G], 0.085
[M,A,M,G,I,G,M,G], 0.166

Memory after 35 moves
[M,A,M,G,I,G,I,A,I,A,I,G], 0.450
[M,G,M,G,I,G,M,A,I,G,M,A], 0.617
[M,A,M,A], 0.050
[M,A,M,G,I,G,I,G], 0.085

```
[M,A,M,G,I,G,M,G], 0.166
[M,G,M,G,I,G,M,G,I,G], 2.018
[M,A,I,G], 8.050

Memory after 40 moves
[M,A,M,G,I,G,I,A,I,A,I,G], 0.450
[M,G,M,G,I,G,M,A,I,G,M,A], 0.617
[M,A,M,A], 0.050
[M,A,M,G,I,G,I,G], 0.085
[M,A,M,G,I,G,M,G], 0.166
[M,G,M,G,I,G,M,G,I,G], 2.018
[M,A,I,G,M,A], 18.780
[M,G,M,G,M,G,I,G], 321.905

Memory after 45 moves
[M,A,M,G,I,G,I,A,I,A,I,G], 0.450
[M,G,M,G,I,G,M,A,I,G,M,A], 0.617
[M,A,M,A], 0.050
[M,A,M,G,I,G,I,G], 0.085
[M,A,M,G,I,G,M,G], 0.166
[M,G,M,G,I,G,M,G,I,G], 2.018
[M,A,I,G,M,A,I,G,I,A,I,A], 2422.438
[M,G,M,G,M,G,I,A], 5815.975
[M,A,I,G,M,A,M,A], 2747.032

Memory after 50 moves
[M,A,M,G,I,G,I,A,I,A,I,G], 0.450
[M,G,M,G,I,G,M,A,I,G,M,A], 0.617
[M,A,M,A,I,G], 197786.497
[M,A,M,G,I,G,I,G], 0.085
[M,A,M,G,I,G,M,G], 0.166
[M,G,M,G,I,G,M,G,I,G], 2.018
[M,A,I,G,M,A,I,G,I,A,I,A], 2422.438
[M,G,M,G,M,G,I,A], 5815.975
[M,A,I,G,M,A,M,A,I,G], 13735.182
[M,A,M,A,I,G], 197786.497
[M,G,M,A,I,G], 131857.658
```

Appendix B

This experiment's key observation was the time of maturation of the child's mind. An example of a single run is shown in Figure A1. As you can see, the fitness of the child stays relatively low until the point at which a "critical mass" is achieved; once that point is reached, the child's payoff explodes. This critical quantity is representative of the child's play maturing. For our trials, the threshold to determine

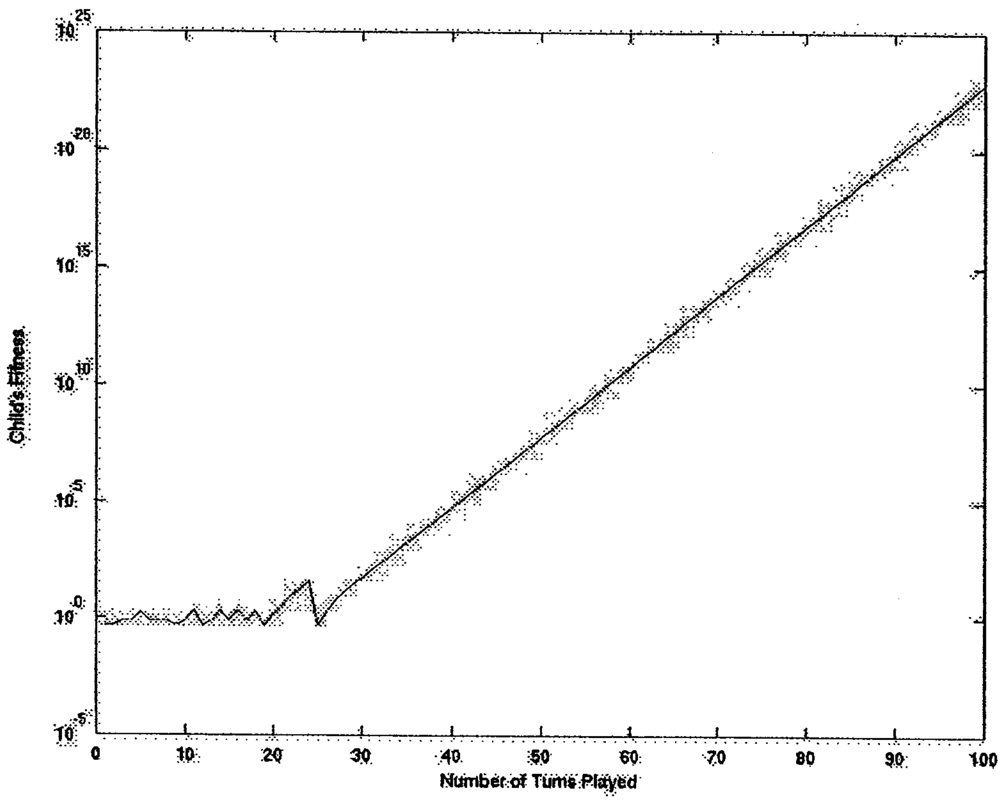


Figure B1: Example run

reaching of maturity was Fitness=10^5. This threshold was chosen because, as seen in Figure B1, the fitness can bounce around quite a bit before reaching the point of interest. In our trials, we ran 40 runs and used the point at which the fitness exceeded the threshold to determine the point of maturation. Any skew caused by the somewhat high threshold value remains constant throughout the experiments, and thus can be disregarded.

