Neural Net Applications

Willard L. Miranker
April 2008
TR-1404

Table of Contents

Classification of Phonetic Feature Ensembles using Asymmetric Neighborhood Functions and Affine Transformations in $\mathbb{R}^2$

David Smalling

***Abstract*— It is well-known that both Self-Organizing Maps and K-Means Clustering have the ability to be effectively modified in ways that allow their accuracy in specific applications to be substantially amplified. We use Self-Organizing Maps to classify acoustic features over various domains, and present a methodology to improve its performance using topological information. We utilize group theory to investigate the behavior Gaussian forms and to derive an asymmetric neighborhood function that learns from the topological structure of a data-set. We present an analysis of clustering in speech frequency data and show that the Self-Organizing Map proves itself as a useful tool in this area. In particular we present a mechanism that substantially improves the convergence of speaker-type classes by using a dynamic, self-aligning neighborhood function.**

I. Introduction

A Self-Organizing Map (SOM) is a type of competitive learning based Neural Network that consists of components called nodes. Associated with each node is a synaptic weight vector of the same dimension as the input data vectors and a position in the map space. A neuron that wins the competition is called a winning neuron. In the context of an SOM winning means that a neuron is the closest to the input sample according to some distance metric. In an SOM the neurons are placed at nodes in a lattice that is usually 2 dimensional, and are then allowed to converge to stable points which should minimize the distance between each synaptic weight and the elements in the input space that belong to its cluster. The stabilized node to which a point from the input lies closest to may act as a category for that input pattern. The resulting categories form what is known as the *self-organized feature space*. [1,2,7,8,9,10]

A. Phonetics

Phonetics involves the study of the sounds of human speech and is mostly concerned with the audible properties of speech sounds, their production, audition and perception. Care should be taken not to confuse phonetics with phonology, which emerged from phonetics, and is the study of sound systems and abstract sound units. Phonetics deals with sound elements themselves rather than the contexts in which they are used in languages.

When identifying dissimilar sounds such as human vowels, the ears are most sensitive to peaks in the signal spectrum. These resonant peaks in the spectrum are called formants. The frequencies of these peaks correspond to resonant frequencies of vocal tract. Each vowel has different formant frequencies and bandwidths. Furthermore, every human being has his or her own unique formant frequencies and bandwidths. Using these characteristics of vowel sound production mechanism, a band-limited impulse train can be used as a glottal source. [3,4,5,6]

B. The Peterson/Barney data-set

The Peterson/Barney vowel data-set consist of measurements of four acoustic features, F0, F1, F2 and F3 values, for two repetitions of 10 different vowels by 76 speakers of British English (33 adult males, 28 adult females, and 15 children).

The Peterson/Barney data-set is formatted as follows: there are 1520 vowel tokens; each row represents a vowel, and there are additional features (columns) for each token as follows:

Column	Information	Key
1	Speaker Type	1 = Male, 2 = Female 3 = Child
2	Speaker Number	Unique id for each speaker
3	Vowel Identity	1 = *i*, 2 = ɪ, 3 = e, 4 = æ 5 = ʌ, 6 = ɑ, 7 = c' 8 = ʊ, 9 = u, 10 = ɛ
4	F0	Fundamental Frequency
5	F1	First Formant
6	F2	Second Formant
7	F3	Third Formant

Vowels may be categorized on the basis of various articulatory or acoustic features. One standard articulatory-based representation of the vowel space is shown Figure 1, which is known as the International Phonetic Alphabet vowel chart. The lateral displacement (front-ness) and height of the human tongue during speech underpins some of the characteristics of the frequency blends of human speech. This chart (Fig. 1) represents features the relative front-ness (left/right) and

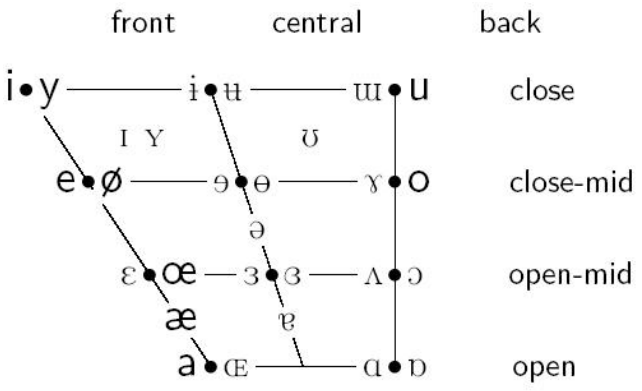


Fig. 1. Articulatory Vowel Chart [18]

relative height (up/down) of the tongue. Vowels in the upper left are high and front; vowels in the lower right are low.

The four characterizing measurements provided in the data for each vowel token are the fundamental frequency (in Hertz) and the frequencies of the first three formants. As mentioned above Formants are peaks in the acoustic frequency spectrum and depend on the shape of the vocal tract at the time of production. F0, or fundamental frequency represents the pitch of the speakers voice. The first two formants, F1 and F2, are usually enough to disambiguate a vowel because each vowel has different F1 and F2 values. [14,15,16,17,18]

In the following sections with will attempt to use an SOM to classify speech data into vowels and into child/male/female. We will also show that there exist structural properties within the data set that may be exploited in the task of optimizing the clustering of the data.

II. Performance Evaluation Methodology

One way to evaluate the quality of the the clustering is by comparing its same vs. different judgments to those of the target. That is, for any pair of distinct vowels v1 and v2, we can check whether or not the clustering puts them in the same cluster, and whether or not this is correct (by looking at the true vowel identities). This will yield four cases:

- true positives: v1 and v2 are in the same cluster and are the same vowel
- false positives: v1 and v2 are in the same cluster but are not the same vowel
- true negatives: v1 and v2 are in different clusters and are not the same vowel
- false negatives: v1 and v2 are in different clusters but are the same vowel

In our experiments we use both convergence rate and the usual statistical methods to evaluate the performance of each method's classification. In particular, we determine the following four evaluators of performance.

Precision = $\frac{NumberofTruePositives}{NumberofTruePositives+FalseNegatives}$

Recall = $\frac{NumberofTruePositives}{NumberofTruePositives+FalsePositives}$

Accuracy = $\frac{NumberofTruePositives+TrueNegatives}{Totalnumberofmeasurements}$

f-Score = $\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$

Qualitatively, accuracy is the level of conformity of a measured quantity to its actual value. Precision on the other hand is the degree to which further measurements or calculations show parallel results. The f-Score is a measure of a test's accuracy. It is defined as the ratio of twice the product of precision and recall to the sum of precision and recall. That is, the f-score is simply the harmonic mean of precision and recall.

III. Experiment 1: Classification of Vowels and Speakers in the Peterson & Barney Vowel Data using a Self-Organizing Map.

A. Design

In the first (preliminary) experiment the usual framework of SOM design was used to classify vowels based on first formant and second formant frequencies. The relevant quantities are:

Distance Metric, $d_{j,i}^2 = \| \mathbf{r}_j - \mathbf{r}_i \|^2$

Neighborhood Function, $h_{j,i(\mathbf{x})}(n) = e^{-\frac{d_{j,i}^2}{2\sigma^2(n)}}$

Where, $\sigma(n) = \sigma_0 e^{-\frac{n}{\tau_1}}$

Learning-rate Parameter, $\eta(n) = \eta_0 e^{-\frac{n}{\tau_2}}$

Algorithm 1 Self Organizing Map

1: Choose random values for the initial synaptic weight vectors $\mathbf{w}_j(0)$
2: Draw a sample **x** from the input space (the set of all speech frequency measurements) with a certain probability; the vector **x** will represent an activation pattern.
3: **while** There are noticeable changes in the feature map **do**
4: Search for the matching neuron $i(\mathbf{x})$ at time step n. This is the neuron closest to the input sample. The winning neuron is obtained by: $i(\mathbf{x}) = argmin\|\mathbf{x}(n) - \mathbf{w}_j\|$, j = 1,2...,*l*
5: Update the winning neuron using the update formula $\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i(\mathbf{x})}(\mathbf{x}(n) - \mathbf{w}_j(n))$
6: **end while**
7: return $\{\mathbf{w}_j\}_{j=1}^{l}$

In this experiment, so as to maintain a value of $\eta(n)$ on the interval $[0.01, 0.1]$, η_0 was set to 0.1 and $\tau_2 = \frac{n_{max}}{\ln(10)}$, where n_{max} is an upper-bound on the number of iterations performed in the experiment.

B. Results

In this experiment σ_0 was set equal to 200, and convergence was achieved over approximately 16,000 and 23,000 iterations

for the vowel classification and the speaker-type classification respectively. Vowel classification refers to classifying the data into then unique clusters each of which will signify one of the ten vowels in the IPA vowel chart in Figure 1. On the other hand, speaker-type classification refers to classifying the data into clusters according to whether the speaker was a child, an adult male, or and adult female. Figures 2 and 3 show the results of the clustering with the points color coded to represent points in the same cluster. The performance evaluators from section II yielded inferior results which are presented in the following tables.

Classification:	Vowels
Precision	0.38
Recall	0.41
F-score	0.39
Accuracy	0.88

Classification:	Speakers
Precision	0.37
Recall	0.53
F-score	0.44
Accuracy	0.51

IV. Experiment 2: Classification of Vowels and Speakers in the Peterson & Barney Vowel Data using K-Means Clustering.

A. Design

The data was also clustered using the of K-Means clustering algorithm. This experiment was performed in order to have a benchmark using a deterministic algorithm with which we can compare the results obtained from the SOM algorithm.

The most common form of the K-Means clustering algorithm uses an iterative refinement heuristic known as Lloyd's algorithm. Lloyd's algorithm starts by partitioning the input points into K sets, either at random or using some heuristic. It then calculates the mean point, or centroid, of each set. It constructs a new partition by associating each data point with the closest centroid. Then the centroids are recalculated for the new clusters. This is repeated by alternate application of these two steps until convergence, which is obtained when points no longer switch clusters (or alternatively centroids are no longer changed).[7,8,9,10]

The 3 means for the speaker-type classification were randomly initialized. However, so as to catalyze the convergence of the algorithm, the 10 means for the vowel clustering were initialized on the points in the table below.

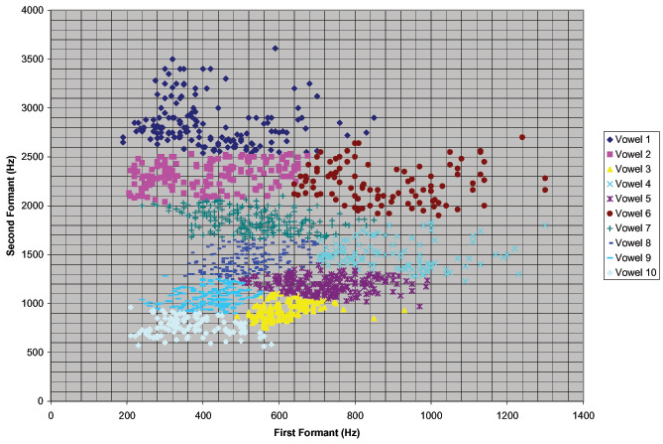


Fig. 2. Vowel Clustering of Peterson/Barney Data using SOM

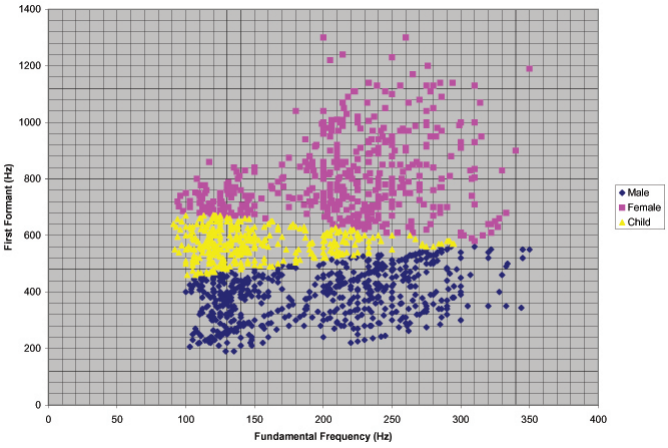


Fig. 3. Speaker Clustering of Peterson/Barney Data using SOM

First Formant (Hz)	Second Formant (Hz)
400	2750
420	2300
600	900
900	1501
750	1250
900	2200
500	1803
500	1509
450	1000
350	807

B. Results

In this experiment convergence was usually achieved over approximately 31 and 15 iterations (of the K-means algorithm) for the vowel classification and the speaker-type classification respectively. Since in each iteration of the K-means algorithm we evaluate the properties of every point in the input space. This process performed in this algorithm is equivalent to approximately $15 \cdot 1,520 = 22,800$ and $31 \cdot 1,520 = 47,120$ for the vowel classification and the speaker-type classification respectively. Figures 4 and 5 show

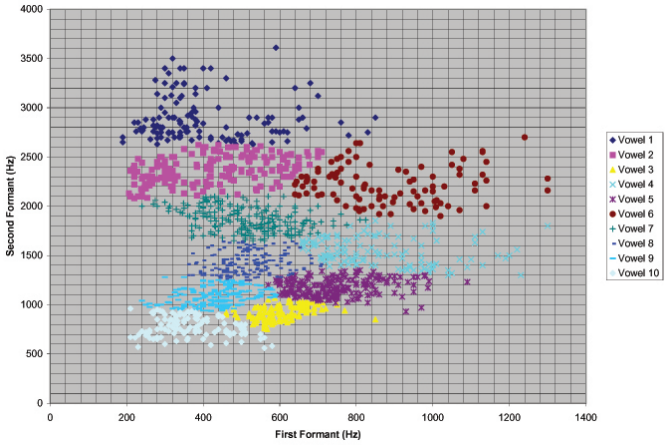


Fig. 4. Vowel Clustering of Peterson/Barney Data using K-Means

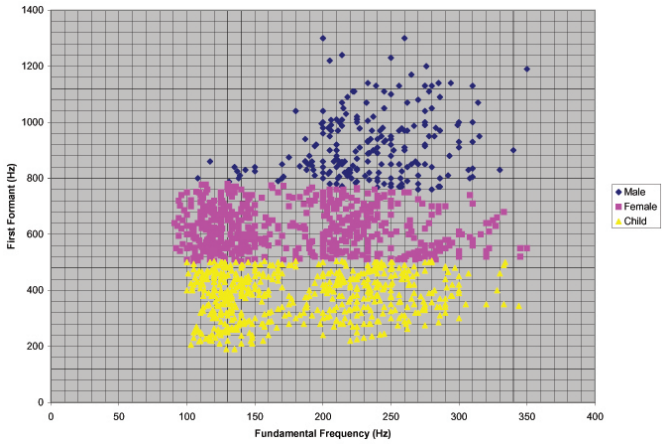


Fig. 5. Speaker Clustering of Peterson/Barney Data using K-Means

the results of the clustering with the points color coded to represent points in the same cluster. The Performance Evaluators yielded the following results.

Classification:	Vowels
Precision	0.36
Recall	0.40
F-score	0.38
Accuracy	0.86

Classification:	Speakers
Precision	0.30
Recall	0.42
F-score	0.36
Accuracy	0.56

The clustering obtained by using K-means clustering is inferior to that obtained by SOM.

V. Generalization of the Gaussian Neighborhood function

We now generalize the SOM neighborhood function through a series of relaxation conditions. The commonly used Gaussian neighborhood function is given by:

$$h_{j,i(\mathbf{x})}(n) = e^{-\frac{d_{j,i}^2}{2\sigma^2(n)}},$$

where $d_{j,i}^2 = \| \mathbf{r}_j - \mathbf{r}_i \|^2$

In the particular case of speech data to be classified over speaker-type (male/female/child), the classification was done using the the fundamental frequency and the first formant. Thus the points to be classified are represented in $\mathbb{R}^2$. Hence, $d_{j,i}^2$ reduces to $(x_j - x_i)^2 + (y_j - y_i)^2$, where x_n and y_n are the frequency coordinates of the data-set in the Euclidean plane. Thus,

$$h_{j,i(\mathbf{x})}(n) = e^{-\frac{(x_j - x_i)^2 + (y_j - y_i)^2}{2\sigma^2(n)}}$$

The neighborhood function may be generalized so as to allow axial distortions and rotations by assigning a variance function $\sigma_i(n)$ for each of x and y, adding an xy term and parameterizing the coefficients of each term by an angle parameter to implement the aforementioned rotations. This generalized form is given by

$$h_{j,i(\mathbf{x})}(n) = e^{-\left(a(n)\cdot(x_j - x_i)^2 + b(n)\cdot(x_j - x_i)(y_j - y_i) + c(n)\cdot(y_j - y_i)^2\right)}$$

where the matrix $\begin{bmatrix} a & b \\ b & c \end{bmatrix}$ is chosen to be positive definite.

Now if we let

$$a(n) = \left(\frac{\cos\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\sin\theta}{\sigma_y(n)}\right)^2$$

$$b(n) = -\frac{\sin 2\theta}{\sigma_x^2(n)} + \frac{\sin 2\theta}{\sigma_y^2(n)}$$

$$c(n) = \left(\frac{\sin\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\cos\theta}{\sigma_y(n)}\right)^2$$

Then the generalized neighborhood function in $\mathbb{R}^2$ may be rotated by an angle θ. Whence our generalized form is given by

$$\begin{aligned} h_{j,i(\mathbf{x})}(n) = \exp\Bigg(- \Bigg(& \left[\left(\frac{\cos\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\sin\theta}{\sigma_y(n)}\right)^2\right]\cdot(x_j - x_i)^2 \\ & + \left[-\frac{\sin 2\theta}{\sigma_x^2(n)} + \frac{\sin 2\theta}{\sigma_y^2(n)}\right]\cdot(x_j - x_i)(y_j - y_i) \\ & + \left[\left(\frac{\sin\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\cos\theta}{\sigma_y(n)}\right)^2\right]\cdot(y_j - y_i)^2\Bigg)\Bigg) \end{aligned}$$

The ratio $\frac{\sigma_x(n)}{\sigma_y(n)}$ changes the relative axial variance (the spread along the $x - axis$ compared to the spread along the $y - axis$). The effect of variations in these parameters is illustrated in figures 6 through 9. The additional degrees of freedom provided by this generalized neighborhood function is the basis for the derivation in the next section.

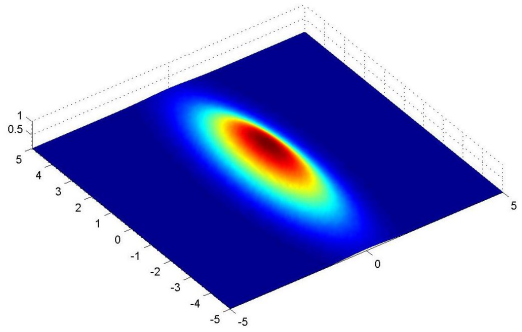


Fig. 6. Sample Neighborhood function 1
$\theta = 0,\ \sigma_x = 1,\ \sigma_y = 3$

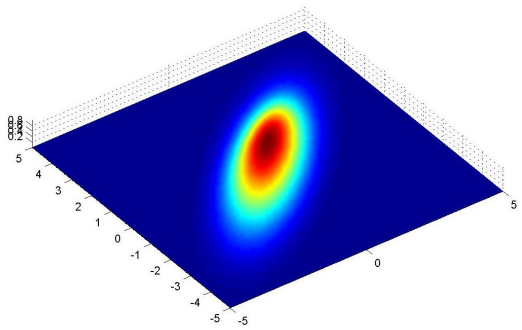


Fig. 7. Sample Neighborhood function 2
$\theta = 45,\ \sigma_x = 1,\ \sigma_y = 3$

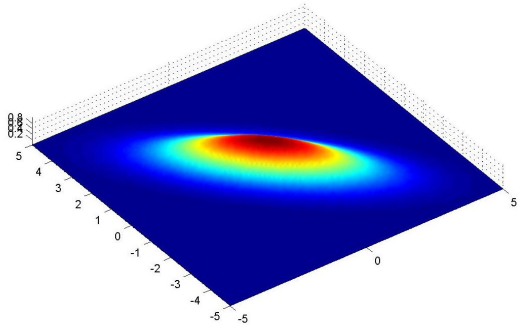


Fig. 8. Sample Neighborhood function 3
$\theta = 45,\ \sigma_x = 3,\ \sigma_y = 1$

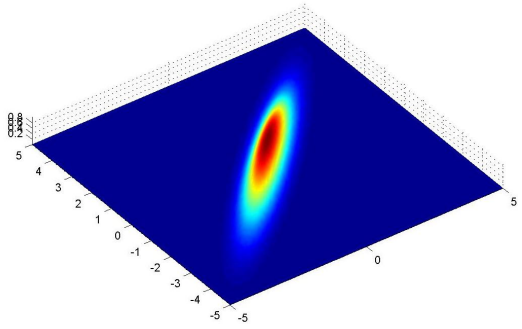


Fig. 9. Sample Neighborhood function 4
$\theta = 135,\ \sigma_x = 3,\ \sigma_y = 0.5$

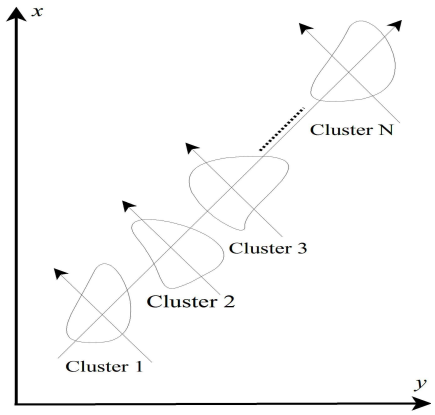


Fig. 10. Generalization of the distribution of speaker-type clustering

VI. Derivation of an Improved Neighborhood Function for Optimized Convergence

Using the generalized neighborhood function presented in Section V, we show that it is possible, but non-trivial to derive an optimized/specialized neighborhood function in the context of a specific model given a hypothesis about the clustering. This is done by varying the angle and relative spread along each axis of the gaussian. We then propose a methodology for implementing a generalized neighborhood function with parameters that learn from the data-set and provide faster and more accurate convergence using topological features within the the data-set.

We can see from the experiments in Sections III and IV that when the fundamental frequency and first formant data are classified into 3 clusters the emerging clusters may reveal information about the gender and age of the speakers, that is, the clusters represent groups corresponding to male, female and children. It is clear from the results of the experiments in Sections III and IV that there exists a line of action such that the each cluster is traversed. That is, the line of regression on the geometric centers of each clusters will have sum of square error that is close to zero. This observation may be exploited to yield improved performance through subtle modifications to the neighborhood function.

Consider an experiment in which data is to be clustered and there exists a general hypothesis about the form of the clustering or the distribution of the feature space. Let us take the simple case which is similar to the Peterson / Barney vowel data over speakers and hypothesize that the clusters will be distributed approximately as depicted in Fig 10. Note that this hypothesis is motivated by the results of the previous experiments.

Note that the cluster separation in Fig 10 is exaggerated and that there is likely to be more mixing between the points of different clusters, that is, the convex hulls of the points in the different clusters are likely to overlap.

Now, during the updating phase of the SOM algorithm we want the neighborhood functions to be axially aligned normal to the line of action through the hypothesized clusters

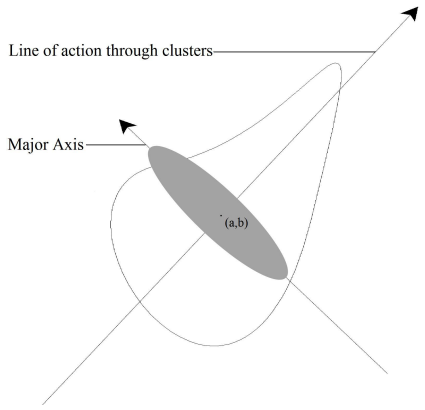


Fig. 11. Biased Neighborhood function at point (a,b)

as in Fig 11. A line of action is a the line obtained by linear regression on the geometric centers of the clusters. This assertion is motivated by the desire to have the major axis of the neighborhood function lie as parallel as possible to lines of "best-separation" of adjacent clusters. Whether the $x - axis$ or $y - axis$ is set to be the major axis is irrelevant since the final angular alignment is made by means of variations in θ. The minor axis is depicted in Figure 12 as the predominant line that crosses the three clusters. Without loss of generality assume that the major axis is the image of the $x - axis$ and that the minor axis is the image of the $y - axis$. Under this assumption we seek

$$\lim_{n\to+\infty} \frac{\sigma_{major}(n)}{\sigma_{minor}(n)} = \lim_{n\to+\infty} \frac{\sigma_x(n)}{\sigma_y(n)} = \infty,$$

Where n is the current iteration.

While obeying the usual rules of SOM that state

$$\lim_{n\to+\infty} \sigma_x(n) = \lim_{n\to+\infty} \sigma_y(n) = 0$$

Qualitatively this means that the peakedness of the major axis of the neighborhood function becomes more pronounced as the synaptic weights approach the limiting values even though the overall spread of the neighborhood function is decreasing. To achieve efficient convergence of the ratio $\frac{\sigma_x(n)}{\sigma_y(n)}$ the decay constant of $\sigma_x(n)$ must be much greater than that of $\sigma_y(n)$. So we impose the condition $\tau_{\sigma_x} \gg \tau_{\sigma_y}$. Where τ_i is the decay rate of σ_i.

Analytically it makes little difference if we set $\sigma_y(0) = \sigma_x(0)$ since it is the divergence of the *ratio* that we are interested in.

Let $\Phi = \frac{\tau_{\sigma_y}}{\tau_{\sigma_x}}$ denote the divergence rate of the variances along the x and y axes. (Larger Φ leads to divergence for the term in the numerator)

Fig 12 shows the unclassified Peterson / Barney vowel data with a line of action through the clusters derived using linear regression and a partitioning which is an approximation of the convex hulls of each cluster.

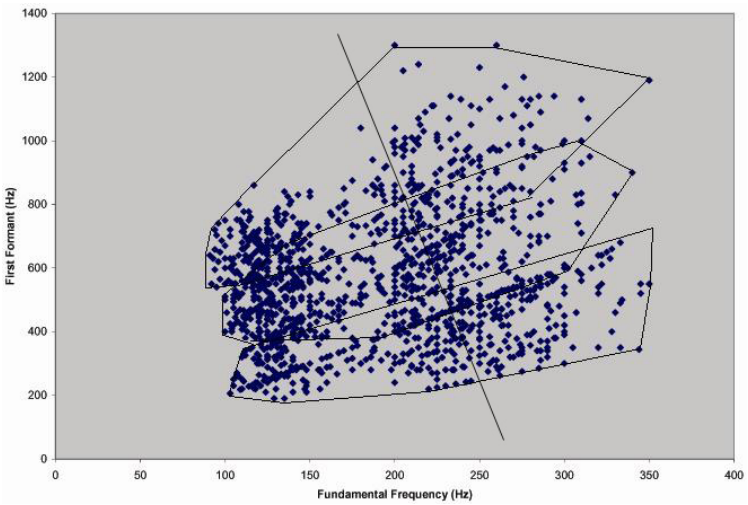


Fig. 12. Approximate Partitioning of Peterson / Barney data into 3 Clusters

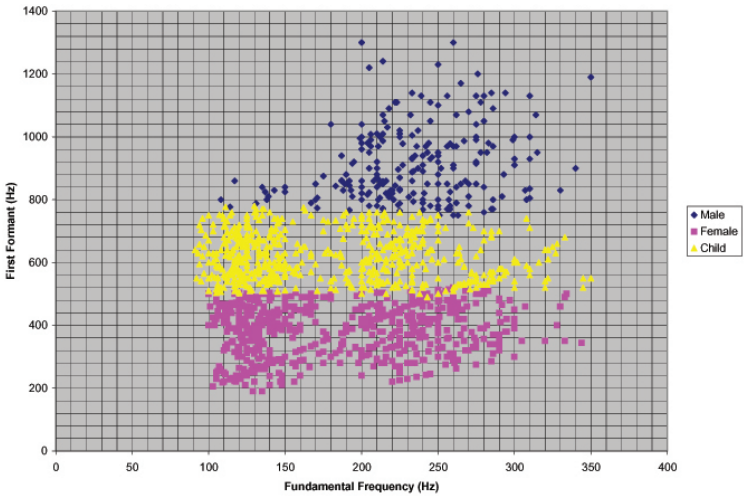


Fig. 13. SOM on Peterson / Barney vowel data (speaker-type) using the Optimized Neighborhood function

A. *Experiment 3: SOM on Peterson / Barney vowel data (speaker-type) using the Optimized Neighborhood function*

The speaker data was clustered using an SOM with an asymmetric neighborhood function whose parameters were chosen so that the major axis of the gaussian in the SOM lay perpendicular to the line of action in Fig 12. The accuracy and convergence rate of the resulting clustering was improved substantially and is depicted in Fig 13.

In this final classification the parameters used are
$\theta = 0.5rad\ (\approx 28.8°)$
$\sigma_x(0) = \sigma_y(0) = 500$
$\tau_{\sigma_y} = 20,000$
$\tau_{\sigma_x} = 10 \cdot \tau_{\sigma_y} = 200,000$

Thus, for this experiment the variance divergence parameter $\Phi = 10$.

We ran the algorithm 10 times and clusters converged in average 7133 iterations and yielded superior results (shown below).

Classification:	Speakers
Precision	0.48
Recall	0.52
F-score	0.40
Accuracy	0.61

These results represent a 10% increase in accuracy from the unregulated SOM algorithm.

VII. Groups of Generalized Neighborhood Functions

We now perform an analysis of the characteristic matrices of generalized neighborhood functions. Using this analysis we are then able to provide insight into the emergence of matrix groups whose underlying sets characterize families of optimized neighborhood functions on arbitrary clusters or data-sets.

Consider the usual 2-dimensional generalized neighborhood function given by

$$h_{j,i(\mathbf{x})}(n) = e^{-\left(a\cdot(x_j-x_i)^2+b\cdot(x_j-x_i)(y_j-y_i)+c\cdot(y_j-y_i)^2\right)}$$

The parameters a, b and c are functions of n and θ given by

$$a(n,\theta) = \left(\frac{\cos\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\sin\theta}{\sigma_y(n)}\right)^2$$

$$b(n,\theta) = -\frac{\sin 2\theta}{\sigma_x^2(n)} + \frac{\sin 2\theta}{\sigma_y^2(n)}$$

$$c(n,\theta) = \left(\frac{\sin\theta}{\sigma_x(n)}\right)^2 + \left(\frac{\cos\theta}{\sigma_y(n)}\right)^2$$

Where the characteristic matrix, $\Omega(n,\theta) = \begin{bmatrix} a(n,\theta) & b(n,\theta) \\ b(n,\theta) & c(n,\theta) \end{bmatrix}$ is positive definite.

Writing out the characteristic matrix we have

$$\Omega(n,\theta) = \begin{bmatrix} \frac{\cos^2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_x}}}} + \frac{\sin^2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_y}}}} & \frac{\sin 2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_y}}}} - \frac{\sin 2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_x}}}} \\ \frac{\sin 2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_y}}}} - \frac{\sin 2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_x}}}} & \frac{\cos^2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_y}}}} + \frac{\sin^2\theta}{\sigma_0^2 e^{-\frac{2\cdot n}{\tau_{\sigma_x}}}} \end{bmatrix}$$

We now introduce the notation $\Delta_x = e^{\frac{2}{\tau_{\sigma_x}}}$ and $\Delta_y = e^{\frac{2}{\tau_{\sigma_y}}}$. Δ_x and Δ_y represent the *squared effective decay rate* of the axial variances of the neighborhood function. Using this notation, the characteristic matrix becomes

$$\Omega(n,\theta) = \frac{1}{\sigma_0^2}\begin{bmatrix} \cos^2\theta\Delta_x^n + \sin^2\theta\Delta_y^n & \sin 2\theta\Delta_y^n - \sin 2\theta\Delta_x^n \\ \sin 2\theta\Delta_y^n - \sin 2\theta\Delta_x^n & \cos^2\theta\Delta_y^n + \sin^2\theta\Delta_x^n \end{bmatrix}$$

We further reduce the characteristic matrix by replacing the trigonometric quantities since they are constant in the variable n. Substituting $cos^2\theta$, $sin^2\theta$ and $\sin 2\theta$ for κ_1, κ_2 and κ_3 respectively, the characteristic matrix becomes

$${}^{\theta}\Omega(n) = \frac{1}{\sigma_0^2}\cdot\begin{bmatrix} \kappa_1\cdot\Delta_x^n + \kappa_2\cdot\Delta_y^n & \kappa_3\cdot\Delta_y^n - \kappa_3\cdot\Delta_x^n \\ \kappa_3\cdot\Delta_y^n - \kappa_3\cdot\Delta_x^n & \kappa_1\cdot\Delta_y^n + \kappa_2\cdot\Delta_x^n \end{bmatrix}$$

Note that $\kappa_1 + \kappa_2 = 1$ and $4(\kappa_1\cdot\kappa_2) = \kappa_3^2$.

Definition The *effective radius* σ_0, of a data-set, is the radius of the hyper-sphere that bounds the convex hull of the data set.

Definition Let θ_C be the slope of the line of least squares through the points a given cluster C. The *angular bias function space* $\mathbb{B}_{\sigma_0}^{\theta_C}$ is the set of neighborhood functions that bias the convergence of the synaptic weights along the line of least squares of cluster C. $\mathbb{B}_{\sigma_0}^{\theta_C} = \{{}^{\theta}\Omega(n) | n \in \mathbb{Z}\}$ where Ω has initial spread σ_0.

We now develop the tools to show that for a data-set of effective radius σ_0 there exists a binary operation over which the matrices in $\mathbb{B}_{\sigma_0}^{\theta_C}$ for fixed θ form a group.

Let us decompose the identity matrix as follows $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ and adopt the notation I_1 for $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and I_2 for $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$. Thus $I = I_1 + I_2$

Define the functions $\Gamma^+(A) = (I_1AI_1 + I_2AI_2)$ and $\Gamma^-(A) = (I_1AI_2 + I_2AI_1)$ over the space of real 2x2 matrices.

It is easy to see that Γ^+ performs the mapping

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} \mapsto \begin{bmatrix} x & 0 \\ 0 & w \end{bmatrix}$$

And that Γ^- performs the mapping

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} \mapsto \begin{bmatrix} 0 & y \\ z & 0 \end{bmatrix}$$

The functions Γ^+ and Γ^- have the effect of isolating the diagonal elements.

Theorem 7.1: The set of neighborhood functions over a data-set of effective radius σ_0 in the angular bias function space for a fixed angle θ forms a group with the binary operation $\circ$, $M_1 \circ M_2 = \sigma_0^2\left[M_1 * M_1 - \frac{3}{4}\left(\Gamma^-(M_1) * \Gamma^-(M_2)\right)\right]$ where $*$ represents matrix multiplication.

See Appendix for Proof

A. Applications of Group Theory

The computational complexity of the SOM algorithm and any of its variants may be reduced by employing the use of the established group structure. This reduction in computational complexity is performed by first computing the generator of the group $\Omega(1)$ and then generating the entire family of neighborhood functions by using simple matrix multiplication.

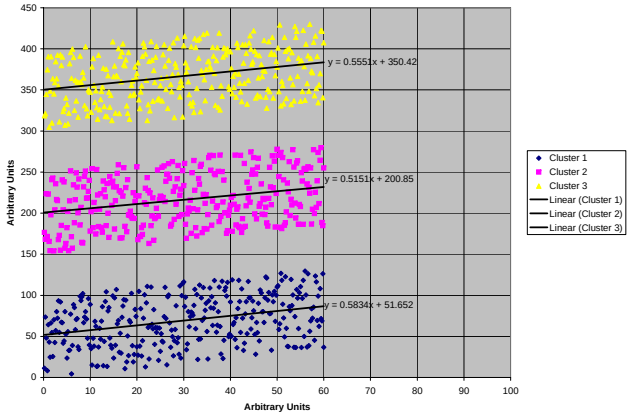


Fig. 14. Artificial Data-Set used to Mirror Linear Regression

VIII. Extensions of the use of Generalized Neighborhood Functions for Fixed-Angle Cluster Orientation

The generalized neighborhood function was tested on an artificial data-set in a context that mirrors linear regression.

A. Data-Set

The functional form of a cluster in the data-set was given by $y = x \cdot tan(\theta) + K_1 + U(-50, 50)$, where $U(a, b)$ is a continuous uniformed distribution on the interval $[a, b]$. The data-set comprised 3 clusters of 300 points given by,

$$y_1 = x_1 \cdot tan(\tfrac{\pi}{6}) + 50 + U(-50, 50)$$
$$y_2 = x_2 \cdot tan(\tfrac{\pi}{6}) + 200 + U(-50, 50)$$
$$y_3 = x_3 \cdot tan(\tfrac{\pi}{6}) + 350 + U(-50, 50)$$

The clustered data is illustrated in the Fig 14.

B. Experiment Setup

We set $\sigma_0 = 400$, $\phi_x = 1000$, $\phi_y = 10$ and $\theta = \frac{\pi}{6}$ in the generalized neighborhood function. The usual bounds of $\eta_0 = 0.1$ and $\tau_2 = \frac{n_{max}}{\ln(10)}$, where n_{max} is an upper-bound on the number of iterations performed in the experiment, were implemented. The results of 5 trials were averaged over varying numbers of iterations. A phenomenal increase in clustering performance was observed. The performance is outlined in Fig 15 and Fig 16.

Iterations	Precision	Recall	F-Score	Accuracy
4000	0.895	0.897	0.896	0.931
5000	0.923	0.924	0.924	0.949
10000	0.927	0.929	0.928	0.952
20000	0.947	0.947	0.947	0.965
30000	0.947	0.948	0.947	0.965
40000	0.953	0.953	0.953	0.969
50000	0.957	0.958	0.957	0.972
60000	0.959	0.960	0.960	0.973
70000	0.964	0.965	0.965	0.976
80000	0.967	0.967	0.967	0.978
90000	0.981	0.981	0.981	0.987
100000	0.982	0.982	0.982	0.988
150000	0.989	0.989	0.989	0.993

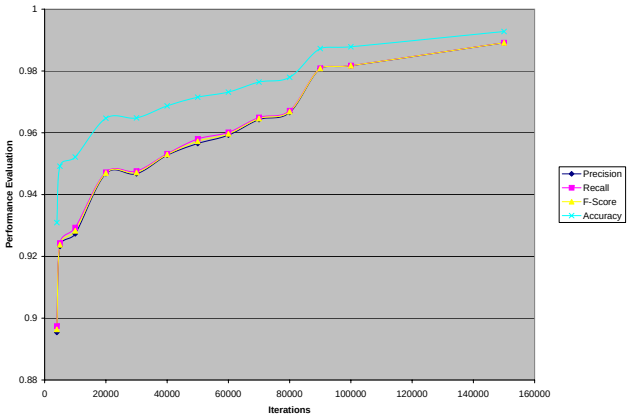


Fig. 15. Performance of Optimized SOM on Artificial Data-Set to Mirror Linear Regression

IX. Extensions of the use of Generalized Neighborhood Functions for Varying-Angle Cluster Orientation

[h] The generalized neighborhood function was tested on an artificial data-set in a context that mirrors non-linear regression.

A. Data-Set

The functional form of a cluster in the data-set was given by $y = K_1 - \sqrt{|300^2 - (x + U(0, 50))^2|} + U(0, 50)$, where $U(a, b)$ is a continuous uniformed distribution on the interval $[a, b]$. The data-set comprised 3 clusters of 300 points given by,

$$y = 300 - \sqrt{|300^2 - (x + U(0, 50))^2|} + U(0, 50)$$
$$y = 500 - \sqrt{|300^2 - (x + U(0, 50))^2|} + U(0, 50)$$
$$y = 700 - \sqrt{|300^2 - (x + U(0, 50))^2|} + U(0, 50)$$

The clustered data is given illustrated in Fig 17.

Iterations	Precision	Recall	F-Score	Accuracy
4000	0.644	0.648	0.646	0.764
5000	0.645	0.653	0.649	0.765
10000	0.657	0.663	0.660	0.773
20000	0.658	0.663	0.660	0.773
30000	0.658	0.664	0.661	0.774
40000	0.664	0.680	0.672	0.778
50000	0.667	0.673	0.670	0.779
60000	0.667	0.674	0.671	0.780
70000	0.668	0.678	0.673	0.780
80000	0.678	0.686	0.682	0.787
90000	0.688	0.689	0.689	0.793
100000	0.695	0.710	0.702	0.799
150000	0.710	0.712	0.711	0.807

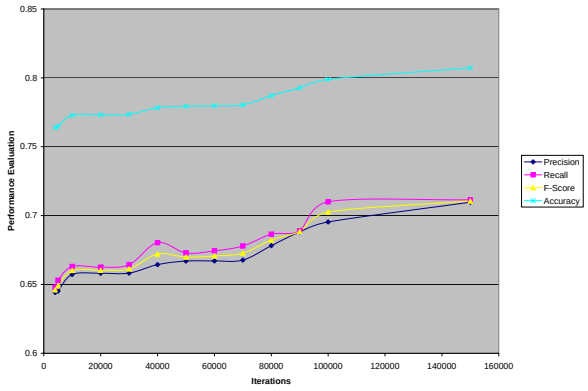


Fig. 16. Performance of Unregulated (ie: $\phi_x = \phi_y$) SOM on Artificial Data-Set to Mirror Linear Regression

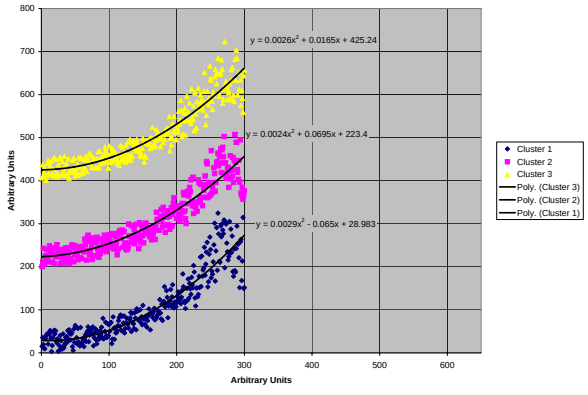


Fig. 17. Artificial Data-Set used to Mirror Non-Linear Regression

B. Experiment Setup

We again set $\sigma_0 = 400$, $\phi_x = 1000$, $\phi_y = 10$. $\theta(x) = \frac{x \cdot \pi}{600}$ in the generalized neighborhood function so that $\theta(x) \to \frac{\pi}{2}$ as $x \to 300$. The usual bounds of $\eta_0 = 0.1$ and $\tau_2 = \frac{n_{max}}{\ln(10)}$, where n_{max} is an upper-bound on the number of iterations performed in the experiment, were implemented. The results of 5 trials were averaged over varying numbers of iterations. A phenomenal increase in clustering performance in non-linear regression was also observed. The performance is outlined in Fig 18 and Fig 19.

Iterations	Precision	Recall	F-Score	Accuracy
4000	0.779	0.792	0.785	0.856
5000	0.778	0.794	0.786	0.856
10000	0.779	0.796	0.787	0.857
20000	0.780	0.795	0.788	0.857
30000	0.780	0.796	0.788	0.857
40000	0.781	0.797	0.789	0.858
50000	0.781	0.797	0.789	0.858
60000	0.782	0.798	0.790	0.859
70000	0.783	0.797	0.790	0.859
80000	0.784	0.800	0.791	0.860
90000	0.784	0.801	0.792	0.860
100000	0.785	0.802	0.793	0.861
150000	0.786	0.802	0.794	0.862

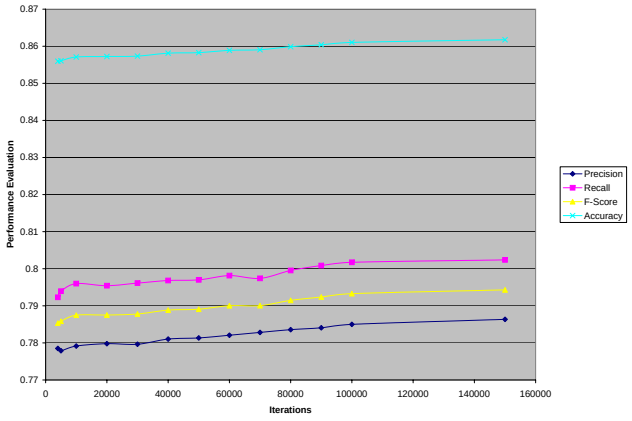


Fig. 18. Performance of Unregulated SOM on Artificial Data-Set to Mirror Non-Linear Regression

Iterations	Precision	Recall	F-Score	Accuracy
4000	0.385	0.639	0.480	0.540
5000	0.385	0.612	0.472	0.545
10000	0.386	0.617	0.475	0.547
20000	0.392	0.656	0.491	0.547
30000	0.403	0.718	0.516	0.552
40000	0.410	0.733	0.526	0.560
50000	0.411	0.737	0.528	0.561
60000	0.412	0.736	0.528	0.563
70000	0.413	0.737	0.529	0.564
80000	0.413	0.736	0.529	0.564
90000	0.415	0.737	0.531	0.567
100000	0.416	0.740	0.533	0.568
150000	0.417	0.741	0.533	0.569

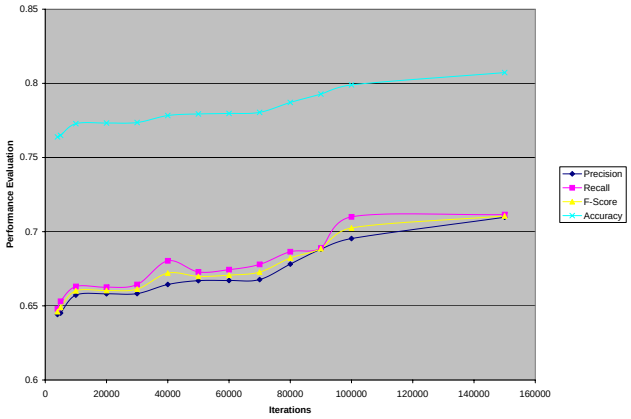


Fig. 19. Performance of Unregulated (ie: $\phi_x = \phi_y$) SOM on Artificial Data-Set to Mirror Non-Linear Regression

The use of the optimized neighborhood function substantially improves the performance of the SOM algorithm in the context of a non-linear regression.

X. Extensions of the use of Generalized Neighborhood Functions for Arbitrary Cluster Orientation

Though improvements may not be as pronounced, the use of generalized neighborhood functions that vary in some may according to overall properties for the data-set may be extended to clustering with arbitrary orientation.

Algorithm 2 describes a generalized mechanism for speeding up clustering when there tends to be little overlap in the convex hulls of the feature spaces.

XI. Conclusion

The Peterson Barney data-set was analyzed and the emergence of clustering based on vowel and speaker-type was observed. It was shown that it is possible to exploit geometrically evident properties of the data-set by implementing an adaptive neighborhood function that utilized a very weak hypothesis regarding the cluster distribution. We proposed a general abstraction in various contexts and obtained superior results; It is possible extend this approach to higher dimensions and more complex geometric hypotheses. The field of asymmetric neighborhood functions hold a wealth of opportunities for the design of heuristics and other dynamic techniques for improving clustering performance through accuracy and convergence improvements.

Algorithm 2 Dynamic Generalized Neighborhood Function

1: Draw the sample $\mathbf{x}$ from the input space with a certain probability; the vector $\mathbf{x}$ will represent the usual activation pattern.
2: Search for the winning neuron $i(\mathbf{x})$ at time step n. This is the neuron closest to the input sample. The winning neuron is obtained by: $i(\mathbf{x}) = argmin\|\mathbf{x}(n) - \mathbf{w}_j\|$, j = 1,2...,$l$
3: Search for the neuron $\mathbf{z}$ closest to $\mathbf{x}$.
4: Use some auxiliary mechanism to perform a linear approximation of the boundary between the cluster corresponding to the winning neuron and the cluster corresponding to the second place neuron
5: Align the neighborhood function such that the major axis is parallel to the line that best separates the feature spaces corresponding to $\mathbf{z}$ and $\mathbf{x}$.
6: Compute $h_{j,i(\mathbf{x})}$ given the value of θ from the alignment.
7: return $h_{j,i(\mathbf{x})}$

References

[1] Ultsch, Alfred (2007). Emergence in Self-Organizing Feature Maps, In Proceedings Workshop on Self-Organizing Maps (WSOM '07). Bielefeld, Germany. ISBN 978-3-00-022473-7.
[2] Haykin, Simon (1999). "9. Self-organizing maps", Neural networks - A comprehensive foundation, 2nd edition, Prentice-Hall. ISBN 0-13-908385-5.
[3] Abercrombie, D. (1967). Elements of General Phonetics. Edinburgh University Press: Edinburgh.
[4] Catford, J. C. (1977). Fundamental problems in phonetics. Bloomington, IN: Indiana University Press. ISBN 0-253-32520-X.
[5] Clark, John; & Yallop, Colin. (1995). An introduction to phonetics and phonology (2nd ed.). Oxford: Blackwell. ISBN 0-631-19452-5.
[6] Klatt, D. 1980, "Software for a Cascade/Parallel Formant Synthesizer" Journal of the Acoustical Society of America, 67:13-33
[7] J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1:281-297
[8] J. A. Hartigan (1975) "Clustering Algorithms". Wiley.
[9] J. A. Hartigan and M. A. Wong (1979) "A K-Means Clustering Algorithm", Applied Statistics, Vol. 28, No. 1, p100-108.
[10] D. Arthur, S. Vassilvitskii (2006): "How Slow is the k-means Method?," Proceedings of the 2006 Symposium on Computational Geometry (SoCG).
[11] An efficient k-means clustering algorithm: Analysis and implementation, T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, IEEE Trans. Pattern Analysis and Machine Intelligence, 24 (2002), 881-892.
[12] H. Zha, C. Ding, M. Gu, X. He and H.D. Simon. "Spectral Relaxation for K-means Clustering", Neural Information Processing Systems vol.14 (NIPS 2001). pp. 1057-1064, Vancouver, Canada. Dec. 2001.