Performance Analysis of Different Neurogenetic Behaviors

Yihui Qian
Department of Computer Science
Yale University
New Haven, CT 06520, U.S.A.
Email: yihui.qian@yale.edu

Abstract — Neural networks have been proved to be good modeling tools for simulation of neurogenesis phenomena. In this paper, after developing a feed forward neural network, we analyze the performance of different neurogenetic behaviors. We investigate the effects of the proportion of replaced neurons, the distribution of the replaced neurons and the neuron selection criteria. Through simulation and analysis, we show that the learning speed of neural networks can be improved by increasing the number of replaced neurons, or by changing the distribution of replaced neurons appropriately, or by replacing those neurons with large weights.

Keywords — neurogenesis, apoptosis, neural network, hippocampus

1. INTRODUCTION

Neurogenesis in mammals, the fact that new brain neurons are born and developed even in adulthood, is a recently discovered phenomenon and has been widely accepted. It has also been demonstrated that the adult neurogenesis in mammalian species occurs in the dentate gryrus (DG) of the hippocampus and in the olfactory system (Eriksson et al, 1998). Since the hippocampus is prominently identified with short-term memory functions, long-term memory retrieval and emotion related activities etc., it is suspected that neurogenesis plays a key role in many memory related activities.

Recent studies have shown that neurogenesis favorably informs memory development (Chambers et al, 2004). However, there are still some questions remaining to be investigated regarding functional significance of different kinds of neurogenetic behaviors. For example, what is the relationship between the number of replaced neurons and the effect it brings to brain activities? Does the distribution of replacement also affect the performance? Furthermore, is the replacement deterministic or random?

In this paper, aiming to answer these questions, we build a feed forward neural network to simulate the neurogenesis process and study the performance of various neurogenetic behaviors. Specifically, the performance is referred to as the mean convergence time for learning process and the quality of recall process in a noisy environment. With these two performance metrics, three experiments are conducted: 1) neurogenesis simulations with different proportions of replaced neurons; 2) neurogenesis simulations with different distributions of replaced neurons; and 3) neurogenesis simulations with different neuron selection criteria.

The remainder of this paper is organized as follows. The experiment model is introduced in Section 2. The experiment results are presented in Section 3. We provide analysis and discussion of the experiment results in Section 4. Finally, we draw our conclusions and outline some future work in Section 5.

2. EXPERIMENT MODEL

2.1 Training and Learning Datasets

In our experiments, two character sets are employed as the training and learning datasets. First, the Roman alphabet is applied to train the network. Then the Greek alphabet, which has 14 letters in common with Roman, is presented to the network. Each character of both alphabets is encoded as a 7×5 pixilated image. The encoding of the characters in our experiment is the same as that in (Miranker, 2004).

2.2 Network Architecture

Human hippocampus is comprised of three layers — an input layer (Entorhinal Cortex), a middle layer (DG) and a final output layer (CA3). The Entorhinal Cortex is projected to the DG layer via modifiable perforant path axodendritic connections, and the DG is projected to CA3 via modifiable mossy fiber axodendritic connections. In order to capture the characteristics of this system and obtain meaningful simulation results, we build a three-layer feed forward network — one Input layer, one Middle layer and one Output layer — to model the process.

The Input layer, Middle layer and Output layer has 35, 16 and 35 neurons, respectively. The nodes in neighboring layers are fully connected, but there are no intra-layer connections between the neurons in the same layer. Each neuron in the Middle and Output layer also has a firing threshold (bias) applied to it.

The Log-Sigmoid Transform Function

$$Y_k = \frac{1 - e^{-aV_k}}{1 + e^{aV_k}} \tag{1}$$

is used as the transformation function for the Middle and Output layer, where Y_k is the output of the k-th neuron and V_k is the k-th neuron's weighted sum of inputs. Note that the log-sigmoid function provides a computationally equivalent simulation to the somato-dendritic activation in hippocampus (Chambers et al, 2004).

2.3 Simulation Settings

Weights and biases initialization are performed as follows. The Input-Middle layer weights are uniformly randomly chosen from the interval [-1, 1], and the Middle-Output layer weights are uniformly randomly chosen from [-0.01, 0.01]. These choices have been shown to be appropriate for the character recognition task (Chambers et al, 2004).

We use back propagation training with adaptive learning rate algorithm[1] as the training method for both Roman and Greek learnings, since this algorithm provides an effective learning mechanism with a relatively fast convergence rate.

The simulation has two phases. The first one is the learning of Roman alphabet and the second is the learning of Greek alphabet.

- The learning of Roman

  The encoding of the whole alphabet, which is a binary valued 35×26 matrix, is presented as input. The training target (expected output) is the same as the input matrix. In terms of training parameters, we set the training goal as MSE (mean square error) $\leq 10^{-3}$ and the maximal training epochs as 1000.

- The learning of Greek

  After the Roman alphabet is learned, the Greek alphabet is presented to the network. The number of maximal training epochs is still fixed as 1000, but the learning goal is set to MSE $\leq 5 \times 10^{-3}$.

The apoptosis and neurogenesis process are simulated via re-initialization of appropriate Middle layer neurons. The re-setting of a neuron's weights in random manner models the process in which that neuron dies and is replaced by a nascent neuron.

3. THE EXPERIMENT RESULTS

MATLAB 6.5 and MATLAB Neural Network Toolbox was used to build the network and simulate the process. As introduced in Section 2, the number of neurons in Input, Middle and Output layer is 35, 16 and 35, respectively. Moreover, the 16 neurons in the Middle layer are numbered from 1 to 16 in each of the following experiments.

3.1 Experiment 1

In this experiment, we investigate the difference of performance caused by changing the proportion of neurons undergoing neurogenesis.

The network is trained with Roman and Greek alphabets as explained above. The convergence epochs for learning Greek is used as one performance metric. We conduct the experiment with 0, 2, 4, 8, 10, 14, 16 neurons replaced, respectively. In each setting, the lowest numbered neurons are selected for replacing, e.g. if the total number of replaced neurons is 2, the 1-st and 2-nd neurons will be replaced. Because of the random initialization of

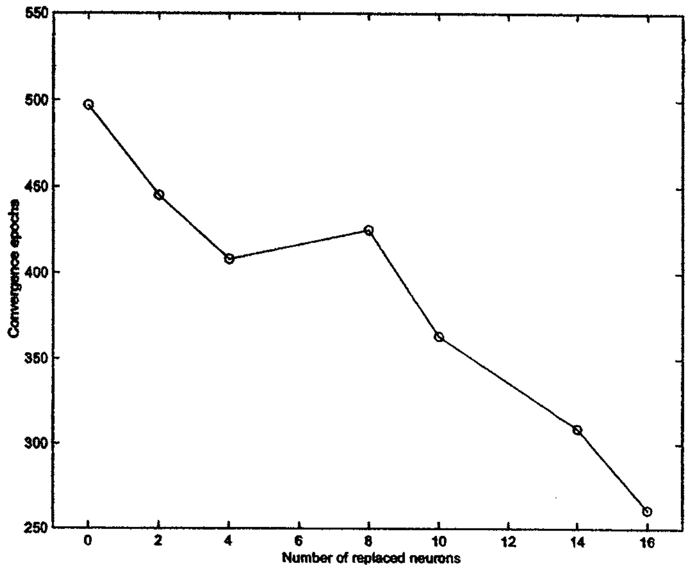


Figure 1:
Convergence epochs v.s. number of replaced neurons

Table 1:
The results of experiment 1

# of replaced neurons	0	2	4	8	10	14	16
Mean # of convergence epoches	497	445	408	425	363	309	261

weights and biases, we repeat the simulation for six times for each setting and average the results.

Table 1 and Figure 1 show the results. We can see that when the number of neurons undergoing neurogenesis increases, the network needs fewer epochs to learn Greek. This shows while the learning of Roman provides a good starting point for learning those common characters in Greek alphabet, it may provide a poorer setting for learning those disparate characters. Thus, the overall performance depends on which kind of effect plays a more significant role. In this experiment, it is demonstrated that the negative effect dominates the performance. Consequently, the learning speed of Greek is improved when more neurons are replaced by nascent ones.

3.2 Experiment 2

In this experiment, we investigate the performance-distribution relationship of neurogenesis process. The number of replaced neurons is fixed as 4, but we change the distribution

[1]Matlab code "traingd"

Table 2:
The results for setting 1 of experiment 2 — the neurons undergoing neurogenesis are 13, 14, 15, 16

	Convergence Epochs	NSSE
1	341	1.7164
2	562	0.1625
3	441	0.6433
4	422	0.9893
5	508	0.2424
6	497	1.0184
7	653	0.2409
8	398	0.5133
9	456	1.5994
10	347	2.1934
Mean	**462.50**	**0.9319**
S.D.	**96.53**	**0.7044**

Table 3:
The results for setting 2 of experiment 2 — the neurons undergoing neurogenesis are 7, 8, 9, 10

	Convergence Epochs	NSSE
1	487	1.4753
2	511	0.1599
3	505	1.4093
4	609	1.1105
5	465	1.3327
6	581	1.0221
7	514	0.2584
8	500	0.2393
9	431	1.6407
10	476	0.2782
Mean	**507.90**	**0.8926**
S.D.	**52.59**	**0.5932**

of these neurons in each test.

Besides using the convergence epochs to evaluate the performance, we also use the sum of error squares after recalling with noisy inputs as another performance metric. The meaning of this value is explained as follows.

After training the network with Roman and Greek alphabets, we present the network

Table 4:
The results for setting 3 of experiment 2 — the neurons undergoing neurogenesis are 1, 5, 9, 13

	Convergence Epochs	NSSE
1	360	1.1862
2	541	1.4665
3	388	1.0275
4	355	1.6797
5	503	1.4905
6	340	1.2099
7	639	0.9650
8	380	2.6778
9	525	1.1491
10	394	0.6302
Mean	**442.50**	**1.3482**
S.D.	**101.64**	**0.5540**

Table 5:
The results for setting 4 of experiment 2 — the neurons undergoing neurogenesis are randomly selected

	Convergence Epochs	NSSE
1	359	0.7601
2	436	1.3900
3	348	1.5011
4	1000	0.0445
5	272	1.2825
6	461	0.8268
7	441	0.5844
8	440	0.6874
9	890	1.1964
10	445	0.8385
Mean	**509.20**	**0.9112**
S.D.	**238.50**	**0.4397**

with a noisy input matrix. This matrix is 35×5 in dimension, consisting of noisy version of five characters - say,A,B,H,K and Ψ- in Greek alphabet. (The noisy input matrix can be found in Appendix). The output is compared with the standard A, B, H, K and Ψ in Greek and the difference is reflected by a scalar NSSE (noise input sum square error), which is calculated as:

$$\text{NSSE} = \text{SSE}\{A\} + \text{SSE}\{B\} + \text{SSE}\{H\} + \text{SSE}\{K\} + \text{SSE}\{\Psi\},$$

where

$$\text{SSE}\{A\} = \sum_{i=1}^{35}(A_i - A_i')^2$$

where A' is the output after recalling with noisy version of A. SSE$\{B\}$, SSE$\{H\}$, SSE$\{K\}$ and SSE$\{\Psi\}$ are computed in the same way.