[13] Chris Ding and Xiaofeng He. "K-means Clustering via Principal Component Analysis". Proc. of Int'l Conf. Machine Learning (ICML 2004), pp 225-232. July 2004.
[14] Ball, Martin J.; John H. Esling & B. Craig. Dickson (1995). "The VoQS system for the transcription of voice quality". Journal of the International Phonetic Alphabet 25 (2): 71-80.
[15] Duckworth, M.; G. Allen, M.J. Ball (December 1990). "Extensions to the International Phonetic Alphabet for the transcription of atypical speech". Clinical Linguistics and Phonetics 4 (4): 273-280.
[16] Hill, Kenneth C. (March 1988). "Review of Phonetic symbol guide by G. K. Pullum & W. Ladusaw". Language 64 (1): 143-144. doi:10.2307/414792.
[17] International Phonetic Association (1989). "Report on the 1989 Kiel convention". Journal of the International Phonetic Alphabet 19 (2): 67-80.
[18] International Phonetic Association (1999). Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet. Cambridge: Cambridge University Press. ISBN 0-521-65236-7 (hb); ISBN 0-521-63751-1 (pb).

XII. Appendix

A. Proof of Group Structure over Angular Bias Function Space

Proof:

Identity Element

${}^{\theta}\Omega(0) = \frac{1}{\sigma_0^2} \cdot \begin{bmatrix} \kappa_1+\kappa_2 & \kappa_3-\kappa_3 \\ \kappa_3-\kappa_3 & \kappa_1+\kappa_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma_0^2} & 0 \\ 0 & \frac{1}{\sigma_0^2} \end{bmatrix}$ is the identity element of the group since,

${}^{\theta}\Omega(0) *^{\theta} \Omega(n) =^{\theta} \Omega(n) *^{\theta} \Omega(0) = \frac{1}{\sigma_0^2} {}^{\theta}\Omega(n)$

And

$\Gamma^-({}^{\theta}\Omega(0)) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

So that,

$\Gamma^-({}^{\theta}\Omega(n)) * \Gamma^-({}^{\theta}\Omega(0)) = \Gamma^-({}^{\theta}\Omega(0)) * \Gamma^-({}^{\theta}\Omega(n)) = 0$

Whence, ${}^{\theta}\Omega(n) \circ^{\theta} \Omega(0) =^{\theta} \Omega(0) \circ^{\theta} \Omega(n) =^{\theta} \Omega(n)$

Closure

Given integers n and k, we need to show that ${}^{\theta}\Omega(n) \circ^{\theta} \Omega(k) \in \mathbb{B}^{\theta}_{\sigma_0}$. That is, the operation $\circ$ must preserve the fixed angle θ. Note that for both ${}^{\theta}\Omega(n)$ and ${}^{\theta}\Omega(k)$ the constants κ_1, κ_2 and κ_3 agree since θ is fixed. Let us first consider ${}^{\theta}\Omega(n) *^{\theta} \Omega(k) = \frac{1}{\sigma_0^4} \begin{bmatrix} e_1 & e_2 \\ e_3 & e_4 \end{bmatrix}$, the elements of which we compute term by term for clarity.

$$\begin{aligned} e_1 =& (\kappa_1\Delta_x^n + \kappa_2\Delta_y^n) \cdot (\kappa_1\Delta_x^k + \kappa_2\Delta_y^k) + \\ & (\kappa_3\Delta_y^n - \kappa_3\Delta_x^n) \cdot (\kappa_3\Delta_y^k - \kappa_3\Delta_x^k) \\ =& \kappa_1\kappa_1\Delta_x^{n+k} + \kappa_1\kappa_2\Delta_x^n\Delta_y^k + \\ & \kappa_1\kappa_2\Delta_x^k\Delta_y^n + \kappa_2\kappa_2\Delta_y^{n+k} + \\ & \kappa_3^2\Delta_y^{n+k} - \kappa_3^2\Delta_x^k\Delta_y^n + \\ & \kappa_3^2\Delta_x^{n+k} - \kappa_3^2\Delta_x^n\Delta_y^k \\ =& \kappa_1\Delta_x^{n+k} + \kappa_2\Delta_y^{n+k} + \\ & 3\kappa_1\kappa_2\Delta_y^{n+k} - 3\kappa_1\kappa_2\Delta_x^k\Delta_y^n + \\ & 3\kappa_1\kappa_2\Delta_x^{n+k} - 3\kappa_1\kappa_2\Delta_x^n\Delta_y^k \end{aligned}$$

So that e_1 is given by,

$$\kappa_1\Delta_x^{n+k} + \kappa_2\Delta_y^{n+k} + \tfrac{3\kappa_3^2}{4}(\Delta_y^{n+k} - \Delta_x^k\Delta_y^n + \Delta_x^{n+k} - \Delta_x^n\Delta_y^k)$$

and by the symmetry in the form of Ω, it is easy to see that e_4 is of a similar form in which the κ_1 and κ_2 terms are swapped. e_4 given by,

$$\kappa_1\Delta_y^{n+k} + \kappa_2\Delta_x^{n+k} + \tfrac{3\kappa_3^2}{4}(\Delta_y^{n+k} - \Delta_x^k\Delta_y^n + \Delta_x^{n+k} - \Delta_x^n\Delta_y^k)$$

Let us make the substitution

$\varepsilon = \frac{3\kappa_3^2}{4}(\Delta_y^{n+k} - \Delta_x^k\Delta_y^n + \Delta_x^{n+k} - \Delta_x^n\Delta_y^k)$

So that,

$e_1 = \kappa_1\Delta_x^{n+k} + \kappa_2\Delta_y^{n+k} + \varepsilon$

and

$e_4 = \kappa_1\Delta_y^{n+k} + \kappa_2\Delta_x^{n+k} + \varepsilon$

Additionally we see that,

$$\begin{aligned} e_2 =& (\kappa_1\Delta_x^n + \kappa_2\Delta_y^n) \cdot (\kappa_3\Delta_y^k - \kappa_3\Delta_x^k) + \\ & (\kappa_3\Delta_y^n - \kappa_3\Delta_x^n) \cdot (\kappa_1\Delta_y^k + \kappa_2\Delta_x^k) \\ =& \kappa_3[\kappa_1(\Delta_x^n\Delta_y^k - \Delta_x^{n+k}) + \kappa_2(\Delta_y^{n+k} - \Delta_y^n\Delta_x^k) + \\ & \kappa_1(\Delta_y^{n+k} + \Delta_y^n\Delta_x^k) - \kappa_2(\Delta_x^n\Delta_y^k + \Delta_x^{n+k})] \\ =& \kappa_3[(\kappa_1+\kappa_2) \cdot (\Delta_y^{n+k} - \Delta_x^{n+k})] \\ =& \kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k} \end{aligned}$$

It is also easy to see from the symmetry of Ω that $e_2 = e_3$. So that e_2 and e_3 are both given by,

$\kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k}$

${}^{\theta}\Omega(n) *^{\theta} \Omega(k) = \frac{1}{\sigma_0^4} \begin{bmatrix} \kappa_1\Delta_x^{n+k} + \kappa_2\Delta_y^{n+k} + \varepsilon & \kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k} \\ \kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k} & \kappa_1\Delta_y^{n+k} + \kappa_2\Delta_x^{n+k} + \varepsilon \end{bmatrix}$

A second calculation shows that $\Gamma^-({}^{\theta}\Omega(n)) * \Gamma^-({}^{\theta}\Omega(k))$,

$\Gamma^-({}^{\theta}\Omega(n)) * \Gamma^-({}^{\theta}\Omega(k)) = \frac{1}{\sigma_0^4} \begin{bmatrix} e_5 & e_6 \\ e_7 & e_8 \end{bmatrix}$

It is clear that $e_5 = e_8 = 0$ and $e_6 = e_7$ since,

$\Gamma^-({}^{\theta}\Omega(n)) = \frac{1}{\sigma_0^2} \cdot \begin{bmatrix} 0 & \kappa_3 \cdot \Delta_y^n - \kappa_3 \cdot \Delta_x^n \\ \kappa_3 \cdot \Delta_y^n - \kappa_3 \cdot \Delta_x^n & 0 \end{bmatrix}$

And,

$\Gamma^-({}^{\theta}\Omega(k)) = \frac{1}{\sigma_0^2} \cdot \begin{bmatrix} 0 & \kappa_3 \cdot \Delta_y^k - \kappa_3 \cdot \Delta_x^k \\ \kappa_3 \cdot \Delta_y^k - \kappa_3 \cdot \Delta_x^k & 0 \end{bmatrix}$

So we have,

$$\begin{aligned} e_6 =& (\kappa_3\Delta_y^k - \kappa_3\Delta_x^k) \cdot (\kappa_3\Delta_y^n - \kappa_3\Delta_x^n) \\ =& \Delta_y^{n+k} - \Delta_x^k\Delta_y^n + \Delta_x^{n+k} - \Delta_x^n\Delta_y^k \\ =& \frac{4 \cdot \varepsilon}{3} \end{aligned}$$

Hence, $\Gamma^-({}^{\theta}\Omega(n)) * \Gamma^-({}^{\theta}\Omega(k)) = \frac{1}{\sigma_0^4} \begin{bmatrix} 0 & \frac{4\cdot\varepsilon}{3} \\ \frac{4\cdot\varepsilon}{3} & 0 \end{bmatrix}$

So that $\sigma_0^2 \left[{}^{\theta}\Omega(n) *^{\theta} \Omega(k) - \frac{3}{4}(\Gamma^-({}^{\theta}\Omega(n)) * \Gamma^-({}^{\theta}\Omega(k)))\right]$ gives

$\begin{bmatrix} e_1 & (e_3 - \frac{3e_6}{4}) \\ e_3 & (e_4 - \frac{3e_8}{4}) \end{bmatrix} = \frac{1}{\sigma_0^2} \begin{bmatrix} \kappa_1\Delta_x^{n+k} + \kappa_2\Delta_y^{n+k} & \kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k} \\ \kappa_3\Delta_y^{n+k} - \kappa_3\Delta_x^{n+k} & \kappa_1\Delta_y^{n+k} + \kappa_2\Delta_x^{n+k} \end{bmatrix}$

which is equal to ${}^{\theta}\Omega(n+k) \in \mathbb{B}^{\theta}_{\sigma_0}$.

Inverse Elements

The inverse element of ${}^{\theta}\Omega(n)$ is ${}^{\theta}\Omega(m)$ where n and m are additive inverses. This is true since,

${}^{\theta}\Omega(n) \circ^{\theta} \Omega(-n) =^{\theta} \Omega(n + (-n)) =^{\theta} \Omega(0)$

Associativity
Associativity also follows trivially since,
$[{}^{\theta}\Omega(i) \circ^{\theta} \Omega(j)] \circ^{\theta} \Omega(k) =^{\theta} \Omega(i + j) \circ^{\theta} \Omega(k) =^{\theta} \Omega(i + j + k)$
and,
${}^{\theta}\Omega(i) \circ [{}^{\theta}\Omega(j) \circ^{\theta} \Omega(k)] =^{\theta} \Omega(i) \circ^{\theta} \Omega(j + k) =^{\theta} \Omega(i + j + k)$
Whence,
${}^{\theta}\Omega(i) \circ [{}^{\theta}\Omega(j) \circ^{\theta} \Omega(k)] = [{}^{\theta}\Omega(i) \circ^{\theta} \Omega(j)] \circ^{\theta} \Omega(k)$ ■

Modeling the Allocation of Behavior in Steady-State and Transitioning Concurrent-Schedule, Variable Interval, Reinforcement Learning

Erica J. Newland

***Abstract*—The acquisition of choice behaviors has been associated with a positive/negative/neutral reinforcement system that is well-modeled by neural networks. A multi-layer perceptron trained using back-propagation with a modified desired-output vector is used to model this behavior allocation in both steady-state and transitioning concurrent-schedule, variable-interval, reinforcing learning situations. The simulated results are compared to those in a laboratory setup and demonstrate that although the model there exist many general similarities between the model and the experimental results, the model is not well suited for the operant conditioning experiments simulated.**

***Index Terms*—Concurrent Variable Interval Schedule Learning, Multi-Layer Perceptron, Neural Networks**

I. Introduction[1]

Experiments studying associative learning generally fall into two categories: classical and operant conditioning. Classical conditioning trains involuntary, reflexive behavior that occurs because of a trained association between stimuli and response. Operant conditioning, on the other hand, trains voluntary behavior under the expectation of reinforcement. Reinforcement occurs only if the operant response occurs [1]. A typical operant conditioning experiment involving rats, for example, might feature lever pressing that is reinforced on a probabilistic basis (Expected number of reinforcements per minute = x). In classical conditioning experiments, when reinforcement ceases, the response continues. In operant conditioning, when reinforcement ceases or becomes independent of response, responses become less frequent. This is also known as the *assignment of credit*, and understanding the mechanism by which this occurs is known as the *assignment of credit problem* [2].

General principles of learning have been well established through experimental evidence [3] and a number of mathematical models have been developed to try to explain the learning process [4][5]. The models generally postulate that the better predicted a reinforcer is, the less efficient that reinforcer is at altering behavior. Put otherwise, an unpredicted reinforcer will have a substantial impact on future behavior. The difference between the predicted value of the reinforcer and its actual value is known as the error term. For example, if at time-step *t,* the event *reinforcer* is assigned a value of "1", if the event *no reinforcer* is assigned a value of "0", and if confidence of reinforcement is 0.6, then if there is a reinforcer given at time-step *t,* the error is 0.4. Indeed, this is qualitatively intuitive: if a reinforcer is predicted with 90% certainty and is received, then this leads to little change in the information available to the subject. If a reinforcer predicted with only 10% certainty is received, then this provides significant new information to the subject and suggests the need for an updated set of prediction values. This concept is also the guiding philosophy for the back-propagation algorithm on the multi-layer perceptron. Thus in this paper, we suggest a neural network-based model of learning that employs the back-propagation algorithm, under a particular operant conditioning paradigm.

The molar model also appears to have correlates in neuronal activity, in particular in dopamine neural networks. Dopamine has been identified as a "neural substrate of prediction and reward" and the aptly named dopamine neurons have been linked to the prediction of affect of particular events [6]. Dopamine neurons emit a positive signal in response to an event that is "better" than expected and a negative signal if the event is "worse" than expected. No signal is emitted if an event occurs as expected. This simple reinforcement schedule well matches the mathematical models described above. But although our model follows the general philosophy associated with dopamine neurons and employs a similar error signal, the structure of our network differs from that of neurons in the basal ganglia, where dopamine neurons are located [7]. Thus it is more accurate to consider the model presented here as one that can be used to describe and predict behavior and learning patterns but not necessarily one that accurately models how reinforcement learning is carried out in the brain.

II. MOTIVATION

The model presented is developed as a simulation of a concurrent-schedule laboratory setup used by Banna and

The author is with the Computer Science Department, Yale University, New Haven, CT 06520 USA. (e-mail: erica.newland@yale.edu).

[1] Currently, citations tend to point to works that summarize the relevant piece of information. More specific citations will be added in the final draft.

Newland to obtain an understanding of choice acquisition in rats[7][8]. Under this particular experimental paradigm, subjects are trained in a single session on a number of different learning-schedules. The single-session approach is particularly appealing for our simulation as we do not have to consider the influence of temporal delays between sessions.

In concurrent schedule learning, two choices, for example two levers, are available to the subject. A response on each manipulandum is reinforced with a pre-determined probability. This probability is determined by a variable interval (VI) schedule of reinforcement. On a VI-30 schedule on the left lever, for example, the first response on the left lever to occur an average of 30 seconds after the trial begins is reinforced with a sucrose pellet. It is important to recognize that the actual interval between trial commencement and reinforcement is variable. The expected time for reinforcement may be 30 seconds, but sometimes reinforcement will be activated after 5 seconds have passed; other times it will take a minute for a press to yield reinforcement. The other lever, meanwhile uses its own schedule of reinforcement which may also be VI-30 or may have some a different expected time of reinforcement, for example VI-90. The procedure described is known as the two-key procedure[8][9].

Concurrent schedules can be arranged to be independent or dependent of each other. Under a dependent schedule, a simulation of which is presented in this paper, when one reinforcer becomes active (on), the timer for the other schedule is put on hold until the first reinforcer is actually delivered [8]. Changeover delays are used in concurrent schedule designs to avoid reinforcing a steady alternation between manipulanda. A changeover delay is a forced delay that occurs when the subject switches from one reinforcement schedule to the other. For example, in a two-key arrangement, if the rat has been pressing the right lever and then switches to the left lever, a certain number of seconds pass before the left lever's schedule becomes activated. That is, for that many seconds, there is no chance of reinforcement[8][9].

III. System design

A multi-layer perceptron trained on a modified back-propagation algorithm was used to model the acquisition of choice. Back-propagation is usually associated with the learning-with-a-teacher paradigm, in which a set of expected outcomes is compared to the neural network's output and the network's weights are adjusted according to the calculated error. Put otherwise, in traditional back-propagation learning, the entirety of information about the environment is available to the network [10]. Models of the acquisition of choice in a two-key procedure, however, should reflect the fact that subject only has available minimal information about the environment. Thus most of the modifications made to the back-propagation algorithm in our model are made to reflect the lack of available information about the environment. These adjustments thus involve the design of algorithms for determining, and regularly updating, the desired-output vector. Importantly, we were not interested in a particular set of "final" or convergent weights for our model but instead on the decisions that are made throughout the entirety of the learning process.

We let each neuron's activation function be tanh(x). Let there be n steps in the learning process (n chances for weight updates); let each step i represent a time-step in the learning process. Let $x(i)$ denote an input to network at time step i, let $y(i)$ denote an output of the network at time step i, and let $d(i)$ denote the "desired-output" at time step i. The concept, and re-definition of, desired-output will be discussed below. The training will occur sequentially (the network will be updated after the presentation of each pair $(x(i), d(i))$. The seed input is a 0; each subsequent $x(i) = y(i-1)$.

A. Decisions as Expectations

Conditioning models are often built around a hypothesis that the learning process involves the generation of expectancies of future events [11]. This hypothesis is well summed up by Gallistel in his book *The Organization of Learning:* "When confronted with a choice between alternatives that have different expected rates for the occurrence of some to-be-anticipated outcome, animals, humans, and otherwise, proportion their choices in accord with the relative expected rates."

By setting the value of a simulated reinforcer on lever 1 to 1 and the value of a simulated reinforcer on lever 2 to -1, we make it simple to express the expectations of reinforcement on lever1 and lever2 as probabilities. If, at time step i, the expected reinforcement on lever 1 is 0.6, then this can be interpreted as 60% confidence that there will be reinforcement on lever 1. If, however, the expected reinforcement on lever 1 is 0.2 then this indicates that it is considered more likely for there *not* to be reinforcement on either lever than there to be reinforcement on one of the levers. We work under the reasonable assumption that with no expected reward, a subject would prefer not to exert the energy to press a lever and thus, neither lever should be chosen.

So we design the output of our neural network to be in the form of expected reinforcement rates. An output $y(i) = 1$ is associated with 100% confidence in choosing lever 1, an output $y(i) = -1$ is associated with 100% confidence in choosing lever 2. $|y(i)| \neq 0.5$ means that the simulated subject does not press a lever at time-step i.

B. Memory

An important concept often employed in the construction of our model is that of memory. Consider a vector C of values $C(i)$ that represent knowledge of some characteristic of the state of the environment precisely, and only, at time-step i. But to take $C(i-1)$ as the subject's knowledge of the system at $C(i)$ would be shortsighted: this would suggest that the only information the subject has at time-step i is information about the preceding time step when indeed the subject might have information about many preceding time steps. Yet it would be similarly brash to average the $C(1)$ through $C(i-1)$ to average the values of $C(i)$; this would imply that experiences at the beginning of the session are just as influential as recent

experiences. A considerable mass of experimental data suggests that a feasible model for weighing memories involves the use of the leaky integrator, which was introduced by Bush and Mosteller in 1955[12]. The leaky integrator is a linear operator commonly used in dynamic models of short-term memory in operant conditioning [2]:

$$M_i = wM_{i-1} + (1-w)C(i-1)$$

A small constant *w* places higher weight on new experiences while a large constant places higher weight on old experiences.

C. Motivating the Design and Adaptation of the Desired-Output Vector

Traditional training by back-propagation ultimately hinges on the comparison of the neural network's output at each time step with a desired output for that time step. However, the desired output in the experimental paradigm being modeled is generally probabilistic in nature and at specific times even requires modification in response to the neuron's output. That is to say, the *d(i)* that exists at time *t = 1,* may be very different from the *d(i)* that exists at time *t = i.*

We derive the rules for designing and updating the desired-output vector from a set of motivating examples. In this discussion, the schedules are referred to as VI-x, VI-y, where x and y represent the expected number of seconds until the reinforcer for lever 1 or lever 2, respectively, is activated. Observe that VI-x indicates that the reinforcer for lever 1 is activated, on average, x seconds after the commencement of a trial. That is, the expected number of reinforcers from lever 1 in one minute is *60/x*. Define the length of a session as the number of time steps (for example, seconds) in the learning process. Observe that one activation can last longer than a second; indeed, a single activation on lever j lasts until the subject presses lever j.

First consider time step i at which lever 1's reinforcer is first activated. If the rat presses lever 1 (if $y(i+1) > 0.5$), then a reinforcer is received. If $y(i+1) < 1$, then the simulated subject has received a better reward than was expected. In order to match the general models of reinforcement learning, the decision process that led to the pressing of lever 1 should be strengthened and the rat should be encouraged to press lever 1. On the other hand if the expected reinforcement was 1, then the subject's expectations were precisely met and the reinforcer should have no impact on future behavior. Setting $d(i+1) = 1$ will give us the desired results. In this case, $d(i+1)-1 = 0$, so no changes in the network will occur if $y(i+1)=1$. On the other hand, if $y(i+1)<1$, then $d(i+1)- y(i+1)> 0$, so the weights will be adjusted to encourage future outputs of 1.

Thus we let $d(i) = 1$ correspond to a time-step when lever 1's reinforcement is activated. Similarly, each $d(i) = -1$ will correspond to a time-step when lever 2's reinforcement is activated. Initially, (100/x) percent of the entries in d should be 1 and (100/y) percent of the entries in d should be -1. The rest of the entries in *d* will be filled with 0's, although these 0's will be replaced by different values (see below) during the learning process.

If $y(i) =1$, then no changes in the weights of the perceptron should be made; the subject's expectations were precisely met. Because the reinforcement is the equivalent of "telling" the simulated subject that lever 1's reinforcer was indeed on, we can assign the value of $d(i+1) = 1$ without giving the rat information about the environment that it shouldn't actually have.

Now consider if lever 1's reinforcer has been turned on at time step *i* but the simulated subject does not press lever 1 at time step $i+1$ ($y(i+1) \# 0.5$) .The lever will remain activated and should the rat finally press lever 1 at time-step *j*, we will want $d(j) = 1$. Thus, for each iteration such that $d(i+k) = 1$ and $y(i+k) \# 0.5$, we set $d(i+k+1) = 1$.

Replacing the original value of $d(i+k+1)$ is not, however, a feasible solution. Consider the situation when the initial value of $d(i+k+1)$ is -1. By replacing -1 with 1 we are altering the number of reinforcements provided on VI-y. In a laboratory experiment that runs dependent concurrent schedule, VI-y would simply be delayed until lever 1 were finally pressed. At this point, VI-y would resume. We would encounter the same problem would if the original $d(i+k+1)$ were 1. We would be replacing a distinct activation of VI-x with a different activation. As a result, there would be fewer distinct activations of VI-x. To overcome this problem, instead of *replacing* $d(i+k+1)$, we simply insert a new value in between $d(i+k)$ and the original $d(i+k+1)$, thus bumping $d(i+k+1)$ to the $(i+k+2)$-place in d.

But the insertion of value $d(i+k+1) = 1$ also presents another difficulty. Consider the duration between time-steps $i+1$ and *j*. If the simulated subject does not press lever 1 at time $i+k<j$, it should not receive a reinforcer. If we have set $d(i+k) = 1$, however, then this amounts to telling the subject that it should be pressing lever 1. This is an inaccurate representation of the subject-environment interaction: in the experimental setup that we are modeling, the subject does not know that the reinforcer is "on" at lever 1 until it actually chooses to press lever 1.

First consider the case, at time-step $i+k< j$, when the simulated subject selects to neither press lever 1 nor lever 2 while lever 1 is "on." By pressing no lever, the subject receives no new information about the environment. That is, no weights in the neural network should be updated. The only way to accomplish this is to set $d(i+k) = y(i+k)$. This will result in an error value of 0 and no learning.

Next consider the case when the simulated subject selects to press lever 2 at time $i+k<j$. The subject will not receive reinforcement (because we are employing dependent schedules, lever 2's reinforcer will remain off). Although the subject should not be informed that lever 1's reinforcer is on, it should receive *some* information upon selecting lever 2: it should learn that lever 2's reinforcer is not on. Put otherwise, the decision to select lever 2 should be negatively reinforced. But in what way should the lever be negatively reinforced? Should the subject now be encouraged to press lever 1 or should it be encouraged to not press any lever? In other words, how do we set $d(i+k)$ such that we are giving the subject only the information about the environment that it is allowed to know.