The neurons in Middle layer are numbered from 1 to 16. The experiments are carried out with the following four settings: 1) the replaced neurons are located at one side — the replaced neurons are 13, 14, 15, 16; 2) the replaced neurons are located in the center — the replaced neurons are 7, 8, 9, 10; 3) the replaced neurons are distributed — the replaced neurons are 1, 5, 9, 13; 4) the replaced neurons are randomly selected.

As before, due to randomness of initialization, we repeat each simulation for ten times and average the results.

The results are given in Table 2-5. Note that when the replaced neurons are distributed in a scattered manner, the network learns faster than when the replaced neurons are clustered no matter in the center or at one side. However, regarding recalling with noise, the clustering distribution performs better than the scattered distribution, i.e. have a smaller NSSE.

3.3 Experiment 3

The third experiment is designed to test whether the neurogenesis process conforms to some deterministic rules to select neurons to be replaced. The two selection criteria we investigate are: 1) replace neurons that are used most often; and 2) replace neurons that are used least often. For the first one, it aims to test whether the most frequently used neurons tend to age fast and thus be replaced soon. For the second one, it aims to verify if the "use it or lose it" theory can apply in the neurogenesis process — if a neuron is not used very often, it will be replaced soon because of its uselessness.

To model these two processes, we calculate the SW (sum of weight squares) for each neuron in the Middle layer:

$$\text{SW}\{\text{neuron } k\} = \sum_{i=1}^{35}|w_{ki}|^2 + \sum_{j=1}^{35}|w_{jk}|^2, k = 1, 2, ..., 16$$

where w_{ki} is the weight between the i-th node in the Input layer and the k-th neuron in the Middle layer, and w_{jk} is the weight between the k-th neuron in the Middle layer and the j-th neuron in the Output layer.

According to the definition, the larger the value of SW is, the more frequently the neurons is used.

In each simulation, the number of neurons to be replaced is fixed as 4. As before, we use both the convergence epochs for learning Greek and the NSSE of recalling with noise as the performance metrics. We repeat each test for nine times and average the results.

Table 6:
The results for case 1 of experiment 3 — replace the 4 neurons with the largest SW

	Convergence Epochs	NSSE
1	363	1.2633
2	630	0.6642
3	496	0.6948
4	363	0.5904
5	319	2.1844
6	328	0.8909
7	395	1.9198
8	432	1.7544
9	262	1.9617
Mean	**398.67**	**1.3238**
S.D.	**109.93**	**0.6386**

Table 7:
The results for case 2 of experiment 3 — replace the four neurons with the smallest SW

	Convergence Epochs	NSSE
1	420	0.6310
2	303	1.1450
3	586	1.0471
4	498	0.5801
5	642	0.2625
6	512	1.1308
7	537	1.1421
8	484	1.3125
9	289	1.6748
Mean	**474.56**	**0.9918**
S.D.	**118.97**	**0.4286**

The results are shown in Table 6-7. We can see that selecting the neurons with largest SW makes the network converge faster. However, in terms of noise recalling, selecting neurons with smallest SW performs better.

4. ANALYSIS AND DISCUSSION

Experiment 1 shows that when the number of neurons undergoing neurogenesis increases, the overall learning performance can be improved. It indicates that previous knowledge may sometimes cause negative effect for the leaning of new knowledge. For example, a person usually needs more time to learn a second foreign language than his/her first foreign language. One of the reasons is because he/she needs extra time to memorize the similarities and differences between these two languages. Under this circumstance, learning without former knowledge outperforms learning with previous "disturbing" knowledge.

Experiment 2 shows that when the replaced neurons are distributed in a scattered manner, the network converges faster but gets a worse recalling ability. On the other hand, if the replaced neurons are clustered, the network learns slower but performs better under noise recalling. This indicates that when the network has more time (relatively more convergence epochs) to learn, its memorizing ability can be developed more sufficiently. As a result, it performs more robustly under noise perturbing. In summary, if the replaced neurons are scattered, short-term memory is improved, while long-term memory development is impaired.

From Experiment 3, we have seen that replacing neurons that are used most often, which with larger weights, can improve the network's learning performance. This is indeed what to be expected if we interpret the network's behavior from an "infomax theory" perspective:

Let X and Y denote the Middle layer's input and output respectively, with X_i and Y_i being the i-th element. Let V_k denote the induced local field of k-th neuron in the Middle layer and let b_k denote that neuron's bias (k=1, 2, ..., 16). Then we have

$$V_k = \sum_{i=1}^{35} w_{ki} X_i + b_k, \tag{2}$$

and

$$Y_k = \frac{1 - e^{-aV_k}}{1 + e^{aV_k}}. \tag{3}$$

Since V_k is in the neighborhood of the origin, for each neuron k, we can approximate its output as:

$$\begin{aligned} Y &= \frac{1 - e^{-aV}}{1 + e^{aV}} \\ &= \frac{1 - (1 - aV) + O(V^2)}{1 + (1 - aV) + O(V^2)} \\ &\approx \frac{aV}{2} \times \frac{1}{1 - \frac{aV}{2}} \\ &= \frac{aV}{2} \times \left(1 + \frac{aV}{2} + (\frac{aV}{2})^2 + \cdots\right) \\ &\approx \frac{aV}{2}. \end{aligned} \tag{4}$$

Assuming b_i is a Gaussian random variable with zero-mean and variance σ_b^2, from (??), each V_k is also a Gaussian random variable whose variance can be obtained as:

$$\begin{aligned}\mathrm{Var}(V_k) &= \mathrm{Var}\Big(\sum_{i=1}^{35} w_{ki}X_i + b_k\Big)\\ &= \mathrm{Var}\Big(\sum_{i=1}^{35} w_{ki}X_i\Big) + \sigma_b^2\\ &= \mathrm{E}\Big[\sum_{i=1}^{35} w_{ki}X_i - \mathrm{E}\Big(\sum_{i=1}^{35} w_{ki}X_i\Big)\Big]^2 + \sigma_b^2,\end{aligned} \tag{5}$$

where the second equation holds because $\sum_{i=1}^{35} w_{ki}X_i$ and b_k are independent.

Therefore, the variance of each Middle layer neuron's output is:

$$\begin{aligned}\sigma_{Y_k}^2 &= \mathrm{Var}\Big(\frac{a}{2}V_k\Big)\\ &= \frac{a^2}{4}\mathrm{Var}(V_k)\\ &= \frac{a^2}{4}\Big\{\mathrm{E}\Big[\sum_{i=1}^{35} w_{ki}X_i - \mathrm{E}\Big(\sum_{i=1}^{35} w_{ki}X_i\Big)\Big]^2 + \sigma_b^2\Big\}.\end{aligned} \tag{6}$$

The mutual information of X, Y can be computed as (Haykin, 1999):

$$I(Y;X) = \frac{1}{2}\log\Big(\frac{\sigma_Y^2}{\sigma_b^2\sum_{i=1}^{35} w_i^2}\Big). \tag{7}$$

Assuming the bias variance σ_b^2 is a constant, the mutual information $I(Y;X)$ is maximized by maximizing the ratio $\frac{\sigma_Y^2}{\sigma_b^2\sum_{i=1}^{35} w_i^2}$. The relationship of σ_Y^2 and w_i is given by (??).

From (??) we have

$$\begin{aligned}\frac{\sigma_Y^2}{\sigma_b^2\sum_{i=1}^{35} w_i^2} &= \frac{\frac{a^2}{4}\Big\{\mathrm{E}\Big[\sum_{i=1}^{35} w_{ki}X_i - \mathrm{E}\Big(\sum_{i=1}^{35} w_{ki}X_i\Big)\Big]^2 + \sigma_b^2\Big\}}{\sigma_b^2\sum_{i=1}^{35} w_i^2}\\ &= \frac{a}{4}\Big(\frac{1}{\sum_{i=1}^{35} w_i^2} + \frac{O(1)}{\sigma_b^2}\Big).\end{aligned} \tag{8}$$

Since σ_b^2 is a constant, (??) can be maximized by minimizing $\sum_{i=1}^{35} w_i$. This means that replacing those frequently used neurons, which with large $\sum_{i=1}^{35} w_i$, by "naive" neurons, which with relatively smaller $\sum_{i=1}^{35} w_i$, can maximize the Middle layer's output. This can further help to produce a more desirable output at the third layer.

However, with regard to noise recalling, replacing neurons that are not used very often (those neurons with small weights) generates a smaller NSSE. This is because when it takes more epochs for the network to converge, the network also has more time to learn. Hence the long-term memory can be developed more sufficiently.

5. CONCLUSION AND FUTURE WORK

In this paper we have analyzed the performance of various neurogenetic behaviors after building a three layer feed forward network and simulating the neurogenesis process. It is demonstrated that the learning speed of the network can be improved while the number of replaced neurons increases, or the replaced neurons are distributed in a scattered manner, or the replaced neurons are those with large weights. However, with regard to recalling with noisy inputs, the faster the network converges, the worse it performs under noise perturbing.

Further research is necessary for more conclusive results.

First of all, the simulation model needs to be further refined. Currently, we only use back propagation algorithm to train the network, which is a supervised learning procedure. However, the memory development in human brain is actually an unsupervised learning process. Thus an unsupervised training algorithm, for example, the Hebb's learning rule would be more desirable for the simulation.

Besides, we did not include lateral connections between neurons in the same layer in our model. However, in real life scenario, lateral connections do exist between some neurons in the same layer. Taking this factor into consideration may produce a more realistic model and thus a more informative result.

ACKNOWLEDGEMENT

The author would like to thank Prof. Miranker for his guidance and advice.