One possible approach, and the one employed in this model, is to model the subject's knowledge about the

reinforcement schedule on lever 1. In order to guarantee that we are negatively reinforcing lever 2, we set $0<d(i+k)<1$. We then determine the value of $d(j+k)$ by determining the expected reinforcement on lever 1 at this point in the algorithm. We accomplish this by using the leaky integrator, which was described earlier; we let $C(i)$ = (# of reinforcements provided to subject from lever 1 at time i).

Finally, we do not want to encourage the subject to continuously switch back and forth between two levers. That is, there must be a cost for changing levers. In the laboratory environment, this is enforced with a changeover delay. If the rat presses lever 1 at time step *i,* and then the next time that he presses a lever he presses lever 2, for the next 2 seconds after the first press on lever 2, he will not be able to receive a pellet, even if the reinforcer is on. This can also be expressed in our *d* by not allowing a value of 1 or -1 for the first two time steps after a "change of levers" occurs. Here, we replace with values of $d(i+1)$ and $d(i+2)$ with 0. More accurately, we insert a new $d(i+1)=0$ and $d(i+2)=0$ in between $d(i)$ and the previously named $d(i+1)$ entry. We choose a replacement value of 0 in order to compensate for the lack of time information conveyed by the back-propagation algorithm. That is, we are investigating if setting d(i+1) and d(i+2) to zero can adjust for the fact that back-propagation algorithm makes the association between switching levers and the lack of reinforcement essentially impossible to make (see [7] and [11] for a discussion of the TD-learning algorithm and its use in behavioral modeling) .

D. Design and Adaptation of the Desired-Output Vector

Below we list the rules, which were justified above, used for creating, and updating, the desired-output vector during training.

1) *If $y(i) < 0.5$ AND $d(i) = 1$,* insert a value $d(i+1) = 1$ *into d. If $y(i) > -0.5$ AND $d(i) = -1$,* insert a value $d(i+1) = d(i)$ into *d.* If a lever's reinforcer is activated but the rat does not press that lever, then the reinforcer remains activated.

2) *If $y(i) > 0.5$ but $d(i)$ is not 1, $d(i) = wM_{i-1}+(1-w)C(i-1)$, where $C(i-1)$ = # of reinforcements on lever2 at time i-1. If $y(i) < -0.5$ but $d(i)$ is not -1, $d(i) = wM_{i-1}+(1-w)C(i-1)$ where $C(i-1)$ = # of reinforcements on leve1 at time i-1.* If the simulated subject presses lever 1 but receives no response and this is not because of a changeover delay, then the subject decides between lever 2 and not pressing a lever based on its memory of lever 2's reinforcement schedule.

3) *If $|y(i)| \neq 0.5$, set $d(i) = y(i)$.* Nothing is learned in this time-step.

4) *If $y(i) > 0.5$, then at the occurrence first k for which $y(i+k) < -0.5$, then $d(i+k) = d(i+k+1) = 0$.* I*f $y(i) < -0.5$, then at the occurrence first k for which $y(i+k) > 0.5$, $d(i+k) = d(i+k+1) = 0$.* This represents a changeover delay.

IV. Results and Analysis

The figures that follow illustrate simulations using a neural net with one input node, two hidden layers of two nodes each, and one output node, a training rate of 0.3, and a momentum of 0.3. Each time-step represents one second.

Banna and Newland conducted sessions that lasted 120 minutes each. During the first 30 minutes, which we will refer to as Stage 1, two VI-30 schedules were run; this yielded an expectation of 2 reinforcers per minute. For the next 90 minutes, which we will refer to as Stage 2, the original schedules were replaced with new schedules. These schedules featured a "rich" lever and a "lean" lever. In each session

$$\frac{E(\text{reinforcements on the poor lever})}{E(\text{reinforcements on the rich lever})} \in \left\{1, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\right\}$$

In both Stage 1 and Stage 2, and in all individual sessions, the expected number of reinforcers per minute was held constant at 2.

Experimentally, steady-state behavior in concurrent scheduling is described by the generalized matching relation [13]:

$$\log \frac{B_1}{B_2} = \log c + a \log \left(\frac{R_1}{R_2} \right)$$

where B1/B2 represents the ratio of responses on lever 1: responses on lever 2 and R1/R2 represents the ratio of reinforcers on lever 1 to reinforcers on lever 2. The steady state ratios were calculated for each of the five VI schedules modeled in the simulation and a least-squares line was fit to the results. On consistent runs of the simulation, the regression resulted in a remarkably good fit. We plotted and fit the data in two distinct ways. In Figure 1(a), each data point represents the cumulative values for a session run under one of the five schedule pairings. In Figure 1(b), each data point represents the cumulative values at each time-step over a 3-stage experiment. That is, Figure 1(b) tracks the learning rate at every single time-step, while Figure 1(a) provides a summary of behavior under each of the different schedule pairings. In Figure 1(a) below, the equation of the line is

$$Y = -0.032518 + 0.55243x$$

The intercept of -0.032518 indicates no bias for either lever and the coefficient of 0.55248 suggests considerable underfitting of the general matching equation. However, this is consistent with results of [8] and [9]. In these works, the average slope value

for a single rat's behavior in one session was 0.63. In Figure 1(b), the slope is 0.6827 and the intercept is -0.0066. These results are also consistent with those found experimentally and suggest that in the model, just as in the laboratory, the general learning – or "success" – trend, measured in terms of the ratio of number of responses to number of reinforcers, is consistent no matter the reinforcement schedule.

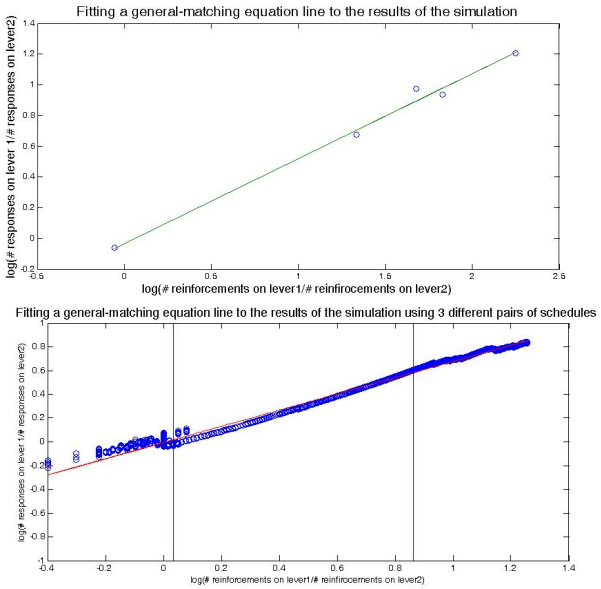


Figure 1(a) and (b). (a), top, shows a fitting of the general matching equation to the results of the simulation where each data point represents the cumulative values for one pair of schedules (ie, the cumulative reinforcements and responses for a 32:1 schedule pairing, 16:1 pairing, 8:1 pairing, 4:1 pairing, and 1:1 pairing). (b), bottom, shows a fitting of the general matching equation to the results of the simulation where each data point represents the cumulative reinforcements and responses at that time step. The vertical lines represent the transition between a 1:1 schedule ratio and a 32:1 ratio, and the transition between a 32:1 ratio and a 16:1 ratio, respectively.

In our model, setting the expected number of reinforcers per minute to two resulted in very few responses on the lean lever and often no reinforcement on the lean lever in a simulated 30 minute block of a session. This suggests one of two conclusions about our model. Either the rich lever is favored more heavily than in the experimental setup or the percent of time spent not responding on a lever is higher in our simulation than in the laboratory. The consistency of our findings with the general matching equation, as well as the results of [8] and [9], suggest the latter.

In Figure 2(a) and (b), we show box plots for the number of responses per simulated minute under a 32:1 paradigm with an average of 2 reinforcers per minute. Figure (c) shows the results from [8] and [9] for laboratory trials under the same set of reinforcement
conditions. We observe that for stage 2, in particular, the difference in lever presses between the simulation and the laboratory results are extreme. The number of lever presses per

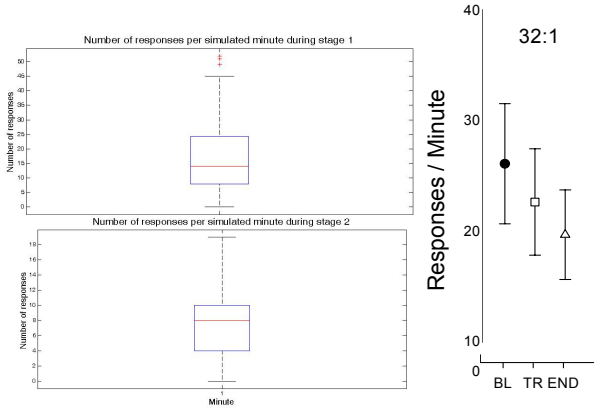


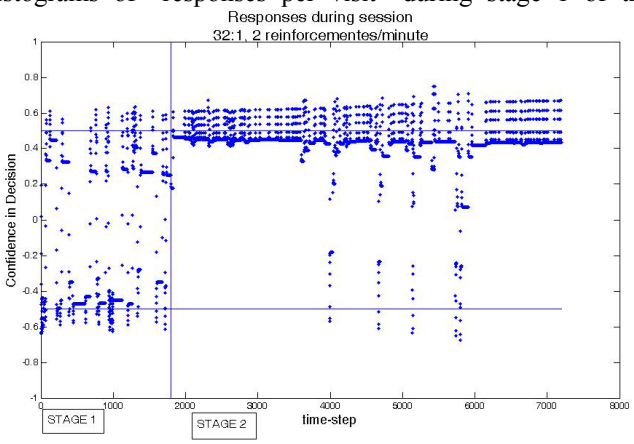
Figure 2. 2(a) and 2(b) (Left top and left bottom) show number of lever presses/minute under a 32:1 paradigm with average of 2 reinforcers per minute. 2(c) (right), shows the results from [8]. BL indicates stage 1 (baseline), TR indicates transition between stage 1 and stage 2. END indicates the end of stage 2.

minute in the 75h quartile of the simulation is less than 10, while in the laboratory results, the number of lever presses per minute does not fall below ten.

A better understanding of the response pattern can be acquired by studying Figure 3, which shows the pattern of responses throughout an entire session. The horizontal lines indicate the lever-press cutoffs of 0.5 and -0.5. The points on the extremes of each of these horizontal lines indicate lever presses. As is expected from a back-propagation algorithm, output values change gradually: from a response on lever 1, through no lever press but favoring lever 1 and no lever press but favoring lever 2, to a response on lever 2.

Figure 3 Simulation Responses

We also analyzed the more specific behavior of the neural network within each steady state. Figure 4(a) and 4(b) are histograms of "responses per visit" during stage 1 of the



simulation. A *visit* on lever 1, for example, ends when lever 2 is pressed for the first time following the pressing of lever 1. This first choice of lever 2 represents the beginning of a visit on lever 2. A visit does not end if there is no lever press during a time step; a visit on one lever is only ended by an actual press on the other lever. Figures 4(c) and 4(d) are histograms of responses per visit during the corresponding laboratory experiments (also equivalent schedules) by [8][9]. [2]

The histograms in Figure 4 suggest that, at least during stage 1, the nature of the visits to lever 1 and 2 are similar in the

[2] Permission granted from the authors for reprint.

simulation and the laboratory results. Although the data for the simulations is sparse, we do not observe the (aberrant) high preference for 1-press visits that the simulation produces under other conditions (see Figure 7(a) and (b)). The range of durations of the visits and the distribution of durations are comparable in the two models, although the absolute number of visits is much higher in the laboratory results. The histograms for stage 2, Figure 5, on the other hand, show a different result.

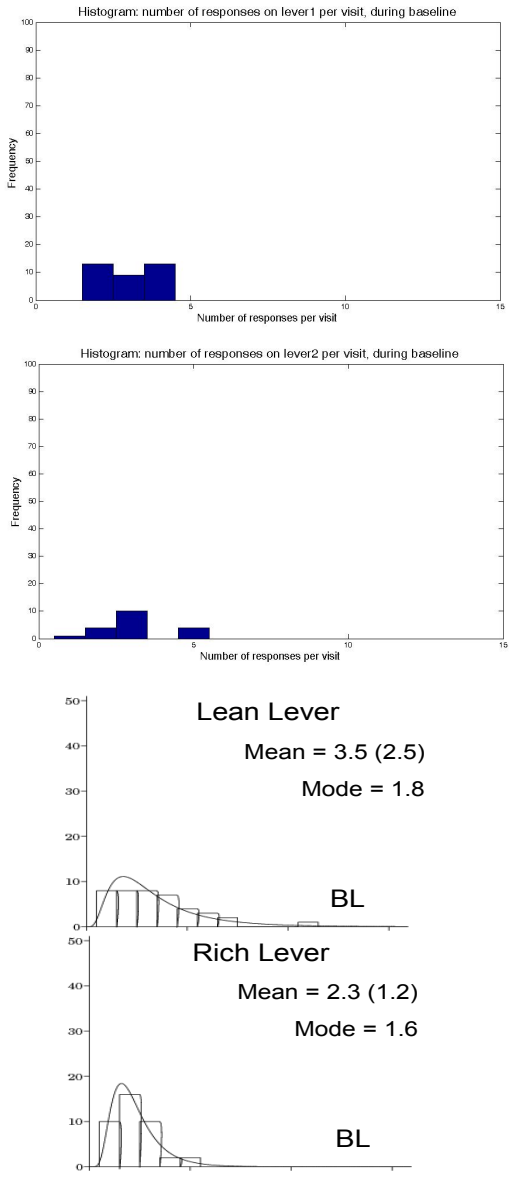


Figure 4, (a), (b), (c), and (d). Histograms of stage 1 (baseline) responses on lever 1 and lever 2 when both are operating under a VI-60 schedule. The data in Figures (c) and (d) are fit with Gaussian curves. Figures (c) and (d) have been provided by [8].

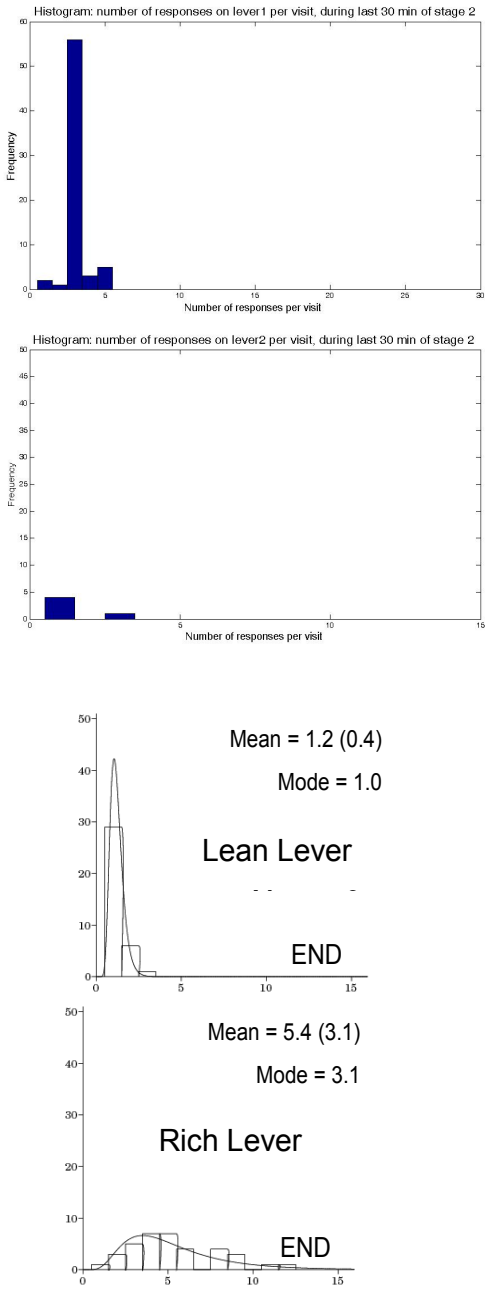


Figure 5, (a), (b), (c), and (d). Histograms of stage2 responses on lever 1 and lever 2 when operating on a 32:1 rich to lean reinforcement ratio with an average of 2 reinforcements activated per minute. The data in Figures (c) and (d) are fit with Gaussian curves. Figures (c) and (d) have been provided by [8].

The peak at a visit of duration equal to 3 is unexplainable. Even when the memory coefficient of 0.3 was varied, when the

changeover delay of 2 seconds was artificially increased within reasonable amounts (+15), or when the network parameters were changed, there remained a peak on the rich lever for visits of duration either 3, 4, 5, or 6. The histogram of the simulation on lever 2 during stage 2 (Figure 5b), suggests that if there were more data points, perhaps the simulation and the laboratory experiments would yield similar results.

It appeared possible that the small number of data points were skewing the data, so we modeled a situation in which 12 reinforcements were expected per minute. We simulated a number of different rich to lean ratios; the result of an 8:1 simulation are shown in Figure 6. We found a high number of visits of duration equal to 1, even when the changeover delay was artificially increased within reasonable bounds. Stage 2, however, was better modeled by this set of conditions, although we again observe the visit duration peak of 3 on the rich lever.

In Figure 6, below, we show the number of changeovers/simulated minute on two different ratio conditions. Although it is difficult to determine from Figure 6 if the change in ratio lead to a transition period in number of changeovers/minute (or if the change was immediate) on the 8:1 condition, it appears that there was a transition period, in the 32:1 condition.

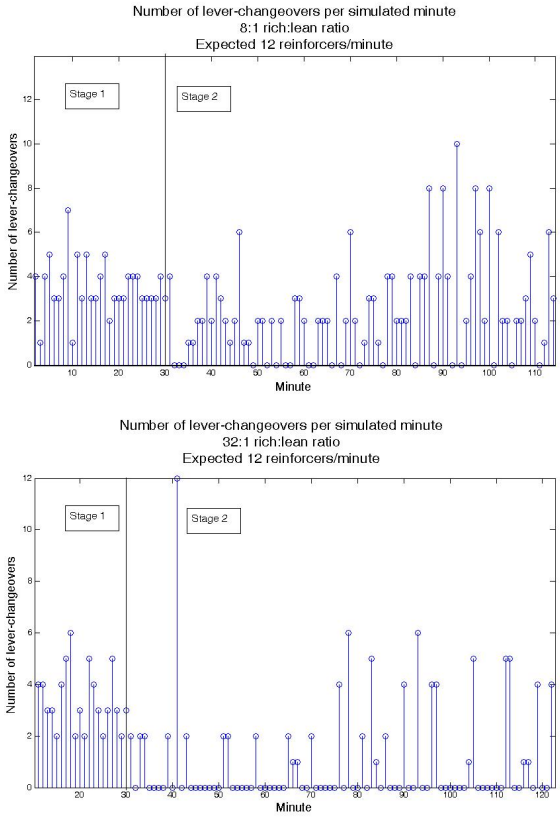


Figure 6 (a)-(b). Changeovers per minute.

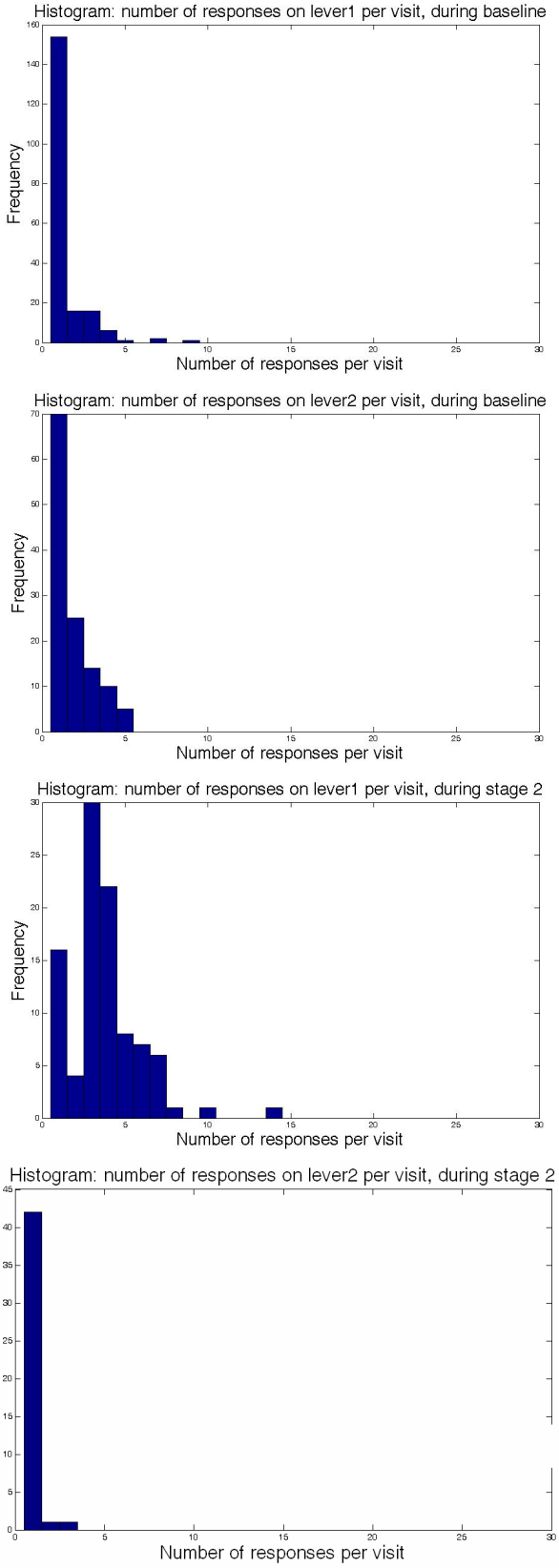


Figure 7(a) –(d). Reinforcement schedules with expected number of reinforcers per minute set to 12. Figures (a) and (b) (top two) are on equivalent schedules. Figure (c) and (d) represent the rich and poor levers, respectively, in an 8:1 reinforcement condition.

Although the histograms and matching equation allow us to compare our model to laboratory results when the subject is learning under a steady paradigms (stage 1, or stage 2), they do not give us information about behavior during the transition between the stages. To determine how our model performs during transitions, we use the following equation, suggested by [14].

$$Log(\frac{B_1}{B_2}) = \frac{P_\infty}{1+e^{k(R_{half}-X)}}$$

Where B_1 represents the responses on lever 1 and B_2 represents the responses on lever 2. The four parameters are fitted using a non-linear least-squares regression. P_{inf} represents the upper asymptote. *k* represents the slope of the straight portion of the S-curve, and R_{half} represents the number of reinforcers cumulatively delivered when the transition is half way complete. In Figures 8 (a) (the simulation) and (b) (the laboratory data), each data point has been computed at the end of a pair of visits, one on lever 1 and one on lever 2. The log of the ratio of responses on lever 1 to responses on lever 2 during that pair of visits represents the dependent variable. The data has been plotted against cumulative reinforcements, which is measured as the number of reinforcements administered thus far in a session at the end of each pair of visits. After being plotted, the data in both figures was smoothed using a LOWESS algorithm and then a non-linear least-squares regression was used to fit the data. In both figures, x = 0 represents the beginning of the transition.

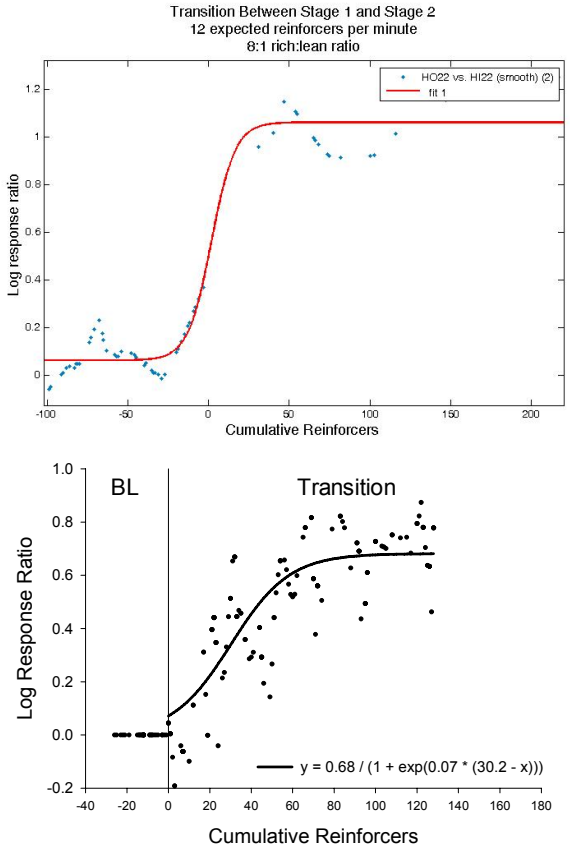


Figure 8 (a), (b): Stage transitions. Figure (b) is on a 32:1 ratio with 2 expected reinforcers/min. (b) provided by [8]

The parameters fit for our model gave us a P_{inf} of 0.9978, a k of 1.903, and an r of 0.1363, all with 95% confidence bounds. The reported R-square was 0.9652.

The ability to fit this model suggests that our network might indeed model behavior transitions. However, the results of many simulations were difficult to fit and simulations run with an expected reinforcer of 2 per minute or with a 32:1 rich:lean lever ratio were not well fit by the transition equation.

V. Two Multi-Layer Perceptrons

A 2-multi-layer perceptron system was also implemented. In this design, each multi-layer perceptron corresponds to one of the levers. The output of each perceptron corresponds to the expectancy of a reinforcer on the lever it represents. The perceptron whose output has the largest absolute value represents the lever chosen by the simulated subject. If no expectancy is greater than 0.5 than no lever is chosen. The rules implemented are very similar to those described earlier for the single multi-layer perceptron system. Only the "winning" perceptron, if there is a perceptron with output greater than 0.5, is updated at each time-step. The results were not promising. Figure 9(a) and 9(b) show the responses at each time step for two consecutive simulations under identical conditions. Clearly the results are extremely variable and do not seem to reflect the conditions nor do they resemble experimental results. This design was not pursued further.

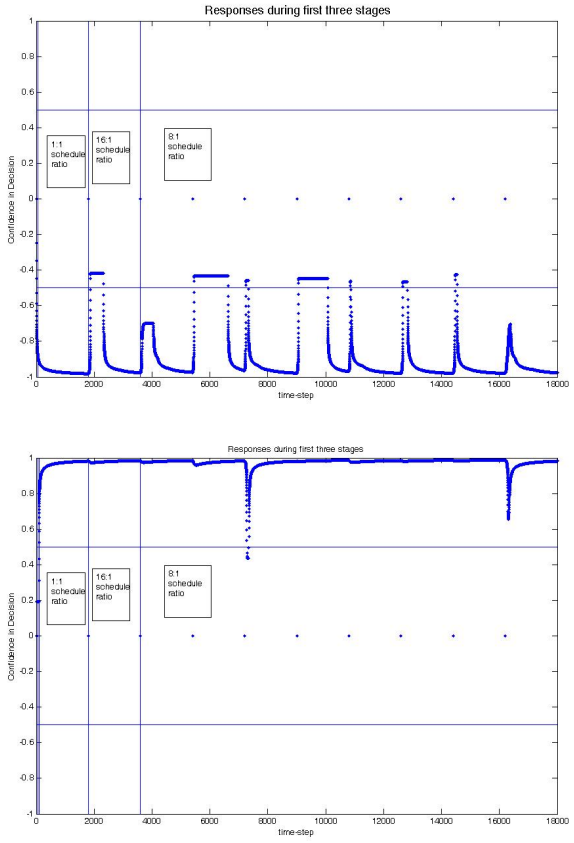


Figure 9(a) and (b). Responses for two consecutive simulations using two multi-layer perceptrons and a winner-takes-all rule. Two reinforcements per minute were expected.

VI. Conclusion

As discussed above, the consistent applicability of the generalized matching equation to the simulation data lends credence to the use of a modified back-propagation algorithm on a single multi-layer perceptron to produce molar models of behavior. However, the simulated model did not well-describe the more molecular behavior of subjects within each state. For example, while the response: reinforcement ratios reported by the model matched experimental results, the pattern in which responses occurred on the two levers differed from those reported experimentally. Moreover, they differed in different ways depending on the conditions tested.

Considerable efforts were devoted to manually changing parameters and an improved algorithm would automatically test these different parameter values in order to obtain optimal results. Generally, the momentum term and learning rate parameter performed best in the range of 0.3-0.5.A large momentum term, for example, was found to produce stays as long as 1,000 seconds on the rich lever under some conditions. Finally, we suggest that a more sophisticated model of the operant conditioning described here could be provided by a different type of network. Suri and Shultz discuss a temporal-delay (TD) algorithm that was used to model dopamine-neuron response in spatial tasks that might be of interest to readers [7]. A TD network would contain more temporal information and thus might better model the learning process.

References

[1] B.F. Skinner, *The Behavior of Organism: An Experimental Analysis,* Prentice Hall, Englewood Cliffs, New Jersey, 1938.

[2] Dragoi, V., & Staddon, J.E.R. (1999). *The Dynamics of Operant Conditioning*. Psychological Review, 106, 20-61. 17 17. Dugatkin, L. A. (1997).

[3] Dickinson A. (1980) *Contemporary Animal Learning Theory*. Cambridge University Press, Cambridge.

[4] Barto A. G., Sutton R. S. and Watkins C. J. C. H. (1990) Learning and sequential decision making. In *Learning and Computational Neuroscience: Foundations of Adaptive Networks* (eds Gabriel M. and Moore J.), pp. 539–602. MIT, Cambridge.

[5] Bauer R. H. and Fuster J. M. (1976) Delayed-matching and delayed-response deficit from cooling dorsolateral prefrontal cortex in monkeys. *J. comp. Physiol. Psychol.* 90**,** 293–302.

[6] A. Pérez-Uribe: Using a Time-Delay Actor-Critic Neural Architecture with Dopamine-Like Reinforcement Signal for Learning in Autonomous Robots. *Emergent Neural Computational Architectures Based on Neuroscience 2001*: 522-533

[7] Summarized in R.E. Suri and W. Schultz. A Neural Network Model With Dopamine-Like Reinforcement Signal That Learns a Spatial Delayed Response Task. Neuroscience, 91(3):871--890, 1999.

[8] Banna, KM. PhD Dissertation, Auburn University, 2007.

[9] Banna, K.M and Newland, M.C. *The Acquisition of Choice,* Accepted for Publication.

[10] Haykin Simon, Neural Networks, A Comprehensive foundation, Prentice Hall, Inc., pp. 351-357, 1999.

[11] Gallistel, C. R. (1990). *The organization of learning*. Cambridge,
MA: MIT Press

[12] Bush, R. R., & Mosteller, F. (1955). Stochastic models for learning. New York: Wiley.

[13] Baum, W. (1974). On two types of deviation from the matching law: Bias and undermatching. *Journal of the Experimental Analysis of Behavior, 22(1)*, pp. 231-242.

[14] Newland, Yezhou, Lögdberg, & Berlin, 1994

[15] Skeleton of code used was obtained from http://www.csee.umbc.edu/~dpatte3/nn/res/backprop.m

A Game Theoretic and Genetic Algorithmic Approach to Modeling the Emergence of Mind in Early Child Development

Jimin Nam

Abstract— **We use a genetic algorithm to study infant and young child development in a game theoretic environment modeling the interaction with either parent or caregiver. By changing the child's interaction time with parental figures, which have several quality levels the development of the child's mind is investigated. This model provides children interaction with several childcare conditions. As increasing exposed time with semi-optimal partner and decreasing quality of a guardian, child's development is delayed**

Index Terms— **Game Theory, Genetic algorithm**

I. Introduction

TO meet the demand of social evolution, many infants and young children are cared at a nursery school for part of the day. Accordingly, child development is effected not only by the quality of their parent but also by the time spent with the parent and at the nursery. Observing child reactions generated by study several childcare environments, we can study child awareness development as it is informed by different levels of care. Awareness is effected by restricted "mentalizing"or "non-mentalizing[2]" interaction. A mentalizing response would be one in which the parent responds to the child in 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 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. However, not all children are exposed to parents only.

We will model the interaction of a normal child with a fully developed parent and contrast this with relations formed by a child with some imperfect parent. Such a model could be represent situations where a child stay in a nursery school and interact with a caregiver, elder friend or peer. This lessen interaction quality compared to that supplied by perfect parent. Interactions with such semi-developed adults should delay child's development[1]. Further, we will vary the amount of daily nursery school time as since world some children spend more time in nursery than others. Moreover, by adding noisy input, we will model an autistic child's development.

Jimin.nam@yale.edu

To model this interaction with parent and child development , a game theoretic approach provided by Miranker and Mayes[2]. This uses a variant of the game Prisoner's Dilemma in a recurrent form to model the interaction between caregiver and child. A genetic algorithm is utilized to model the development of the child's emergence of mind a notation of fitness from the game play derived from increasingly good memory[3].

II. The Model

A. Game Theoretic Environment

To model this interaction with parent and development of child, Prisoner's Dilemma will be used. Each of parent and child can choose a "mentalizing" play or "non-mentalizing play." The parent can ignore or attend the child. On the other hand, the child can use intuition or use his mind. The child wants to get Attention from the parent, and also wants to use his Intuition instead of build his Mind. This reflects 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

Payoff table		
Child\ Parent	Attend	Ignore
Mind	R, R	S, T
Intuition	T, S	P, P

This model becomes a Prisoner's Dilemma when T>R>P>S. The value each participant receives in each iteration of the game hinges on the interaction of their choices with the action selected by their opponent. In this case, the best results for an individual are the result of successfully tricking the opponent into doing more work while engaging in less. The opponent then receives the lowest possible return value. In the real world, such actions result in a loss of trust. A rational agent would learn from such experience and when confronted with that same type of situation would be more apt to anticipate another

deception. The strictly logical course would be to minimize losses by engaging in the less effortful activity. This kind of activity would naturally degrade into the (P,P) situation. Cooperating to achieve the (R,R) situation would benefit both parties more in the long term, but is much more difficult to achieve because of the increased risk associated with that position.

B Mind Model

The parent's strategy is fixed in order focus to isolate child's response. The optimal parent will be play a "tit-for-tat" strategy. This parent plays attend if the child uses his mind while she plays ignored if the child use his intuition. The beginning of the iteration child's memory is empty. After each round of play, the child received payoff. Depending on determines whether up date his memory.

Child's memory holds a list of sequence of plays. To visualize the sequence, we assigned the values T=2, R=1, P=-1 and S=-2. This creates a zero sum game.

In this experiment, child's fitness level is set to 0. After each game, the payoff which child receives is added to his fitness level. The fitness level is an indication of how well he has anticipated the moves of his adversary. A high fitness value results from typically high payoff values returned after participating in the Prisoner's Dilemma and indicates a better response to the strategy being used by his opponent.

After each round of plays, a decision is made to either incorporate or discard the knowledge gained from these with some probability that directly relates to the payoff value that has just been achieved. . Sequences associated with very favorable or very unfavorable payoffs are assigned a lower probability of being discarded. This is analogous to real world situations in which very happy or traumatic events are encoded with greater intensity than those experiences which are not associated with a strong emotion. The probability of a sequence associated with a payoff value T and S are encoded as 0.9 while sequences associated with a value of R or P were encoded with a probability of 0.67. These specific values were chosen arbitrarily.

Memory is limited, so we limit the sequence length that child is able to recollect to 4steps. After each round, the memory is updated by removing the first value (the oldest value), shifting others one step forward, and storing the most recent payoff value is at the end of the memory sequence. When the child is faced with a similar situation in the future, a direct comparison of these two weights can be used to determine which of the two options has more reliably created a better payoff in the past. The memory is used to determine the frequency with which the child will choose the option with a higher weight. In this experiment, the probability, z, that a child would choose the option with a lower weight was

$z= 1/(x-y+2)$

x is the value of the larger weight and y is the value of the smaller. A choice between the two possible plays is then made.

C Exposure to Nursing School

We model the exposure to the child to different levels of adviser by assuming that while child is at nursery school, he would spend time with caregiver, elder child or peer.

The base case is taken for a child staying with an optimal parent whose strategy is "tit-for-tat". The second child will stay at nursery school for 30% of the time for each weekday. We give a discount to caregiver's quality as 70%, 50%, 30%, 0% optimal (totally random). Our model takes it that the child will stay with parent during the weekend as mentioned above.

The actions of the suboptimal parent will be determined by the value of a randomly generated variable, which when greater than some threshold will choose to play randomly rather than use tit-for-tat.

Each game will be composed of 100 rounds, The goal will be to determine and compare the effects of exposure to suboptimal parents of varying degrees of degradation on the child's development and the child's ability to recover when subsequently exposed to optimal parenting.

III. EXPERIMENT

A Overview

Initially, the optimal model was run as a base case. This figure1 shows the case that child stays with optimal parent all day long. It could represent that the child is not going nursery school or that the nursery school caregiver is optimal.

Each time I executed the program, the result values are little bit different each other. This difference of values are generated from memory limit and the beginning part that child didn't build up his memory. From the difference, I measured the mean values of end fitness and standard deviations. For this optimal case the mean end fitness 95 and stand dev 3.7

B Increasing exposure time

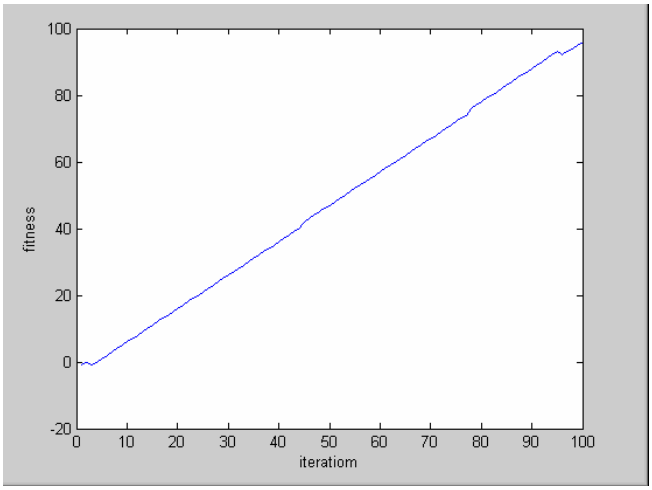


Fig. 1. The base case of the child fitness

In each of the following figures the colors donate the percentage of nursing school time.
I used loop function to divide time as in real world child spend some of his daytime at nursing school then come home and stay with parent whose optimal rate fixed as perfect. If the time is increased the fitness level is going down and the standard deviation value grows up. I set from 90th to 100th iteration as recovering period which means in that part of iteration child interact with perfect caregiver. From the graph the dropping point represent that child in nursing school. Because child's memory is only 4 step long at the beginning, the amount of time spend with semi optimal parent doesn't look like disadvantage. But at the end of graph the fitness end is quite lower than less amount of time as we can see mean end fitness from this table. For the each case the standard deviation is 6.8, 7.3, 7.8, 8.3.
Comparing blue line and ocean blue, even the care giver's semi optimal rate is fixed at 70%. If the child spent 20% of his time the fitness end at 87 and if child spend 50% of his time his fitness end at 52.
When I decreased caregiver's quality from 70% to 50% the mean end of fitness are dropped and standard deviation are

Percentage of time	Mean end Fitness	Standard deviation
20%	87	6.8
30%	76	7.3
40%	62	7.8
50%	52	8.3

Table2. 70% optimal caregiver

increased as table 3.

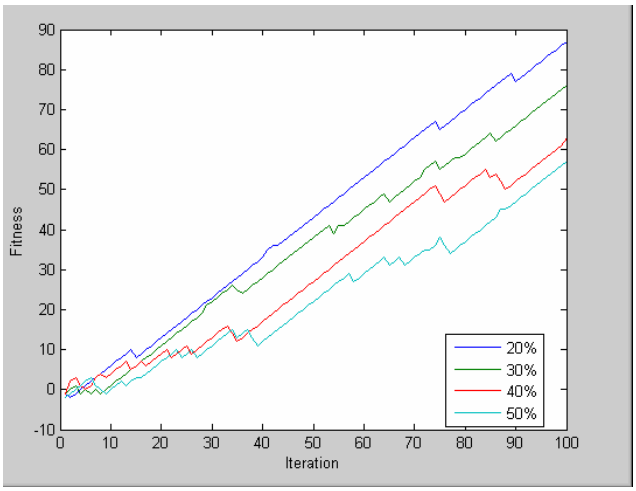


Fig. 2. The child fitness with 70% optimal caregiver

When the caregiver's quality decreased to 30% the mean end of fitness are dropped and standard deviation are increased as table 4. The interesting point is that
By reducing caregiver's optimal level as 0, we can represent the case that child spend his time with peer. The mean end fitness gaps between each colors are relatively big, because the

Percentage of time	Mean end Fitness	Standard deviation
20%	85	11.4
30%	68	12.1
40%	56	12.3
50%	51	12.7

Table3. 50% optimal caregiver

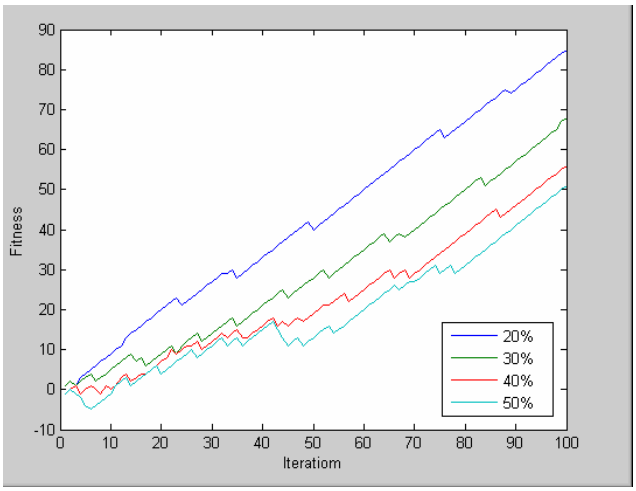


Fig. 3. The child fitness with 50% optimal caregiver

parent's optimal rate is 100% and the caregiver's optimal rate is 0. The mean end fitness are quite low and the standard deviations are relatively big.

Percentage of time	Mean end Fitness	Standard deviation
20%	82	16.8
30%	64	17.2
40%	52	17.9
50%	44	18.7

Table3. 30% optimal caregiver

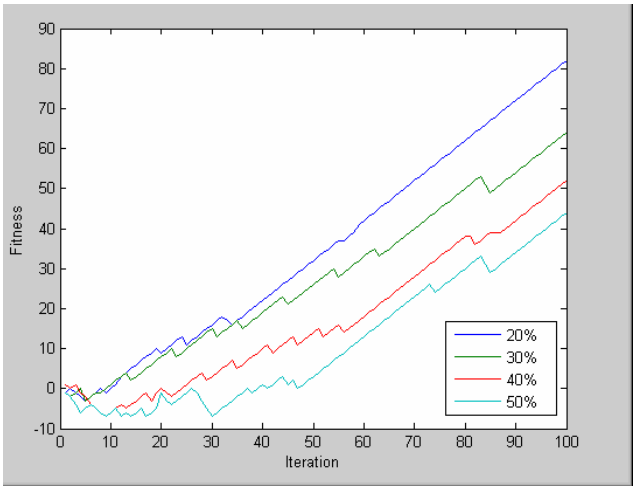


Fig. 4. The child fitness with 30% optimal caregiver

C *Autism child*

Percentage of time	Mean end Fitness	Standard deviation
20%	72	28.2
30%	53	28.9
40%	46	29.8
50%	30	30.3

Table3. with peer

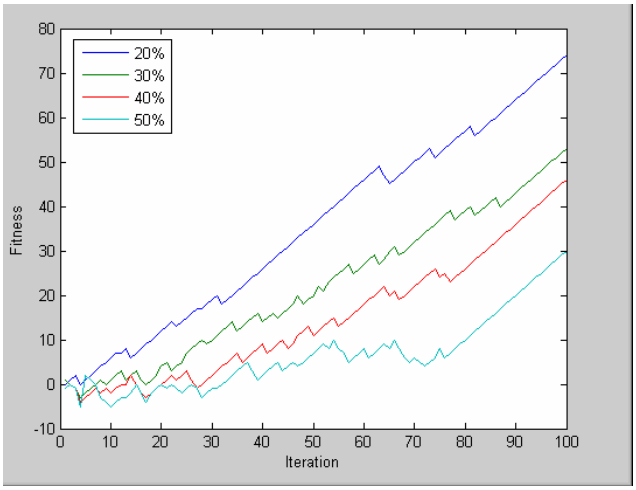


Fig. 5. The child fitness with peer

For autism child case, at first I give 30% random value to input for child and fixed perfect parent. The mean end of fitness was 88 and the standard deviation is 13.8. Same child but the case that spend 20% time with 70% optimal parent, the fitness was about 67 and the standard deviation increase to 19.8%.
The 50% noised child case, all iteration with optimal parent the Fitness was 21 and the standard deviation 49 it just mean random so the result is un measurable.
For the last of cases the standard deviation is too big, the result is doubtful.

IV. CONCLUSION

When we reduced the caregiver's optimal percentages the child's development goes down as we expected. From the result of experiment we recognize that even though child spent part of his time with imperfect care giver. If the percentage of time amount is not big child can get over it The more time child interact with semi-optimal, the longer time to build up memory to child So we can conclude that the quantity time that child spend with optimal parent is as important as the optimal rate of the parent
Autism child case, the experiment of is not clear because the standard deviations are very big. We still can see that the more time with optimal parent can generate the better fitness level

Future work that build some boundary to reduce the standard deviation for autistic case, and a better understanding of the autistic child's problems where biasing could yield better solutions on finding a reliable output.

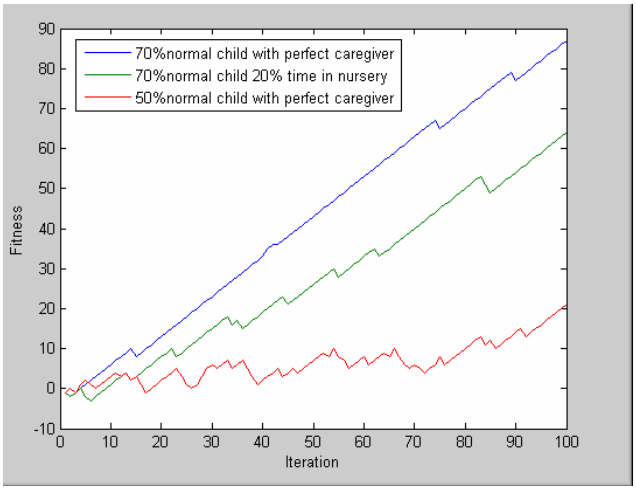


Fig. 6. The child fitness with peer

V. REFERENCES

[1] J. Logan, "A Game Theoretic and Genetic Algorithmic Approach to Modeling the Emergence of Mind in Early Child Development with Exposure to Semi-Developed Parents".
[2] L. Mayes, W. Miranker, "A Game Theoretic and Genetic Algorithmic Approach to Modeling the Emergence of Mind in Early Child Development,"
[3] Laura J. Cehring "Modeling Child Development using a Game Theoretic Approach and Genetic Algorithm'

Stock Market Index Prediction using Neural Network

Kun LI

Abstract— **Researchers have known for some time that nonlinearity exists in the financial markets and that neural networks can be used to forecast market returns. The following research utilizes neural network models for estimation, and results are examined for their ability to provide an effective forecast of future values. A benchmark linear regression model is also employed to show that the neural network models generate higher accuracy in forecasting ability.**

Index Terms— **time series, neural network, linear regression, self-organizing map, recurrent neural network.**

I. Introduction

IMPORTANT changes have taken place over the last two decades within the financial markets, including the use of powerful communication and trading platforms that have increased the number of investors entering the markets. Traditional capital market theory has also changed, and methods of financial analysis have improved. Stock-return forecasting has attracted the attention of researchers for many years and typically involves an assumption that fundamental information publicly available in the past has some predictive relationships to future stock returns or indices. The samples of such information include economic variables, exchange rates, industry- and sector-specific information, and individual corporate financial statements. This is opposed to the general ideal of the efficient market hypothesis which states that all available information affecting the current stock value is constituted by the market before the general public can make trades based on it. Therefore, it is impossible to forecast future returns since they already reflect all information currently known about the stocks.

This is still an empirical issue since there is contradictory evidence that markets are not fully efficient, and that it is possible to predict the future returns with results that are better than random by means of publicly available information such as time-series data on financial and economic variables. These studies identify that variables such as interest rates, monetary-growth rates, changes in industrial production, and inflation rates are statistically important for predicting a portion of the stock returns. However, most of the studies just mentioned that attempt to capture the relationship between the available information and the stock returns rely on simple linear regression assumptions, even though there is no evidence that the relationship between the stock returns and the financial and economic variables is linear. Since there exists significant residual variance of the actual stock return from the prediction of the regression equation, it is possible that nonlinear models could be used to explain this residual variance and produce more reliable predictions of the stock price movements.

Neural networks are a nonlinear modeling technique that may overcome these problems. Neural networks offer a novel technique that does not require a prespecification during the modeling process since they independently learn the relationship inherent in the variables. This is especially useful in security investment and other financial areas where much is assumed and little is known about the nature of the processes determining asset prices. Neural networks also offer the flexibility of various architecture types and learning algorithms. Two neural network approaches that can be used for classification and level estimation will also be briefly reviewed, both multilayer feed-forward neural networks and recurrent neural network.

The resulting data selection and model development, empirical results, and discussion and conclusion will then be presented. Data sources and descriptions are given in the Appendix.

II. Stock Market Index

A stock market index is a listing of stock and a statistic reflecting the composite value of its components. There are many major market indices in finance world, each of the them tracks the performance of a specific group of stocks considered to represent a particular market or sector of the stock market or the economy.

The Dow Jones Industrial Average (DJIA) is an index of 30 of the largest and most widely held public companies in the United States. The Index includes substantial industrial companies with a history of successful growth and wide investor interest. The Index includes a wide range of companies‡from financial services companies, to computer companies, to retail companies.

A typical plot of daily data for Dow Jones Industrial Average looks like Figure 1. Like most data (such as asset prices, exchange rates, GDP, inflation and other macroeconomic indicators) in economics and finance, a stock market index comes in the form of time series exhibiting very high noise, and significant non-stationarity and nonlinearity. Every aspect will be studied later in this report and techniques will be employed to overcome those difficulties.

Kun LI is a student.