REFERENCE

R. A. Chambers, M. N. Potenza, R. E. Hoffman and W. Miranker, (2004) "Simulated apoptosis/neurogenesis regulates learning and memory capabilities of adaptive neural networks," *Neuropsychopharmacology*, Vol. 29, Issue 4, pp. 747-758, Apr. 2004

W. Miranker, (2004) "Apoptosis/Neurongenesis favorably informs memory development," *Technical Report,*, Yale University, DCS TR 1234, 2004

S. Haykin, (1999) *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999

P. S. Eriksson, E. Perfilieva, T. B. Eriksson, A. M. Alborn, C. Nordborg, D. A. Peterson, and F. H. Gage, (1998) "Neurongenesis in the adult human hippocampus," *Nature Medicine*,

Vol. 4, No. 11, pp. 1313 - 1317, Nov. 1998

Appendices I — Noise Version of A, B, H, K and Ψ

Table 8:
Noise Version of A

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

Table 9:
Noise Version of B

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

Table 10:
Noise Version of H

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

Table 11:
Noise Version of K

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

Table 12:
Noise Version of Ψ

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

An Enhanced ASLD Trading System For Traders and Portfolios

James Stewart and Bin Zhou
Yale University
New Haven, CT 06520

Abstract

An Adaptive Supervised Learning Decision Trading (ASLD) System has been devised by Xu, Cheung and Hung in 1997 that is trained by best past investment decisions and then makes a investment decision directly for the future. We use a Monte Carlo method to test the ability of the ASLD System to make money over a larger number of possibilities. Instead of only following one path to test the neural network, we consider different possible future situations. Analyzing the results in terms of mean returns, standard deviation and distribution, suggest further possibilities to enhance the accuracy of the prediction.

1. INTRODUCTION

There are many kinds of neural network trading systems that use different methodologies and different investment strategies to maximize trader profits. In 1997, Xu and Cheung devised a trading system to learn the desired past investment decisions through a supervised learning neural network (which can be concluded through the dataset) that is called "Adaptive Supervised Learning Decision (ASLD) network". This system is trained by the desired investment decision teaching signal, and then it can make the decision to "maximize the profit". They implement the ASLD system by means of an adaptive Extended Normalized Radial Basis Function (ENRBF) network and train the network by the Coordinated Competitive Learning (CCL) algorithm which is described by L. Xu, and Y. M. Cheung, 1997.

Employing the ASLD trading system, we use Monte Carlo method to generate different sample paths to predict different future returns. The results may be analyzed in terms of mean returns, standard deviation and distribution, which may further enhance the accuracy of the prediction.

2. ASLD TRADING SYSTEM

To demonstrate the Monte Carlo method effect in the prediction ability of the neural network, we focus on only one asset for clarity. We assume the price of this asset at time t is: z_t. We suppose exactly one currency can be invested each day. At time t, the trader can invest one currency on the asset, hold the asset without further investment or sell the asset. I_t denotes the trader's investment signal at time t.

1) At time t, if the asset will keep on decreasing in the next q time steps (prediction), the trader's right choice is to sell it out at time t to avoid the devaluation. In this case, we let I_t= -1, which means the trader should sell the asset at time t.
2) At time t, if the asset will keep on increasing in the next q time steps (prediction), the trader's right choice is to buy the asset at time t to make profit. In this case, we let I_t= 1, which means the trader should invest on this asset at time t.
3) Otherwise, the trader's right choice is to neither buy nor sell the asset at time t, which means he will take a neutral position. In this case, we let I_t=0.

Specifically, I_t is determined by:

$$I_t = \begin{cases} -1, \text{ if } z_t > z_{t+1} > \ldots > z_{t+q-1} > z_{t+q} \\ 1, \text{ if } z_t < z_{t+1} < \ldots < z_{t+q-1} < z_{t+q} \\ 0, \text{ otherwise} \end{cases}$$

We suppose there exists a nonlinear relation among I_t, I_{t-1} and the asset's price history: $I_t = f[I_{t-1}, z_t, z_{t-1}, \ldots\ldots z_{t-d+1}]$ where d is the time lag which denotes the past d steps which may influence the invest signal at time t. We shall use a neural network to approximate this function f.

3. EXTENDED NORMALIZED RBF NETWORK WITH COORDINATED COMPETITIVE LEARNING

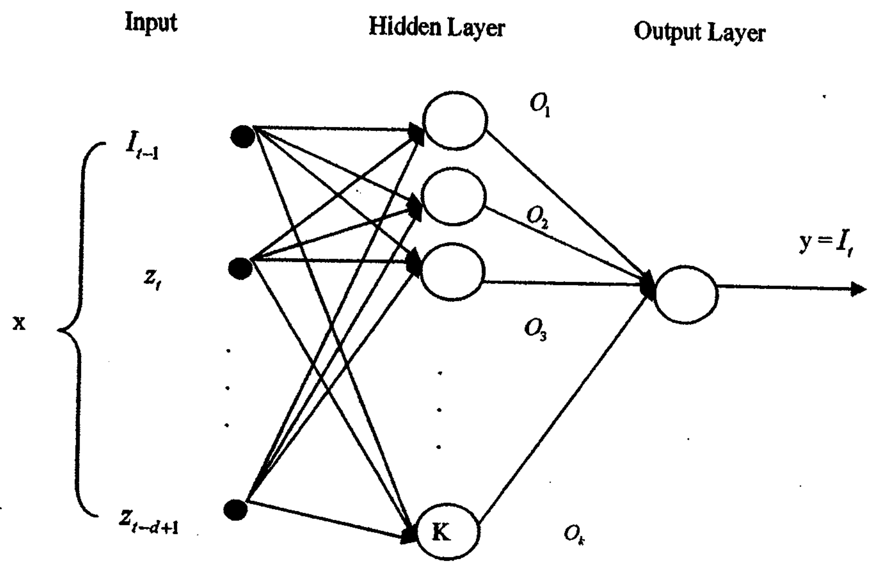


Figure 1. Structure of an ENRBF network

As shown in Figure 1, the network consists of three layers: an Input Layer, a Hidden Layer and an Output Layer. There are $d+1$ neurons in the Input Layer, K neurons in the Hidden Layer and only one neuron in the Output Layer.