Fig. 1. Time series of the Dow Jones Industrial Average (DJIA)

III. The Efficient Market Hypothesis

Professor Eugene Fama at the University of Chicago Graduate School of Business developed the efficient market hypothesis (EMH) as an academic concept of study through his published Ph.D. thesis [1] in the early 1960s, which later became a cornerstone of modern financial theory. The EMH states that it is impossible to consistently outperform the market by using any information that the market already knows, except through luck, because stock market efficiency causes existing share prices to always incorporate and reflect all relevant information. This means stocks always trade at their fair value on stock exchanges, and thus it is impossible for investors to profit from either purchasing undervalued stocks or selling stocks for inflated prices.

The EMH has provided the theoretical basis for much of the financial market research during the seventies and the eighties, but it has been put on trial recently: there is a large body of evidence in support of EMH [2], an equal amount of dissension also exists. For example, the existence of apparently sophisticated professional investors like Warren Buffett is an impossibility according to the EMH. Critics of EMH [3] argue that the predictability of stock returns reflects the psychological factors, social movements, noise trading, and other irrational factors in a speculative market.

> **Warren Buffett**, "I'd be a bum in the street with a tin cup if the markets were efficient." *Fortune* April 3, 1995.

The crux of the EMH is that it should be impossible to predict trends or prices in the market through fundamental analysis or technical analysis. If the EMH was true, then a financial time series could be modeled as the addition of a noise component at each step:

$$y(k+1) = y(k) + \epsilon(k)$$

where $\epsilon(k)$ is a zero mean Gaussian variable with variance σ. The best estimation is:

$$\hat{y}(k+1) = y(k)$$

In other words, if the series is truly a random walk, then the best estimate for the next time period is equal to the current estimate. Now, if it is assumed that there is a predictable component of the series then it is possible to use:

$$y(k+1) = y(k) + f(y(k), y(k-1), \cdots, y(k-n+1)) + \epsilon(k)$$

where $\epsilon(k)$ is a zero mean Gaussian variable with variance σ, and $f(\cdot)$ is a non-linear function in its arguments. The best estimation is given by:

$$\hat{y}(k+1) = y(k) + f(y(k), y(k-1), \cdots, y(k-n+1))$$

Prediction using this model is problematic as the series often contains a trend. A common solution to this is to use a model which is based on the first order differences, instead of the raw time series.

$$\delta(k+1) = \delta(k) + f(\delta(k), \delta(k-1), \cdots, \delta(k-n+1)) + \nu(k)$$

where

$$\delta \equiv y(k+1) - y(k)$$

and $\nu(k)$ is a zero mean Gaussian variable with variance σ, and in this case the best estimation is:

$$\hat{\delta}(k+1) = \delta(k) + f(\delta(k), \delta(k-1), \cdots, \delta(k-n+1))$$

Fig. 2. A high-level block diagram of the system used.

IV. System Details

Figure 2 depicts the hybrid intelligent system model for stock market analysis. We start with data preprocessing, which consists of all the actions taken before the actual data analysis process starts. It is essentially a transformation T that transforms the raw real world data vectors y_i, to a set of new data vectors x_i.

Then the data is fed into a SOM. The output of the SOM is the topographical location of the winning node. Each node represents one symbol. A brief description of the self-organizing map is contained in a later section.

An Elman recurrent neural network is then used which is trained on the sequence of outputs from the SOM. The Elman network was chosen because it is suitable for the problem, and because it has been shown to perform well in comparison to other recurrent architectures. The Elman neural network has feedback from each of the hidden nodes to all of the hidden nodes, as shown in Figure 3. For the Elman network:

A. Data Processing

1) Differencing: Whenever possible, large scale deterministic components, such as trends and seasonal variations, should be eliminated from the inputs.

For this study, the differences $[y_k - y_{k-1}]$ of the variables were provided to the networks so that different input variables can be compared in terms of relative change to the monthly stock returns, since the relative change of variables may be more meaningful to the models than the original values when forecasting a financial time series

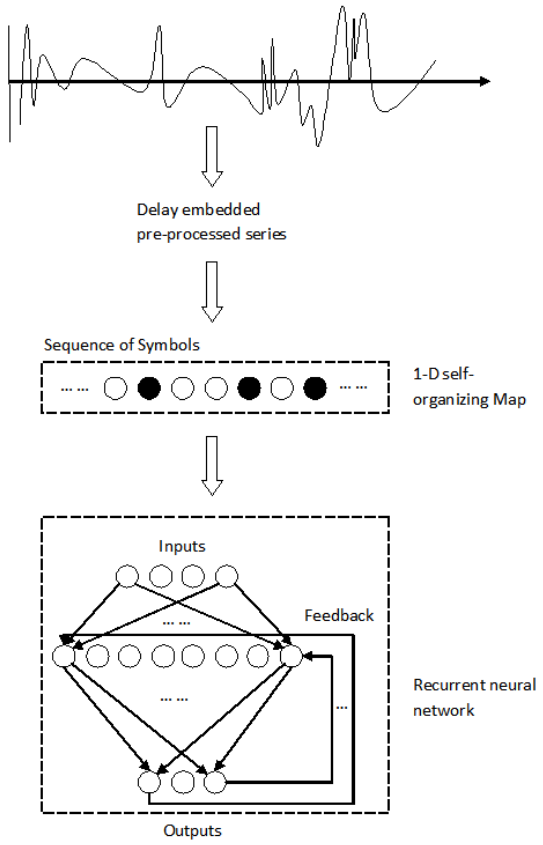


Fig. 3. The pre-processed, delay embedded series is converted into symbols using a self-organizingmap. An recurrent neural network is trained on the sequence of symbols.

2) Log conpression: In order to compress the dynamic range of the series and reduce the effect of outliers, a log transformation of the data is used:

$$x(k) = sign(\delta(k)) \log(|\delta(k)| + 1)$$

B. The Self-Organizing Map

A self-organizing map (SOM) is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two dimensional), discretized representation of the input space of the training samples, called a map. The map seeks to preserve the topological properties of the input space. The model was first described as an artificial neural network by the Finnish professor Teuvo Kohonen, and is sometimes called a Kohonen map.

Like most artificial neural networks, SOMs operate in two modes: training and mapping. Training builds the map using input examples. It is a competitive process, also called vector quantization. Mapping automatically classifies a new input vector.

Consider the problem of charting an n-dimensional space using a one-dimensional chain of Kohonen units. The units are all arranged in sequence and are numbered from 1 to

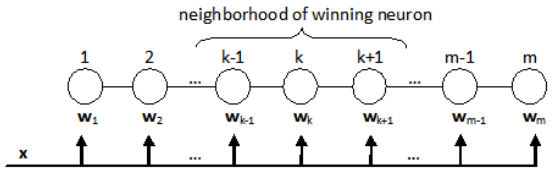


Fig. 4. Lattice of computing units in one-dimensional self-organizing map.

m (Figure 4). Each unit is connected to the n-dimensional input x and computes the corresponding excitation. The n-dimensional weight vectors $w_1, w_2, \cdots, w_m$ are used for the computation. When an input from such a region is fed into the network, the corresponding unit should compute the maximum excitation. Kohonens learning algorithm is used to guarantee that this effect is achieved.

A Kohonen unit computes the Euclidian distance between an input x and its weight vector w. This new definition of excitation is more appropriate for certain applications and also easier to visualize. In the Kohonen one-dimensional network, The neighborhood of radius r of unit k consists of all units located up to r positions from k to the left or to the right of the chain. Units at both ends of the chain have asymmetrical neighborhoods.

Kohonen learning uses a neighborhood function ϕ, whose value $\phi(i,k)$ represents the strength of the coupling between unit i and unit k during the training process. The learning algorithm for Kohonen networks is the following:

Kohonen learning Algorithm

- start : The n-dimensional weight vectors$w_1, w_2, \cdots, w_m$ of the m computing units are selected at random. An initial radius r, a learning constant η, and a neighborhood function ϕ are selected.
- step 1 : Select an input vector ξ using the desired probability distribution over the input space.
- step 2: The unit k with the maximum excitation is selected (that is, for which the distance between w_i and ξ is minimal, $i = 1, \cdots, m$).
- step 3 : The weight vectors are updated using the neighborhood function and the update rule

$$w_i \leftarrow w_i + \phi(i,k)(\xi - w_i), for\ i = 1, ..., m$$

- step 4 : Stop if the maximum number of iterations has been reached; otherwise modify η and ϕ as scheduled and continue with step 1.

After quantizing real-valued time series into symbolic streams, there are many ways to interpret the resulting symbolics. In particular, we perform quantization into symbolic streams over two and four symbols, respectively. The sequence S_t over the binary alphabet $\{1, 2\}$ is as follows:

$$S_t = \begin{cases} 1 \text{ (down)} & \delta_t < 0 \\ 2 \text{ (up)} & \delta_t \geq 0 \end{cases}$$

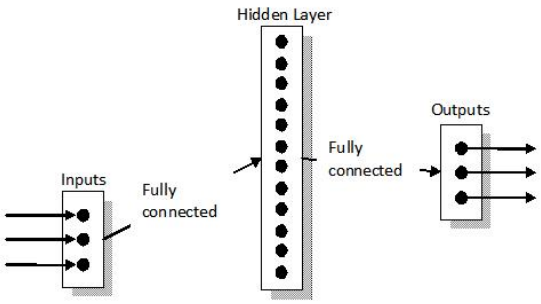


Fig. 5. Non-recurrent network. Multilayer perceptron with single hidden layer.

Quantization using four symbols is more complicated, since we have to determine the positions of the cut values θ_1 and θ_2 separating *normal* from *extremal* differences. The sequence over the alphabet $\{1, 2, 3, 4\}$ is as follows:

$$S_t = \begin{cases} 1 \text{ (extreme down)} & \text{if } \delta_t < \theta_1 < 0 \\ 2 \text{ (normal down)} & \text{if } \theta_1 < \delta_t < 0 \\ 3 \text{ (normal up)} & \text{if } 0 < \delta_t < \theta_2 \\ 4 \text{ (extreme up)} & \text{if } 0 < \theta_2 < \delta_t \end{cases}$$

C. Recurrent Neural Network

For time series forecasting, Neural network architectures can be trained to predict the future values of the dependent variables. What required is the design of the network paradigm and its parameters. The multi-layer feed-forward neural network approach (Figure 5) consists of an input layer, one or several hidden layers and an output layer. Another approach is known as the partially recurrent neural network that can learn sequences as time evolves and responds to the same input pattern differently at different times, depending on the previous input patterns as well.

Some argue that the use of a recurrent neural network instead of an MLP with a window of time delayed inputs introduces another assumption by explicitly addressing the temporal relationship of the inputs via the maintenance of an internal state [6].

None of these approaches is superior to another in all cases [7]; however, an additional dampened feedback, that possesses the characteristics of a dynamic memory, will improve the performance of both approaches.

We can convert a non-recurrent network into an Elman network that remembers a previous state by adding a new set of inputs which are fully connected by recurrent links to the hidden layer outputs (but delayed by one unit of time).

1) Input Delayed Neural Networks: Time-delay neural networks (TDNNs), also known as the neural network finite impulse response architecture has been used quite successfully in a number of practical applications including speech recognition, and time series prediction.

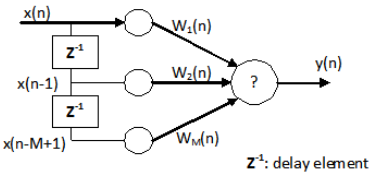


Fig. 6. Input delayed neural network.

A TDNN is similar to a multilayer perceptron in that all connections feed forward. The difference is that with the TDNN, the inputs to any node i can consist of the outputs of earlier nodes not only during the current time step , but during some number d of previous time steps $(t-1, t-2, \cdots, t-d)$ as well. This is generally implemented using tap-delay lines.

A natural restriction of the general TDNN topology is the class of TDNN architectures which have delays only on the input units. We call these input delayed neural networks (IDNNs). An IDNN which has a single output unit, and $d-1$ delays on its only input (Figure 6) computes a function of the most recent d inputs including the current input.

$$y(n) = f\left(\sum_k w_k \cdot x(n-k+1)\right)$$

where, f is a sigmoid function and the weights are corrected by the Back Propagation algorithm (BP).

2) Elman Networks: Elman Networks (Figure 7) are a form of recurrent Neural Networks which have connections from their hidden layer back to a special copy layer. This means that the function learnt by the network can be based on the current inputs plus a record of the previous state(s) and outputs of the network. In other words, the Elman net is a finite state machine that learns what state to remember (i.e., what is relevant). The special copy layer is treated as just another set of inputs and so standard back-propagation learning techniques can be used.

$$y(n) = f\left(\sum_k w_k \cdot x_k(n) + \sum_m w_m \cdot y_m(n-1)\right)$$

V. Stationarity

A common assumption in many time series techniques is that the data are stationary, but in reality, data points are often non-stationary. A stationary process has the property that the mean, variance and autocorrelation structure do not change over time. Non-stationary behaviors can be trends, cycles, random walks or combinations of the three.

Using non-stationary time series data in financial models produces unreliable and spurious results and leads to poor understanding and forecasting. If the time series is not stationary, we can often transform it to stationarity with one of the following techniques.

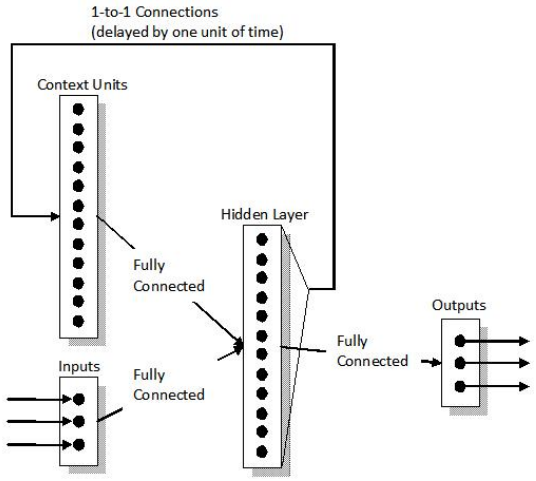


Fig. 7. Elman network. There is a new set of inputs which are fully connected by delay element to the hidden layer outputs.

A random walk can be transformed to a stationary process by differencing and then the process becomes difference-stationary. That is, given the series Y_t, we create the new series $Z_i = Y_i - Y_{i-1}$. The disadvantage of differencing is that the process loses one observation each time the difference is taken. Although you can difference the data more than once, first order differencing is usually sufficient.

A non-stationary process with a deterministic trend becomes stationary after removing the trend, or detrending. We can fit some type of curve to the data and then model the residuals from that fit. For example, $Y_t = \alpha + \beta t + \epsilon t$ is transformed into a stationary process by subtracting the trend $\beta t : Y_t - \beta t = \alpha + \epsilon t$. Since the purpose of the fit is to simply remove long term trend, a simple fit, such as a straight line, is typically used. No observation is lost when detrending is used to transform a non-stationary process to a stationary one.

For non-constant variance, taking the logarithm or square root of the series may stabilize the variance. For negative data, you can add a suitable constant to make all the data positive before applying the transformation. This constant can then be removed from the model to obtain predicted (i.e., the fitted) values and forecasts for future points.

The approach used to deal with the non-stationarity of the signal in this work is to difference original data followed by a logarithm transformation and build models based on a short time period only. This is an intuitively appealing approach because one would expect that any inefficiencies found in the market would typically not last for a long time.

VI. Experimental Results

In order to gauge the effectiveness of the symbolic encoding and recurrent neural network, we investigated the following five systems:

TABLE I
CONFIGURATION OF SOM

Parameter	Value
Dimention	1
# of nodes	2
Learning rule	Kohonen
Learning rate	0.01
Epochs	50

TABLE II
CONFIGURATION OF IDNN.

Parameter	Value
# of hidden nodes	5
Learning rule	BP
Learning rate	0.01
Momentum constant	0.9
Transfer function of hidden layer	tansig
Transfer function of output layer	purelin

1) The system as presented in Figure 3 with SOM and Elman network. The configurations of SOM and Elman network are described in Table I and Table III respectively.
2) The system above but without the symbolic encoding, i.e. the preprocessed data is entered directly into the recurrent neural network without the SOM stage.
3) The system with the recurrent network replaced by a standard MLP network. The configurations of MLP is described in Table II .
4) Standard MLP network only.
5) Simple linear regression model.

Linear regression can be easily implemented and converges fast. But even test it against training data, it can only predict direction of changing 75% of time. We get $R^2 = 0.1733$, the F statistic = 0.6290 and a p value for the full model = 0.6511, and an estimate of the error variance = 22.4130. Every number is sugguesting that linear regression model failed to capture the nature of the problem, which has been pointed out earlier in this report.

Algorithm of SOM is quite simple. As shown in Figure 8, it did a very good job in terms of classification. In this

TABLE III
CONFIGURATION OF ELMAN NETWORK.

Parameter	Value
# of hidden nodes	10
Learning rate	0.01
Momentum constant	0.9
Transfer function of recurrent layer	tansig
Transfer function of output layer	purelin

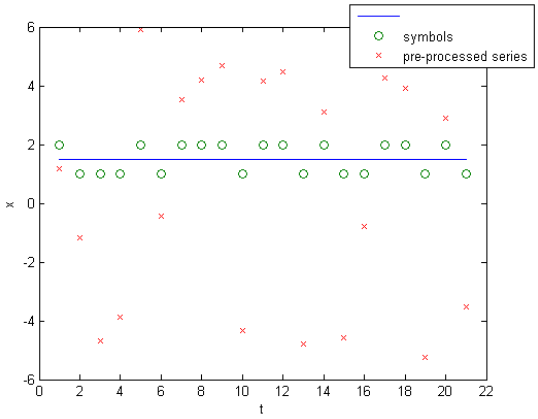


Fig. 8. Symbolical encoding by SOM

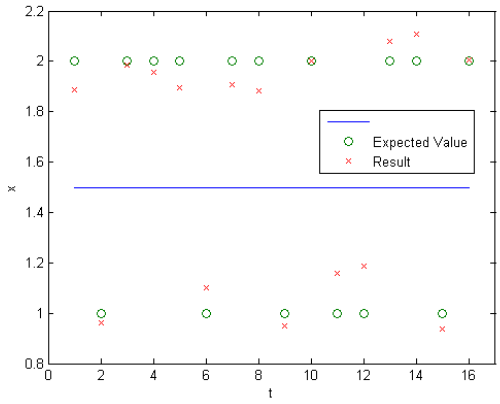


Fig. 9. simulation using training data

TABLE IV
THE RESULTS FOR THE SYSTEMS.

Model	Test error %
SOM + Elman	0.525
Elman	0.53
SOM + MLP	0.4514
MLP	0.445

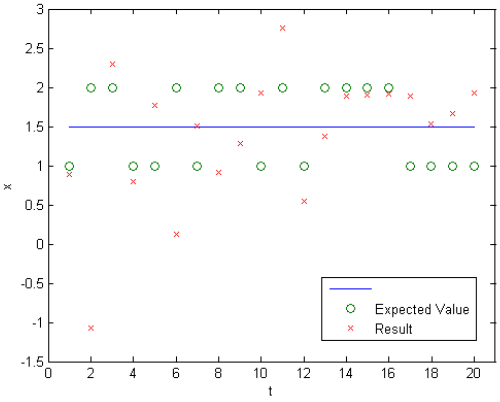


Fig. 10. simulation using test data

particular case, only two neurons are used (one means stock index will go up and the other means it will go down). In future, sensitive study on numbers of dimentions and neurons will be conducted.

Two types of recurrent neural networks have been used here, both Elman networks and MLP. For Elman networks, after less than 100 epochs training, it can perfectly simulate training data almost all the time (Figure 9). That is not surprising because basically it can approximate any function (with a finite number of discontinuities) with arbitrary accuracy if the hidden layer have enough neurons.

But when presented with testing data to simulate a real prediction (forecasting) case, the best performance was obtained with an embedding dimension of 8 and 10 hidden neurons where the error rate was 58.0% (Figure 10). A t-test indicates that this result is significantly different from the null hypothesis corresponding to a random walk at $p = 5.3854^{-3}$.

Sensitive analysis was done for both embedding dimmension number and hidden neuron number. The result are shown in Figure 11 and 12.

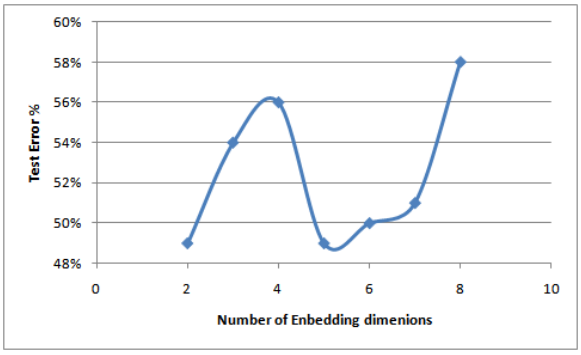


Fig. 11. Sensitive analysis of embedding dimmension

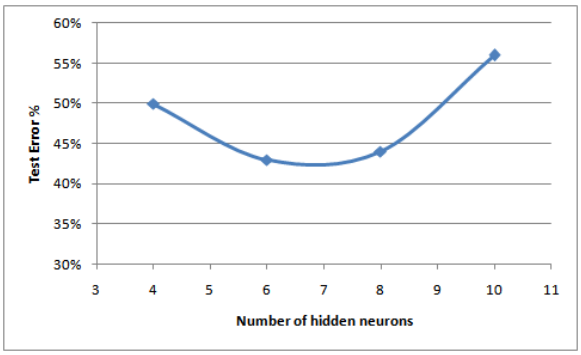


Fig. 12. Sensitive analysis of number of hidden neurons

VII. CONCLUSION

There exists a vast number of articles addressing the predictabilities of stock market return, and many of them are claiming that the simulation can benifit greatly from quantization of the original data into symbolic streams. And also there is no conclusion whether the use of a recurrent neural network, including the temporal relationship of the series explicitly in

the model, is signi?cant for this type of study. The author did a shoulder by shoulder comparison of five diffenrent models, Elman network, MLP network, Elman network with SOM filter, MLP with SOM filter and simple linear regression. The result of the simulation shows that SOM can greatly improve the convergence of the neuron networks and Elman network can do a better job to capture the temporal pattern of the symbolic streams generated by SOM. Simple linear regression is almost unusable in this case. Some sensitivity study was done with respect to neural network parameters. A sensitivity study of SOM parameters, which could be significant here, is left to future work.

APPENDIX
SIMULATION DETAILS

Gist only

- Data: Dow Jones Industrial Average Index from 09/14/06 to 11/23/07. Each time we train the network with 20 numbers and test it with one day data right after the training segment. Google Finance.
- Linear regression: simple linear regression.
- SOM: one dimention model with two neurons. Kohonen Learning Rule. Learning rate = 0.01. 250 epochs.
- Elman networks: tansig neurons in its hidden layer. purelin neurons in its output layer. Batch training. Performance index: mean absolute error. Ten hidden-layer tansig neurons and a single logsig output layer.

ACKNOWLEDGMENT

The authors would like to thank Professor Willard L. Miranker for reviewering an early draft of this report.

REFERENCES

[1] Fama, Eugene (1965). "The Behavior of Stock Market Prices". *Journal of Business* 38: 34-105.

[2] Burton G. Malkiel. "The Efficient Market Hypothesis and Its Critics", CEPS Working Paper No. 91

[3] La Porta R., Lakonishok, J., Shliefer, A. and R. Vishny [1997] "Good news for value stocks: Further evidence on market efficiency", *Journal of Finance* 52, 859-874.

[4] Pavlidis, N.G.; Tasoulis, D.K.; Vrahatis, M.N., "Financial forecasting through unsupervised clustering and evolutionary trained neural networks," *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on* , vol.4, no., pp. 2314-2321 Vol.4, 8-12 Dec. 2003

[5] Peter Tino and Christian Schittenkopf and Georg Dorffner, "Temporal Pattern Recognition in Noisy Non-stationary Time Series Based on Quantization into Symbolic Streams: Lessons Learned from Financial Volatility Trading"

[6] C. Lee Giles, Steve Lawrence, Ah Chung Tsoi, "Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference", *Machine Learning*, Volume 44, Issue - 1, pp161-183, 2001-07-01

[7] B.G. Horne and C.L. Giles. "An experimental comparison of recurrent neural networks." *In Advances in Neural Information Processing Systems 7*, pages 697–704. MIT Press, 1995.

Feature Extraction Using Self Organizing Map

Lin He

***Abstract*—Given a list of movies, a number of people, and the viewership information, we determine clusters consisting of similar movies or people. Different from the classical clustering problem, movies and people can belong to multiple clusters. We will introduce two variations of SOM algorithms to find these clusters. One uses inhibitory synapses to prevent neurons from converging to the same value, and serves to obtain a preliminary cluster grouping. In the second, each input, instead of only examining the closest neuron, operates on all the neurons that could possibly represent the clusters this input belongs to. This algorithm serves to complete the preliminary cluster grouping. A few testcases are given to demonstrate the performance of the algorithms in practice.**

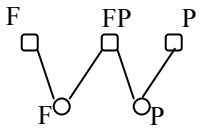
***Index Terms*—Self Organizing Map, Feature Extraction, Inhibitory Synapses**

I. Introduction

Given a fixed number of movies and viewers and the information on who watched what, we seek to determine the reasons behind the choices viewers made.

We assume that each movie has a number of features, and each viewer prefers a number of features. If a movie has a feature that a viewer prefers, we assume that this particular viewer has watched that movie.

For example, suppose there are three movies and two viewers. The viewership information is described in the following graph (squares are movies, and circles are viewers):



F and P are features (F stands for "funny" and P stands for "political"). From the above graph, we can see that the second movie is both funny and political, and the first viewer likes funny movies.

Feature is a useful abstraction because it reveals why a particular viewer watched the movies in question. If we know the features, then when there is a new viewer, we can analyze the movies s/he watches, directly associate him/her with the correct features, and make appropriate movie recommendations. It is easy to see how this approach can also be applied to website or book recommendations.

II. Mathematical Model

Suppose there are m viewers, then we can associate a feature with a vector in the m-dimensional space. If viewer i likes feature X, then the i-th component of X is 1; otherwise it is 0.

For the graph we gave, the two feature vectors are:

$$F=(1,0)$$
$$P=(0,1)$$

Analogously, we also associate a movie with a vector in m-dimensional space. If viewer i watches movie Y, then the i-th component of Y is 1; otherwise it is 0. For the graph we gave:

$$M_1=(1,0)$$
$$M_2=(1,1)$$
$$M_3=(0,1)$$

Now we define the operator "|":

$$A|B = \mathrm{sgn}(A+B)$$

sgn(A) is a vector, the i-th component of which is the sign of the i-th component of A. Specially, the sign of a positive number is 1, the sign of 0 is 0, and the sign of a negative number is -1.

"|" is in fact an extension of the logic OR operation. 1 corresponds to TRUE, and 0 corresponds to FALSE.

Now the assumption we made earlier can be formally stated as follows: if a movie M has features $F_1, F_2, \ldots, F_k$, then:

$$M=F_1|F_2|\ldots|F_k$$

We call this **Cover Condition**.

Another obvious property is: if M has feature F, then:

$$M|F = M$$

Hereafter, we may also say vector M "contains" vector F (which suggests that movie M has feature F).

The problem we are trying to solve can be formally stated as follows. We have n movie-vectors in a m-dimensional space, $M_1, M_2, \ldots, M_n$. Find r feature-vectors $F_1, F_2, \ldots, F_r$, so that for every M_i ($1 \le i \le n$), all the feature-vectors M_i contains, when connected by "|" operator, evaluate to M_i. We want to find the smallest possible r.

We define two more operators.

Operator "•":

Let $A=(x_1, x_2, \ldots, x_m)$, $B=(y_1, y_2, \ldots, y_m)$
then $A \bullet B=x_1y_1+x_2y_2+\ldots+x_my_m$

Operator "*":

Let $A=(x_1, x_2, \ldots, x_m)$, $B=(y_1, y_2, \ldots, y_m)$
then $A*B=(x_1y_1, x_2y_2, \ldots, x_my_m)$

We also introduce the notation:

$$U=(1,1,\ldots,1)$$

So we know A•U equals to the sum of all components of A.

We make two other assumptions:

1. F_i and F_j ($i \neq j$) should not be too similar; otherwise it is difficult to distinguish the two features. Formally, both $(F_i \bullet F_j)/(F_i \bullet U)$ and $(F_i \bullet F_j)/(F_j \bullet U)$ should be small.
2. It is easy to see that movies and viewers are exchangeable. If we interchange the role of movie and viewer, condition (1) should still hold.

The above the conditions are to ensure that viewers and movies are different enough to distinguish features.

III. Self-Organizing Map

The problem we are dealing with, in many ways, resembles a classical clustering problem.

In the classical clustering problem, a number of points are distributed in a space, forming several clusters. A fixed number of neurons are randomly set up in the same space.

All the points are repeatedly processed. In every processing step, the point finds the closest neuron, and pulls the neuron toword itself. The strength with which it pulls is proportional to the distance between the point and the chosen neuron. The farther the distance, the harder it pulls[1].

After a while, the neurons end up in clusters. Since a point only pulls the closest neuron, a neuron is only pulled by points in its cluster. Since all points in the cluster pull the neuron, the neuron eventually stablizes at the center of the cluster as forces from many directions tend to cancel each other out.

In our problem, movies can be viewed as the points. If two movies M and N do not share any feature, M•N should be relatively small (according to condition (1)). In this sense, they are "far apart." Similarly, if two movies M and N share some features, M•N will be relatively large. In this sense, they are "close."

Hence, movies with the same features "cluster" together in the m-dimensional space.

If we can set up an appropriate number of neurons in the m-dimensional space, and apply an appropriate SOM, these neurons should end up at the clusters.

Let's examine a hypothetical cluster consisting of three movies:

$$M_1=A|B$$
$$M_2=A|C$$
$$M_3=A|D$$

M_1, M_2, M_3 are movies. A, B, C are features.

If a neuron N ends up somewhere close to the cluster, it will be pulled by all three neurons. Whenever N is pulled, the "1" components of A are always strengthened (that is, increased). In contrast, most "1" components in B are only strengthened 1/3 of the time, but weakened 2/3 of the time. Similarly for C and D. The net effect is that "1" components of A appear in N too; whereas "1" components of B, C, D are weakened to close to 0 in N. The final neuron N will be very close to feature A.

To put this in the perspective of the classical clustering problem, the center of M_1, M_2, and M_3 is A. The other three features B, C, and D are merely noise. They tend to cancel each other out.

After noticing the resemblance to the classical clustering problem, we will focus on designing a SOM algorithm to solve this viewer-feature problem. However, there is a significant difference between the classical SOM algorithm and the one we need to deal with.

In our problem, in order for a neuron to end up as a correct feature, it has to be pulled by all the movies that have this feature. If only some of the movies containing this feature pull at the neuron, the noise influences might not cancel each other out cleanly; as a result, the neuron may end up somewhere far from the intended feature.

In other words, if a movie has k features, it has to pull all k neurons representing these features. This is a considerable extension to the classical clustering problem which only deals with the closest neuron.

So the basic idea is as follows (assuming we know the number of features in advance. Later we'll discuss how to determin the number of features).

1. First randomly set up some neurons in the space.
2. All the movies are processed repeatedly.
3. For each input (movie), find all the neurons it "contains."
4. Pull each one of these neurons toward that input movie, the strength with which it pulls being proportional to the distance between the neuron and the input movie.

Formally, the algorithm statement is:

Stablizer Algorithm:

1. **Randomly set up r vectors of m-dimension, N_1, N_2, …, N_r.**
2. **Randomly select a movie M to process.**
3. **If $M|N_i = M$ $(1 \le i \le r)$, $N_i = N_i + \mu(M - N_i)$**
4. **Repeat (2) and (3) for a reasonable number of times.**

μ is the learning rate. It is set to 0.05 in my program.

We expect the algorithm to work iff each $N_i (1 \le i \le r)$ is close to a particular feature enough to be identified as a distinct feature. It only serves to expand and stabilize incomplete neurons when they are sufficiently close to the actual features.

I experimented with a number of algorithms based on the above schema, but did not succeed in any one of them. Here is a typical failure:

Expected Result	My Result
THREE features	THREE features
(1, 0, 0)	(1, 0, 0)
(0, 1, 0)	(1, 0, 0)
(0, 0, 1)	(0, 0, 1)

The main reason is that multiple neurons could end up at exactly the same location.

In other words multiple neurons end up in the same cluster. This problem does not exist in the classical SOM algorithm because every input finds a closest neuron, hence every cluster has an impact on the final distribution of the neurons. But in this problem, if we follow the above schema, an input (movie) could very well not contain any neuron simply because none of the neurons converged to the particular features this movie contains. So some clusters are ignored. As a result, multiple neurons could end up in the same cluster. The option of being able to pull multiple neuron also causes the possibility of not pulling any neuron. What an irony!

IV. Inhibitory Synapses

To address the difficulty summarized in the last section, I appended inhibitory synapses among neurons.

More specifically, I impose the following **Sum Condition**:

$$N_1+N_2+\ldots+N_r=U$$

Whenever any change is made to any neuron, all the neurons are normalized so that the Sum Condition always holds.

This essentially forces every viewer to appear in one and only one neuron. This effectively addresses the difficulty mentioned in the last section.

However, with the Sum Condition comes a severe constraint: since a viewer can only appear in one of the neurons, what if s/he happens to like multiple features?

For example in,

$$F_1=(1,1,0)$$
$$F_2=(0,1,1)$$

the second viewer likes both features. How should the neurons represent the overlap of features? We have two options:

Option 1	Option 2
$N_1=(1, 0.5, 0)$	$N_1=(1, 1, 0)$
$N_2=(0, 0.5, 1)$	$N_2=(0, 0, 1)$

Option 1 seems reasonable, but experiments show it is actually very unstable.

Option 2 forces the multi-interest viewer into one of the neurons. In Option 2, N_2 still represents F_2, but N_2 only has part of F_2's "1" components. We denote N_2 as the **"reduced form of feature F_2."**

Now we want to design an SOM algorithm to get a set of neurons, each of which represents the "reduced form" of a distinct feature. Once we have these neurons, we can apply the Stablizer Algorithm to expand the "reduced form" to its full representation.

V. Feature Extraction Algorithm

Feature Extraction Algorithm:

1. **Randomly set up r neurons.**
2. **Process a random movie M.**
3. **Define f(N, M) as the degree to which M contains N.**
4. **For $N_i(1\le i\le r)$, $N_i=N_i+ f(N,M)(\mu N_i*sgn(M-N))$**
5. **Normalize $N_i(1\le i\le r)$ to meet the Sum Condition.**
6. **Repeat (2), (3), (4) and (5) until all $N_i(1\le i\le r)$ converge.**

Note:

a). μ is the learning rate. It is set to 0.05 in the experiments.

b). sgn(A) is the sign vector of vector A. Specifically, The i-th component of sgn(A) is the sign of i-th component of A. The sign of 0 is 0.

c). $f(N, M)=(N*N*N*N)\bullet M / (N*N*N*N)\bullet U$
where $(U=(1,1,\ldots,1))$

The critical part of the algorithm is step 4.

f(N,M) is a function that indicates "the degree" to which M contains N. f(N,M) ranges from 0 to 1.

If we follow the example of Stablizer Algorithm, we only let M pull N when M completely contains N, that is f(N, M)=1. This is not going to work in practice because initially none of the neurons are specialized enough to be contained by any input (movie). So f(N, M) is best used as a measure of how hard M pulls N, rather than an absolute cut-off of whether M should pull N.

A natural design of f(N, M) would be:

$$f(N, M) = (N\bullet M)/(N\bullet U)$$

In other words, if we only look at non-zeros, the above function is the part of N that overlaps with M as a percentage of the whole of N. But this definition does not work well. Consider the following case:

$$M=(1, 1, 0)$$
$$N_1=(0.05, 0.6, 0.65)$$
$$N_2=(0.30, 0.35, 0.65)$$
$$(N_1\bullet M)/(N_1\bullet U)=(0.05+0.6)/(0.05+0.6+0.65)=0.5$$
$$(N_2\bullet M)/(N_2\bullet U)=(0.30+0.35)/(0.30+0.35+0.65)=0.5$$

So $f(N_1,M)=f(N_2,M)$.

However, N_1 has a great potential of developing into (0, 1, …), whereas N_2 is likely to develop into (0, 0, …) because the first two components of N_2 are not strong enough to be pulled sufficiently.

In other words, we should prefer concentrated components over spread-out components. So I designed the following:

$$f(N, M)=(N*N)\bullet M / (N*N)\bullet U$$

When multiplied by itself, a smaller fraction diminishes faster than a larger fraction. So the above function will award focused components. Indeed with the new function, $f(N_1, M)>f(N_2, M)$.

However, the function does not work well for some testcases. I suspected it was not strong enough, so I awarded concentrated components even more amply:

$$f(N, M)=(N*N*N*N)\bullet M / (N*N*N*N)\bullet U$$

Experiments with this function could pass most testcases I designed.

Another point worth noting in step 4 of the Feature Extraction Algorithm is this change from the customary SOM:

$$\text{Customary: } \Delta = \mu(M-N)$$
$$\text{Here: } \Delta = \mu N_i*sgn(M-N)$$

In the customary algorithm, the pull is proportional to the distance between the input and neuron, while here it is proportional to how strong each component of the neuron is!

As mentioned in the last section, when a viewer likes multiple features, s/he should be forced into one of the neurons representing these features. In other words, we want every component of every neuron to converge to 0 or 1.

This is exactly what we are doing. When a component of N_i is large (close to 1), we want to encourage its growth, hence Δ is large. When a component of N_i is small (close to 0), we want it to stay there, hence Δ is small.

VI. Algorithm in Summary

Now assume we know the number of features r. We can run the Feature Extraction Algorithm.

Unfortunately, the Feature Extraction Algorithm does not always gives satisfactory results. Sometimes a neuron is a combination of multiple features. Sometimes multiple neurons represent the same feature in different "reduced forms."

So we run the Feature Extraction Algorithm multiple times.

Each time it is finished, we check two things:

1. Every neuron is fully contained by at least one movie.
2. Every movie contains at least one neuron.

If either of these two conditions fails to hold, we have to run the FEA again with a different set of initial values. If after a specified number of runs the algorithm still fails to find a good result, we call the result a FAILURE.

If we get a set of reduced forms of features satisfying the above two conditions, we run the Stablizer Algorithm to complete the neurons.

After SA is finished, we check if every movie is fully covered by the neurons it contains (Cover Condition). If yes, we have a solution!

If even one movie is not fully covered, we have to go back to FEA and run it again, expecting a better result. Again, if after a specified number of runs of FEA+SA, no solution is found, the result is a FAILURE.

So what happens when a FAILURE happens?

In the beginning we assumed that we know r (the number of features) in advance. This is certainly not true. So in fact what the methodology does is this:

1. r←1
2. Run FEA+SA as described above.
3. If a solution is found, print. Otherwise r←r+1, and go back to (2).

Not surprisingly, if $r = a$ produces a good result, all values greater than a will also produce good results. When there are more neurons than features, some features will duplicate, while others will combine with others to form new enlarged features. For complicated cases, the methodology sometimes gives solutions slightly larger than the necessary number of features.

VII. TEST

For convenience, features are indexed A, B, C, … Here is a typical input file:

3
10
A B C A B C AB AC BC ABC
10
A B C AB AC BC ABC A B C

The number "3" is the number of features.

The first number "10" is the number of viewers.

The following 10 strings indicate which features the viewers like.

The second number "10" is the number of movies.

The following 10 strings indicate which features the movies have.

The features vectors should be:
A: (1,0,0,1,0,0,1,1,0,1) or 1 4 7 8 10
B: (0,1,0,0,1,0,1,0,1,1) or 2 5 7 9 10
C: (0,0,1,0,0,1,0,1,1,1) or 3 6 8 9 10

My program reads in these data, calculates the movie vectors, and then, based on the movie vectors, calculates the features using algorithms described above.

The output for this particular input is:
1 4 7 8 10
3 6 8 9 10
2 5 7 9 10

It is not necessarily in the same order as A, B, and C; but it covers all the features.

VIII. CONCLUSION

Two variations of SOM combined provided a reasonably good solution to the Feature Extraction Problem. In practice, when every movie/viewer associates with at most three features, the algorithm produces very accurate results. However, the whole project is based on the assumption that movies and viewers are different enough to distinguish different features. Whether the premise holds true in the real world still remains to be investigated.

There are several of natural concerns about the algorithm's practical value.

The first concern is whether people actually watch all the movies that have their preferred features. As mentioned before, we can select a relatively small sample of the most-watched movies and the most active viewers from a huge pool of candidates. These fanatic fans are likely to have checked out all the popular movies. Moreover, these viewers and movies are likely to cover a wide range of features; or rather, features not covered by these movies/people are the less popular ones, hence of less significance, which we can afford to ignore.We can use them to calculate features, and then use them as benchmarks to place other people and movies into the appropriate feature categories.

The second concern is scalability since there are an enormous number of movies available. The number of viewers is even larger. But this is not a big issue since we have decided to work on the most-watched movies and most active viewers. In addition, if there are too many movies or viewers, we can use a variation of SOM algorithm, propoosed by Andrew R. Pariser, to reduce the dimensionality first[2].

APPENDIX

I list a couple of non-trivial testcases here.

Testcase 1:

Every viewer and movie possesses at least two features.

6
15
AB AC AD AE AF BC BD BE BF CD CE CF DE DF EF
15
AB AC AD AE AF BC BD BE BF CD CE CF DE DF EF

Expected Output	My Output 1	My Output 2
SIX features	SIX features	SEVEN features
4 8 11 13 15	4 8 11 13 15	5 9 12 14 15
3 7 10 13 14	3 7 10 13 14	3 7 10 13 14
5 9 12 14 15	5 9 12 14 15	4 8 11 13 15
2 6 10 11 12	2 6 10 11 12	2 6 10 11 12
1 2 3 4 5	1 2 3 4 5	1 6 7 8 9
1 6 7 8 9	1 6 7 8 9	1 2 3 4 5
		1 2 3 4 5

Since the algorithm is randomized, my output mostly alternates between the above two. As we can see, when the algorithm gets unlucky, and fails to find a solution in six

features (which is the correct answer), it finds the solution with one more feature. Although there is a duplicate in the seven-feature output, it is easy to detect and remove.

Testcase 2:

On top of the previous case, a number of movie/viewer with three features are added.

6
21
ABC BCD CDE DEF EFA FAB
AB AC AD AE AF BC BD BE BF CD CE CF DE DF EF
21
ABC BCD CDE DEF EFA FAB
AB AC AD AE AF BC BD BE BF CD CE CF DE DF EF

Expected Output	My Output
SIX features	SEVEN features
1 2 6 7 12 13 14 15	1 2 6 7 12 13 14 15
2 3 4 9 13 16 19 20	2 3 4 9 13 16 19 20
1 2 3 8 12 16 17 18	2 3 4 9 13 16 19 20
1 5 6 7 8 9 10 11	1 2 3 8 12 16 17 18
3 4 5 10 14 17 19 21	1 5 6 7 8 9 10 11
4 5 6 11 15 18 20 21	3 4 5 10 14 17 19 21
	4 5 6 11 15 18 20 21

Again, there is a duplicate in my output. But the whole thing is pretty much the same as the desired output.

Interestingly, Pariser's algorithm [2] would not do very well with the two inputs presented here. His algorithm forces every movie to be associated with exactly one group. Unfortunately, in these two inputs, every movie and, morever user, is associated with at least two features. In other words, none of the movies can be cleanly classified into one category. Our algorithm effectively disentangles the plurality in association.

Admittedly, the current algorithm runs very slowly on large data, but I believe improvements on converging speed can be made, and techniques on reducing the number of iteration should be attainable. Due to the constraint of time, we do not pursue further along the line of efficiency.

ACKNOWLEDGMENT

Thank Professor Willard Miranker from the Department of Computer Science, Yale University for giving insights into the problem.

Thank Laura Gehring, Yale College'08 for introducing the idea of investigating movie-viewer relationship.

REFERENCES

[1] Haykin, Simon. Neural Networks, Second Edition. 1999. Princeton Hall Press. New Jersey.
[2] Pariser, Andrew R. Dimensionality Reduction via Self-Organizing Feature Maps for Collaborative Filtering.

Introducing Affect into Competitive Game Play

Maha Alabduljalil

***Abstract*— A dynamic notion of affect (degree of satisfaction) that an agent acquires in competitive iterated game play is developed. Simulated play against both a random environment and a competitor agent is used to study the impact of affect on game playing strategies. For definiteness, the formulation is framed in terms of stock trading. In an appendix, comments on how affect and game play inform a notion of consciousness are given.**

***Index terms*- Affect, Consciousness, Tit-for-two-tat, anti-tit-for-tat, stock trading**

I. Introduction

A notion of affect is introduced into game playing strategies[(1)]. For definiteness and clarity we frame our study in terms of stock trading. So affect maybe thought of as an experience of feeling or emotion of the investor in response to his performance. Affect will playa key part in an investor's interaction with stimuli. In a direct competition between a pair of players, affect also refers to affect display, which is a facial, vocal, or gestural behavior that serves as an indicator of the investor's feelings. For clarity we restrict affect to represent two situations, satisfaction or dissatisfaction, and so we specify affect as follows.

$$a(x) = \mathrm{sgn}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \qquad (1)$$

Here x is the difference between the predicted payoff of a round of play and the actual payoff as we shall see.

We consider a basic case and an advanced case and study both by simulation. The basic case involves an investor playing against an environment represented by random market behavior. If the simulation is able to produce success against a random market, playing against real market behavior is likely to be easier to achieve. The advanced case will involve two competing investors (players), each choosing a distinct strategy.

Strategy superiority is based on both a notion of accumulated satisfaction (affect) of the investor with his play and his total return. Affect, is informally characterized as the satisfaction a person feels when he has done something right. This corresponds to a positive value of affect. Reversely, dissatisfaction is taken as a negative value of affect. We shall see that one particular strategy majorizes all other strategies considered.

For simplicity in describing the process, only binary moves are considered. For instance, in the basic case the rising or falling of a stock price is represented by ±1, respectively. Similarly, ±1 represents the investor's order to buy or sell a stock. The advanced case is similar; each investor either buys or sells, represented by ±1. In the advanced case, one investor's reaction provides stimulus for the second. In both cases, the responser is a function of affect, response, stimuli and round of play (indexed by j) as follows:

$$r_j = \mathrm{sgn}\ \rho(A_j, R_j, S_{j+1}, j) \qquad (2)$$

Here $r_j = \pm 1$ is the investor's response, i.e. his decision to buy or sell.

$R_j = r_0,..,r_{j-1}$
$A_j = a_1,..,a_j$
$S_j = s_0,...,s_{j-1}$

Where $s_j = \pm 1$ is the stimulus from the environment or the competing investor as the case may be.

We shall make a number of choices for ρ. To evaluate the process, we include some well known strategies. These are tit-for-two-tat and anti-tit-for-tat. Notice that j characterizes a non-autonomous dependence on the environment.

II. Methods

A. Basic Case: Player vs. Environment

Here an investor (player) employs one of several different strategies verses random changes in stock price. Simple factors are used for the prediction of the response. These are the number of times a favorable buy was omitted (P), number of times a favorable sale was omitted (L) and the overall satisfaction of the investor. The response (±1) will be specified as a unique function of these factors for each strategy as we

(1) This work was motivated by a set of formal unpublished notes on modeling affect by W. Miranker and G. Zuckerman.

shall see.

There are four strategies illustrated. First, the tit-for-two-tat (T2T), which we take to represent a naïve or a forgiving investor. This investor keeps on either buying or selling for consecutive rounds of play. He will switch his action (buy/sell) only if he is dissatisfied for two consecutive rounds of play. The response strategy is described as follows.

$$\text{T2T:}\quad r_j = \text{sgn}(a_{j-1} + a_{j-2})r_{j-1} \tag{3}$$

Here $a_j, j > 0$ is the affect at play j.

The second is the anti-tit-for-tat strategy (ATT). This is considered the riskiest. The investor keeps alternating his response, disregarding his experience. So, he buys then sells then buys and so on. ATT is described as:

$$\text{ATT:}\quad r_J = -r_{j-1} \tag{4}$$

Third, we develop a new strategy called "Developed Strategy" (DS).

$$\text{DS:}\quad r_j = \begin{cases} r_{j-1} & \alpha_j < 0 \\ -r_{j-1} & \alpha_j \geq 0 \end{cases} \tag{5}$$

Here α_j is the accumulated affect over some specified number N of prior plays.

$$\alpha_{j,N} = \sum_{i=j-N+1}^{j} a_j \tag{6}$$

The computations shown in section C employ N=100 or 500 round of play.

The fourth strategy is called DS-Factors (DSF). Suppose there are to be N rounds in a trial. For the first N/2 rounds of play, the investor will follow T2T (suggesting that he's new to the market place). Then, for second half of the trial his response depends on the affect accumulating in his mind. So, the investor learns from his past performance.

$$\text{DSF:}\quad r_j = \begin{cases} \text{sgn}(a_{j-1} + a_{j-2})r_{j-1} & j \leq N/2 \\ r_{j-1} & j \geq N/2 \wedge \alpha_j \geq 0 \\ -r_{j-1} & j \geq N/2 \wedge \alpha_j \leq 0 \end{cases} \tag{7}$$

Then for $j \geq N/2$ (To account losing trades) we take

$$r_j = \begin{cases} +1 & P_j \geq \theta \wedge L_j < \theta \\ -1 & L_j \geq \theta \wedge P_j < \theta \end{cases} \tag{8}$$

where $\theta = 0.4$ is an arbitrarily specified parameter.

Here

$$P_j = \sum_{i=1}^{j} k \quad \begin{cases} k = 1 & when\ (s_i = -1 \wedge r_i = -1) \\ k = 0 & otherwise \end{cases} \tag{9}$$

$$L_j = \sum_{i=1}^{j} k \quad \begin{cases} k = 1 & when\ (s_i = 1 \wedge r_i = 1) \\ k = 0 & otherwise \end{cases} \tag{10}$$

We see that the factors (parameters) P_j and L_j express the total number of favorable trades that were passed up. Other choices of factors are possible. For instance, they might depend on prices being low in summer and high in spring. However, we leave such refinements to future study.