According to the ENRBF methodology,

$$y = f(x) = \frac{\sum_{j=1}^{k}(w_j^T x + c_j)\exp[-0.5(x-m_j)^T \Sigma_j^{-1}(x-m_j)]}{\sum_{j=1}^{k}\exp[-0.5(x-m_j)^T \Sigma_j^{-1}(x-m_j)]}$$

Where x, y correspond to the input, output. w_j is a $(d+1)\times 1$ matrix which denotes the j th neuron's input weights. c_j is the bias of that neuron. m_j is the center vector of that neuron. Σ_j is the receptive field of the activation function $\phi(.)$. We usually assume that $\Sigma_j = \sigma_j^2 I$ and that σ_j^2 is estimated roughly.and heuristically. In fact, we use Σ_j instead of σ_j^2 in the Multivariate Gaussian Function. Simon Haykin regards this parameter as the Multivariate Gaussian Function's "width" (Haykin, Simon, "Neural Networks, a Comprehensive Foundation.", page 275-276). In the ENRBF network, we use $(x-m_j)^T \Sigma_j^{-1}(x-m_j)$ to define the induced local field. $\phi(x)$= exp (-0.5x). w_j, c_j, m_j, Σ_j are parameters of the network which need to be learned through training.

We use the Coordinated Competitive Learning (CCL) algorithm to perform the training process. There are two kinds of CCL algorithms: Batch CCL Algorithm and Adaptive CCL Algorithm. We use the former for the initial training of the network based on the past data until the time t-1. Then we can predict the best trading decision based on the trained network. We use the latter algorithm to update the network. (Please check the paper "Adaptive Supervised Learning Decision Networks for Trading and Portfolio Management," published by L. Xu, and Y. M. Cheung in 1997 for detail)

4. AN ENHANCED ASLD TRADING SYSTEM BASED ON MONTE CARLO METHOD

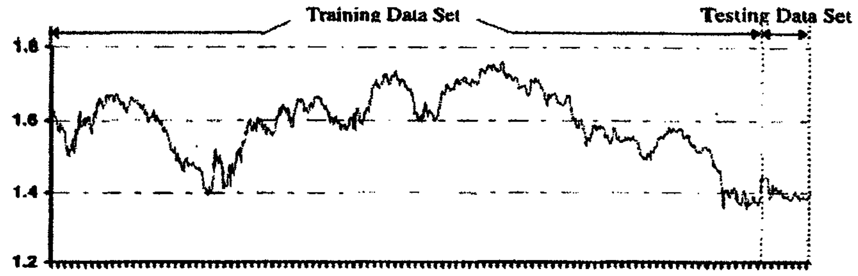


Figure 2. The USD-DEM rate series of 1096 data points. Each horizontal bar represents 10 data points

Here is a figure comes from "*Adaptive Supervised Learning Decision Networks for Traders and Portfolios*" written by Xu and Cheung. The authors use the past USD-DEM rate data set from Foreign Exchange Market to train the network and then use the trained network to produce a predicted testing data about the USD-DEM rate series.

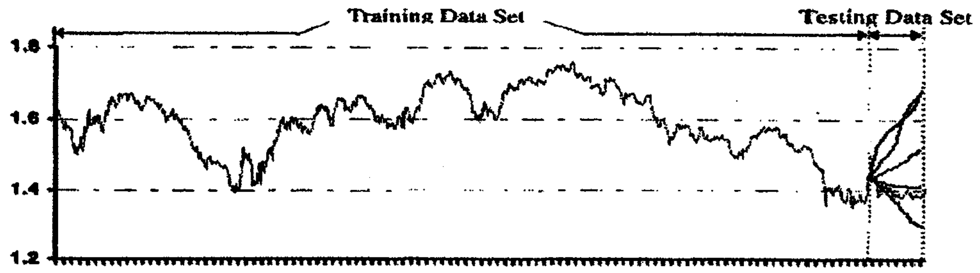


Figure 3. the Neural Network with Monte Carlo Method

We use the figure 3 above to illustrate our approach. Based on the figure2, it is possible to use Monte Carlo method to generate different branches to predict different future returns at time t. Then we can analyze each possible gain generated by the network through each predicted branch, the results can then be analyzed in terms of mean returns, standard deviation and distribution. We can not test our idea because of the problem of the Coordinated Competitive Learning (CCL) algorithm. We will discuss the algorithm later.

If we can implement the neural network, we may get the possible distribution result as in Figure 4:

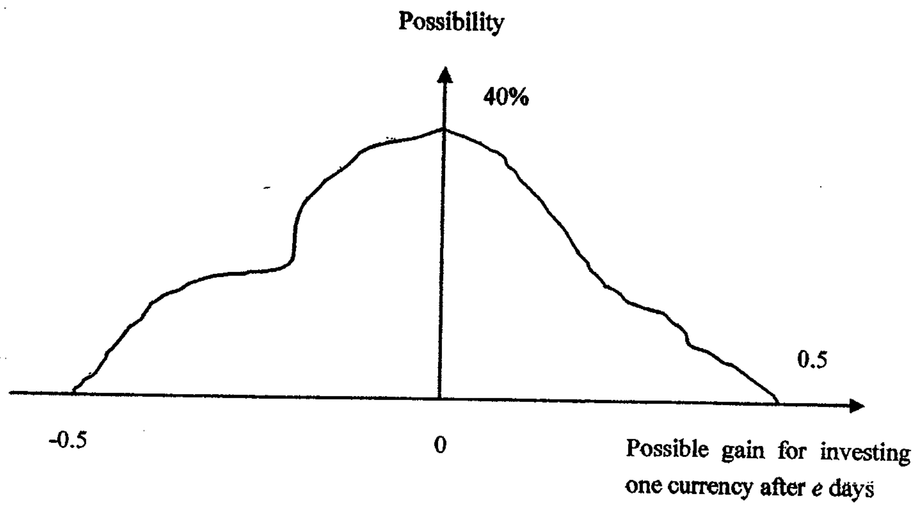


Figure 4. The possible distribution result generated by Monte Carlo method

Our data are drawn from the foreign exchange market, which contains detailed information on the Eur/Usd rate from January 1998 through December 2004. We assume that Euro is a normal good and we consider the Eur/Usd rate to be the price of Euro.

For any given day t, we examine the Eur/Usd rate in the next q days from the Eur/Usd rate data set to determine a rational behavior on the day t. If the rate will keep on increasing in the next q days, we should invest on Euro. If the rate will keep on decreasing in the next q days, we should sell Euros for US dollars. Otherwise, we should neither buy nor sell. The behavior we got is the expected output I_t.

We believe that the asset's price history in the past d days and the investor's behavior on the day t-1 can influence the investor's behavior on the day t. So there should be a relation among them: $I_t = f[I_{t-1}, z_t, z_{t-1}, \ldots\ldots z_{t-d+1}]$ where d is the time lag which denotes the past d days, which may influence the invest signal on the day t. For convenience, we use the log value instead of the actual past d days' price and transform the value into normalized inputs between 0 and1 (we compute the minimum price $MinP$ and the maximum price $MaxP$ during 1998 - 2004, then we compute the normalized inputs: $p_{normalized} = (P - MinP)/(MaxP - MinP)$. If P equals to $MaxP$, the normalized value is 1. If P equals to $MinP$, the normalized value is 0).

We try to implement the Extended Normalized RBF network with Coordinated Competitive Learning algorithm, but we have been unsuccessful. According to the algorithm, we have to compute the matrix Σ_j for the jth neuron and use the inverse of the matrix, Σ_j^{-1}, to update the neural network. We find that the matrix Σ_j is not invertible for some neurons because Σ_j can be a zero matrix in some situations.

Here is a part of the Coordinated Competitive Learning (CCL) algorithm. It is used to updated the parameter Σ_j. (Please check the paper "Adaptive Supervised Learning Decision Networks for Trading and Portfolio Management," published by L. Xu, and Y. M. Cheung in 1997 for detail algorithm)

$$\begin{cases} a_j^{new} = \dfrac{1}{N}\sum_{i=1}^{N} I(j \mid x_i) \\[2ex] m_j^{new} = \dfrac{1}{a_j^{new} N}\sum_{i=1}^{N} I(j \mid x_i) x_i \\[2ex] \Sigma_j^{new} = \dfrac{1}{a_j^{new} N}\sum_{i=1}^{N} I(j \mid x_i)(x_i - m_j^{new})(x_i - m_j^{new})^T \end{cases}$$

From the algorithm above, we find if there is only one $I(j \mid x_i)$ equals to 1 (others equal to 0), $c_j^{new} = 1/N$, $m_j^{new} = x_i$, so that Σ_j^{new} becomes a zero matrix (not invertible).

We believe there may be some mistakes about the Coordinated Competitive Learning algorithm in the original paper which presented by L. Xu in 1997. Because of this, we can not develop our further research.

6. CONCLUSION AND FUTURE DIRECTIONS

We have been unsuccessful to employ the Extended Normalized RBF network with Coordinated Competitive Learning algorithm. In the future work, if we can not fix the error in the Coordinated Competitive Learning algorithm, we will use another learning algorithm (maybe back-propagation algorithm) in order to test our idea. We still believe that by using Monte Carlo method, we can test the ability of the Adaptive Supervised Learning Decision Trading (ASLD) System to make money over a larger number of possibilities. It is partly because we consider different possible future situation instead of only following one path to test the neural network. By analyzing the mean returns, standard deviation and distribution of each branch, we can get a general view about how well the trading system will perform on a variety of different possible price evolutions.

REFERENCES

1. Y. Bengio, "Training a neural network with a financial criterion rather than a prediction criterion," in Proc. Fourth Int. Conf. Neural Networks Capital Markets, A. S.Weigend, Y. Abu-Mostafa, and A.-P. N. Refenes, Eds., 1997, pp. 36–48.
2. L. Xu, "RBF nets, mixture experts, and Bayesian ying-yang learning," Neurocomput., vol. 19, no. 1–3, pp. 223–257, 1998.
3. L. Xu and Y. M. Cheung, "Adaptive supervised learning decision networks for traders and portfolios," J. Comput. Intell. Finance, vol. 5, no. 6, pp. 11–16, 1997.
4. L. Xu, M. I. Jordan, and G. E. Hinton, "An alternative model for mixture of experts," in Advances in Neural Information Processing Systems, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Cambridge, MA: MIT Press, 1995, pp. 633–640.
5. L. Xu, and Y. M. Cheung, "Adaptive Supervised Learning Decision Networks for Trading and Portfolio Management," Computational Intelligence in Finance, 1997
6. Haykin, Simon, "Neural Networks, a Comprehensive Foundation." New Jersey: Prentice Hall, 1999