The overall satisfaction and actual gain or loss is based on a predefined payoff matrix. This matrix represents the reward for each of the possible pairs of response and stimulus.

B. Advanced Case: Two Players

The advanced case (a two investor game) is a generalization of the basic case (a player versus the environment). For clarity, we require that strategies are different for each player.

We shall employ a new strategy somewhat similar to the DSF, called DS-function (DSFn). It is so named because instead of using a signum function to calculate the response, we use a function of the form α sin(accumulated affect)+ βP + τL. Then the response is equal to the signum function of the previous function specified as follows,

$$\text{DSFn:}\quad r_{jk} = -\text{sgn}(\alpha \sin \alpha_{jk} + \beta(P_{jk} + L_{jk})) \tag{11}$$

where k=1,2 indexes the players and α and β are parameters to be specified.

C. Simulation and Results

In the following figures, the abscissa denotes the trial number.

Basic Case: player vs. environment

We employ payoff matrix shown in table 1.

Table 1

player \ stimulus	1	-1
1	-2	15
-1	15	-10

Interpretation: If the investor places an open order to buy and the price rises, we would say he is involved in a losing situation. Similarly, for an open order to sell and the price declines. This situation is captured in the payoff matrix by a negative value.

In Fig.1 we plot the accumulated Affect for 25 different trials each of 100 rounds of play four different strategies.

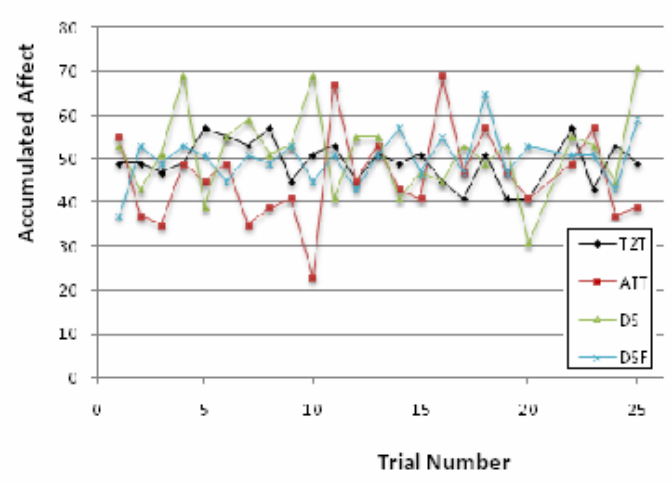


Fig.1 Accumulated Affect for trials of 100 rounds of play

From Fig.1, we see the risks associated with the ATT strategy. It yields highest satisfaction (maximum value of accumulated affect) or the least satisfaction (minimum value of accumulated affect). So, it can't be a winning strategy for the basic case.

Table 2 displays some statistics for the previous simulation. The value of satisfaction has a minimum equal to 23. Deviations of the maximum values on the other hand, are not significant. The better averages of accumulated affect were delivered by the developed strategies.

Table 2

Strategy \ ª	Min	Max	Avg
Tit-for-two-tat	41	57	49.25
Anti-tit-for-tat	23	69	45.8
DS	31	69	51.5
DS - 2 factors	39	59	48
DS – 3 factors	37	65	50.25

In Fig.2 we plot the accumulated Affect of 20 trials each of 500 rounds of play.

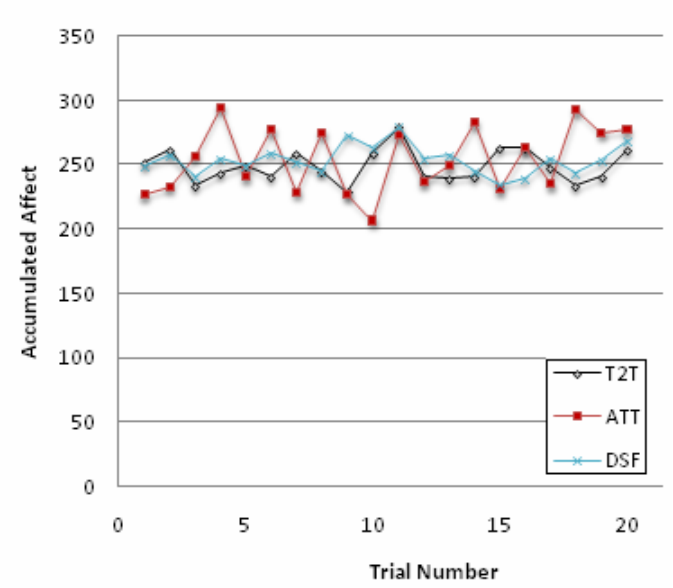


Fig.2 Accumulated Affect for trials of 500 rounds of play

In Fig.2 we increase the trial size to 500 rounds of play. The ATT strategy still produces to the minimum and maximum values of accumulated affect (risky). The DSF strategy is stable in so it maintains the accumulated affect in a high range with minimal variation. To illustrate these behaviors, we plot the averaged variance of accumulated affect of the strategies in Fig.3.

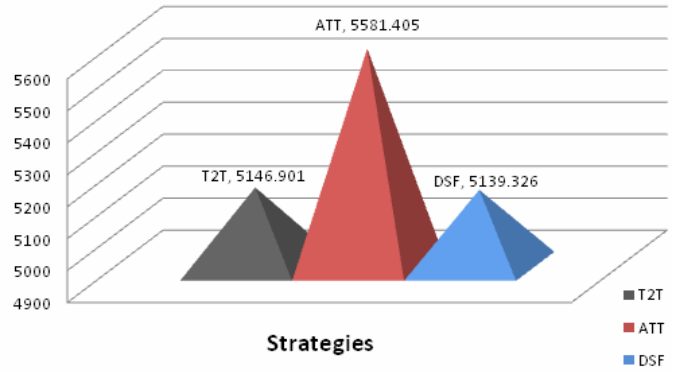


Fig.3 Average variance of accumulated affect for 20 trials, each of 500 rounds of play.

In Fig.4 we plot the sum of payoff of each of 20 trials each with 100 rounds of play

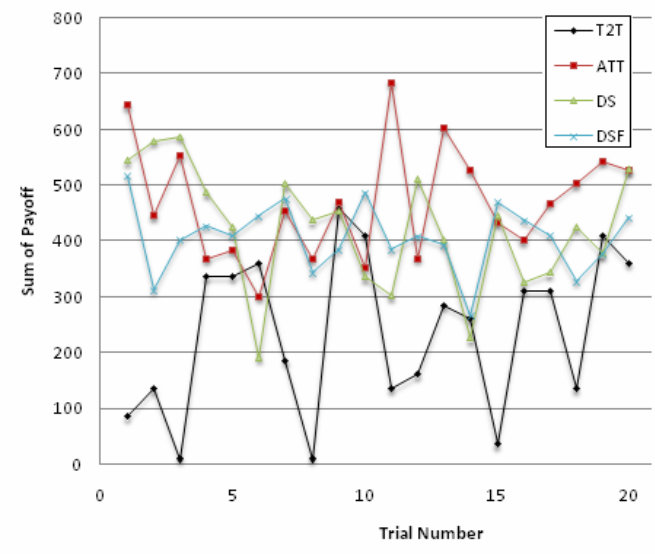


Fig.4 Actual accumulated payoff

Fig.4 induces us to discard the T2T strategy for the remainder of the basic case simulation, because it displays the worst financial performance. Notice, also that the DSF strategy produced reasonable performance, where the accumulated payoff is defined

$$\Pi_j = \sum_{i=1}^{j} p(r_{j-1}, s_j) \qquad (12)$$

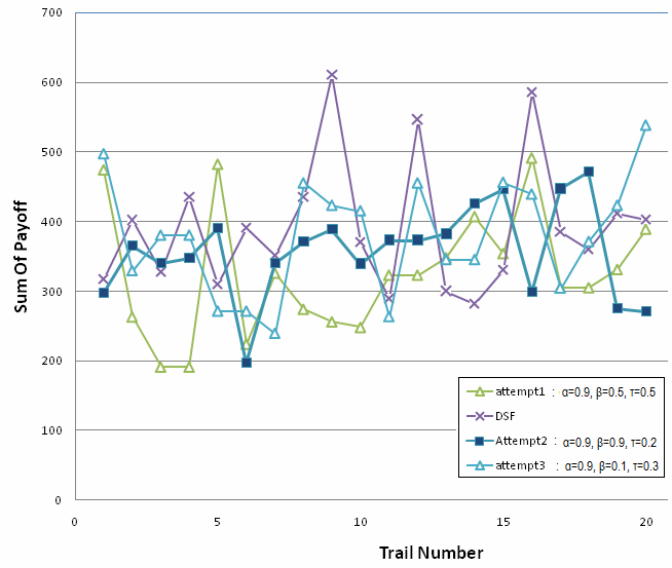


Fig.5 Accumulated payoff for 20 trials each is 500 rounds of play

In Fig.5, we display results comparing the DSF and DSFn strategies. Runs were made for different values for α,β and τ falling in the range [-0.5, 2]. Parameters α, β and τ appearing in (11), were chosen arbitrarily, although some rule of thumb calculations were employed to inform the choices.

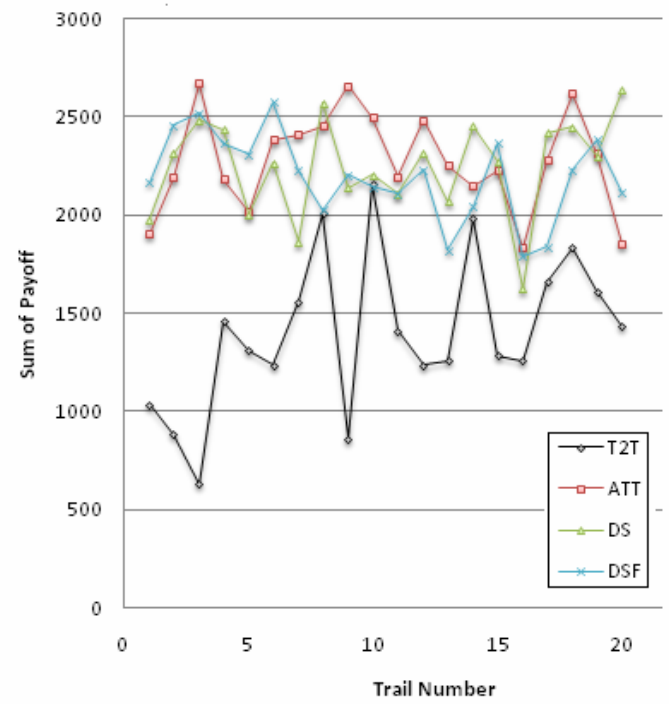


Fig.6 Accumulated payoff for 20 trials each with 500 rounds of play

Fig.6 is an extension of Fig.5 obtained by increasing the trial size from 100 to 500. The results are consistent with those represented in Fig.4, demonstrating the adequacy of the simulation.

The choice of θ for the DSF strategy is also based on numerical experiments. The value θ=0.4 gave highest affect values. Fig.7 illustrates this point.

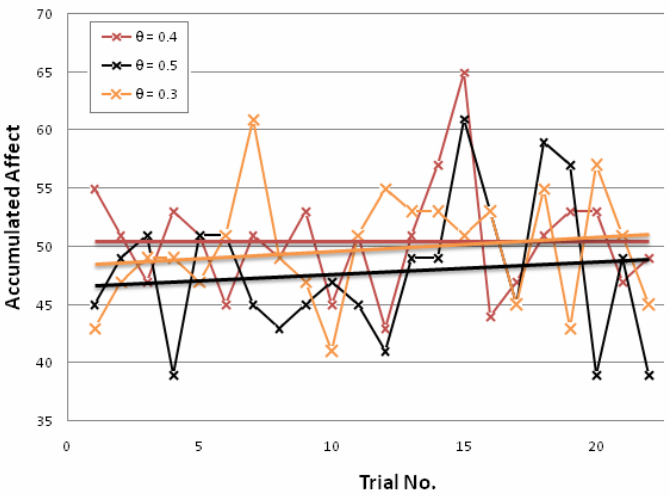


Fig.7 Result of adjusting θ on accumulated affect. The linear plots are least square fits.

Advanced Case: Two-player game:

Table 3 The payoff matrix

player1 \ player2	1	-1
1	-5	15
-1	20	-10

Table 3 is the payoff matrix for both players in a two-player game. There is a common payoff matrix to ensure that the competition between players is fair.

The following Tables 4 and 5 show clearly why we find the DSFn strategy to be superior in a two-player game.

Table 4 First competition between T2T and ATT

Strategies	Payoff	Accumulated Affect
T2T	2480	499
ATT	1260	-1

Table 4 shows that T2T is superior to ATT, because it produced larger payoff and greater satisfaction.

Table 5 Second competition between T2T and DSF

Strategies	Payoff	Accumulated Affect
T2T	2450	499
DSF	1235	497

From table 5 we see that T2T is also superior to DSF. This motivated using the DSFn. The results are striking as table 6 shows.

From table 6 we leave it to reader to decide which strategy is better. Yet, there might be some suspicion about the performance of the DSFn against strategies other than T2T. So we also show the results in Table 7 for additional confirmation.

Table 6 Third competition between ATT and DSFn

Strategies	Payoff	Accumulated Affect
ATT	3510	179
DSFn	5180	499

Table 7 Fourth competition between DSF and DSFn

Strategies	Payoff	Accumulated Affect
DSF	7385	497
DSFn	9830	499

Again, both table 6 and 7 confirms results shown in Tables 4 and 5.

Figure 8 exhibits a comparison of the averaged sum of payoff for the strategies of interest. Correspondingly, Figure 9 exhibits the average accumulated affect.

Since there are 2 players and 4 strategies, each strategy is employed 6 times by a player against his opponent. We average the accumulated payoff (Fig.8) and accumulated affect over each such 6 competitions.

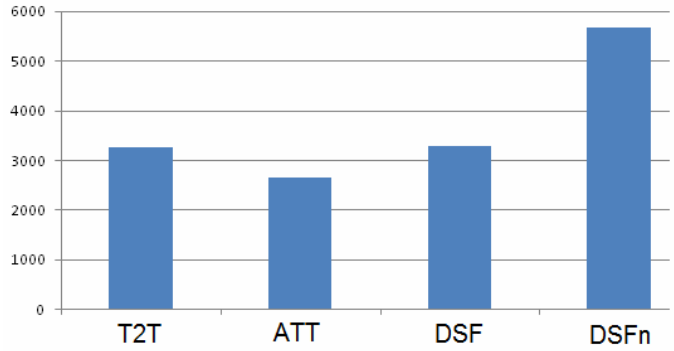


Fig.8 Accumulated payoff averaged over 6 competitions for each strategy

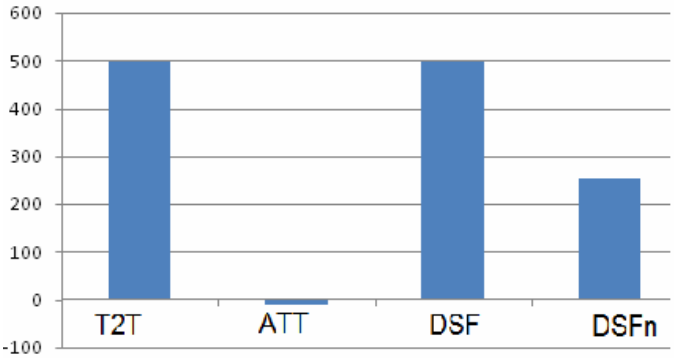


Fig.9 Accumulated affect averaged over 6 competitions for each strategy

Notice that the investor adopting DSFn wasn't the most satisfied even though his performance was best. The reduction of satisfaction of DSFn is mainly due to which player had the first play. Reversely, T2T had the great satisfaction, but not the greatest payoff.

III. APPENDIX: COMMENTS ON CONSCIOUSNESS

We propose a rational representation of consciousness in the basic case. Suppose we take consciousness to be the ability to perceive the relationship between oneself and one's environment. More specifically we take consciousness as an appropriate function of the accumulated affect (satisfaction) and the effective stimulus. The later is taken to be

$$\beta_{j,N} = \operatorname{sgn} \sum_{i=j-N+1}^{j} s_j \quad , \tag{13}$$

the majority rule for a block of size N plays.

Hence, (the content of) consciousness at stage j of the player is taken to be

$$c_{n,m,N} = K(A_{m,n,N}, B_{n,m,N}) \quad (14)$$

Where $A_{n,m,N} = (\alpha_{n,N}, \alpha_{n+1,N}, \ldots, \alpha_{n+m,N})$, similarly $B_{n,m,N} = (\beta_{n,N}, \beta_{n+1,N}, \ldots, \beta_{n+m,N})$ s.t. N=500, n=10. A choice for the function K is the correlation between the two arguments, which in our simulations is 500 (1 whole trial). The correlation formula used is

$$Corr = \frac{m(A_{m,.,N}, B_{n,.,N}) - (A_{n,.,N}, e_n)(B_{n,.,N}, e_n)}{[\{N(A,A) - (A,e)^2\}, \{N(B,B) - (B,e)^2\}]^{1/2}} \quad (15)$$

Where again, n=10, N=500 and m=500,…,510.

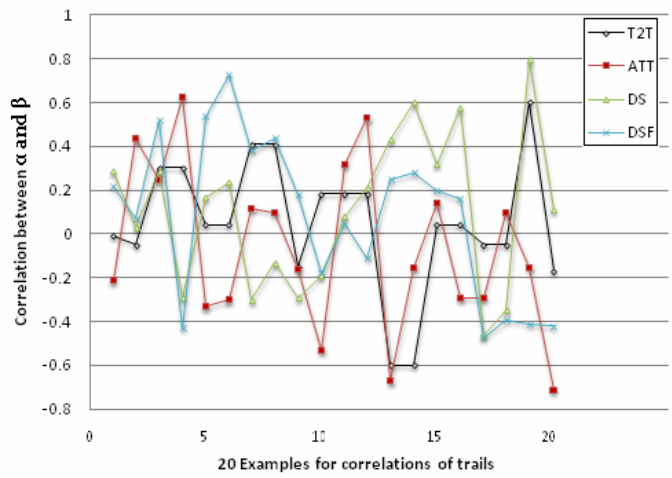


Fig.8 Correlations between α and β

Fig.8 shows the correlation values between10 pairs of α and β, each representing a trial of 500 rounds of play. α and β are vectors: α the values of accumulated affect of 10 trials, and β values of accumulated stimulus of the corresponding 10 trials. Fig.8 shows a repetition of 20 runs of this calculation.

Table 8 shows the correlation averaged over all 20 runs for the T2T, ATT and DSF strategies.

Table 8 Average of correlation over 20 runs

Strategy	Averaged Correlation
T2T	.052
ATT	-.058
DSF	.0805

REFERENCES

[1] Acquisition of General Adaptive Features by Evolution. Dan Ashloc1 and John E. http://www.public.iastate.edu/~jemayf/NLA.pdf
[2] Wikipedia. "Tit for tat"
[3] Wikipedia. "List of games in game theory"
[4] Lab 1 Repeated Play Prisoner's Dilemma. By Mike http://students.cs.byu.edu/~cs670ta/PDLab.html
[5] About.com. "Three Main Influences on Stock Prices". By Ken Little
[6] http://neuron.eng.wayne.edu/bpFunctionApprox/bpFunctionApprox.htm
[7] Wikipedia. "Affect (psychology)"
[8] WordNet Search. For definition of "strategy" and "satisfaction"
[9] "Correlation and Covariance of a Random Signal" by. Michael Haag http://cnx.org/content/m10673/latest/
[10] UNIX SYSTEM. For programming.

Constraint-Based Placement and Routing for FPGAs using Self-Organizing Maps

Michail Maniatakos

***Abstract*—Field-programmable gate arrays (FPGAs) are becoming increasingly popular due to low design times, easy testing and implementation procedures and low costs. FPGAs placement and routing are NP-complete problems dealt well with modern tools using heuristic algorithms. As modern FPGAs increase in size and also new capabilities, such as Run-Time Reconfiguration (RTR), are introduced, the complexity of these problems is greatly increased. In this paper we approach both problems using a modified version of Kohonen Self-Organizing map. The algorithm, consisting of four phases, takes into consideration constraints that may apply to the FPGA design (such as I/O pins, resource constraints like global clock etc). The modified algorithm yields a good topological map of the design to be placed, minimizing the average distance between connecting logic blocks.**

***Index Terms*—FPGA, self-organizing feature map, placement, routing, constraints**

I. Introduction

Field-programmable gate arrays (FPGAs) are semiconductor devices, which consist of programmable components called "logic blocks". These blocks can be programmed to perform different functions (such as AND, OR) or to store data. Logic blocks connect through wires running all over the FPGA board. Many connected logic blocks create an FPGA design that performs a specified operation. An FPGA board can be reprogrammed, while its main counterpart, Application Specific Integrated Circuits (ASICs) are manufactured for a specific application and their operation cannot change. The main disadvantage of using FPGAs compared to ASICs is that FPGAs are pre-manufactured so their cost increases linearly for every board, while ASICs have a huge initial cost but production cost for larger quantities increases slowly. Also, FPGAs are slower and more power consuming. On the other hand, an FPGA has no initial manufacturing cost, it has low recurring engineering costs and is significantly cheaper than ASICs for small quantities.

A user programs the board using High-level description languages (HDLs); the output code is converted by tools to logic blocks. The exact place that a logic block will be stored is defined through the "placement" procedure. Similarly, the wire tracks that will be used to connect logic blocks are defined through the "routing" process. Placement and routing are often interactive because good routing is highly dependent on good placement.

In this paper we approach placement and routing processes using a modified version of Kohonen's Self-Organizing Map algorithm defined in [1]. Specifically, we modify the notion of a winning neuron and which neurons are updated.

In Section 2 the Self-Organizing map algorithm is given, while in Section 3 the placement and routing process of an FPGA is described. In section 4 we introduce our approach on FPGA placement and routing using SOMs. Section 5 presents a case study where our algorithm is used to place a complex design on an FPGA board. Finally, in Section 6 performance figures are presented along with a discussion of the results of the algorithm.

II. Self-Organizing Map Algorithm

Self-organizing maps (SOMs) are a special class of artificial neural networks, based on both competitive and cooperative learning. The main purpose of the SOM is to transform an incoming signal pattern of arbitrary dimension into a one or two dimensional discrete map. Each neuron in the map is fully connected to all source nodes in the input layer [2]. SOM training is based on two basic principles:

Competition. The prototype vector that is most similar to an input data vector (where similarity can be defined in terms of Euclidean distance) "wins" the competition and is then transformed in order to be more "similar" to the input vector. By means of this process the algorithm learns the position of input data.

Cooperation: Besides the winning neuron, all its neighbors (where neighborhood radius must be defined by some parameter) are moved towards the input data vector as well. With cooperation, the map self-organizes.

A brief summary of the SOM algorithm as presented in [2] follows.

Let $[x_1, x_2, \ldots, x_m]^T$ be an input data vector, where m is the dimension of the input data space. The synaptic weight vector of each neuron has the same dimension as the input space and is denoted by $[w_{j1}, w_{j2}, \ldots, w_{j1m}]^T$ for neuron j. The algorithm consists by the following steps:

1. *Initialization*. Initialize weight vectors $w_j(0)$ with random values, while $w_j(0)$ must be different for every j = 1, 2,

Michail Maniatakos is with the Electrical Engineering Department, Yale University, USA (e-mail: michail.maniatakos@yale.edu)

… *l*, where *l* is the number of neurons.

2. *Sampling*. Choose a sample **x** from the input data set. This choice is performed stochastically.
3. *Similarity matching*. Using the Euclidean distance, determine the winning neuron i(x) at the specified time n:

$$i(x) = \arg\min_j \|x(n) - w_j\|, j = 1,2,...,l \quad (1)$$

4. *Updating*. Update the synaptic weight vector of the winning neuron and its nearest neighbors using the formula

$$w_j(n+1) = w_j(n) + \eta(n)h_{j,i}(n)(x(n) - w_j(n)) \quad (2)$$

where the learning rate function η(n) decreases over time and $h_{j,i}(n)$ is a monotonic function that defines each neuron's topological neighborhood. A common choice for this function is the Gaussian function

$$h_{j,i(x)}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), n = 0,1,2,... \quad (3)$$

where $d_{j,i}$ is the distance between the winning neuron i and neuron j, and σ(n) is

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau}\right) \quad (4)$$

where σ_0 is the initial value of and τ is a time constant.

5. *Continuation*. These steps are repeated for a specified number of iterations or until no noticeable changes in the feature map are observed.

III. FPGA PLACEMENT AND ROUTING

An FPGA board consists of several logic blocks. In a simple FPGA design, a logic block contains a Look-Up Table (LUT) and a flip-flop, so it could either perform a specific function defined in the LUT or store a single bit. A LUT contains the truth table of the function implemented. The output of a logic block goes to the input of another logic block through wiring tracks, unless Input/Output connections are specified (usually at the edges of the board). A simple FPGA board layout is shown on Figure 1. This layout style is called island-style architecture, a very popular approach (followed by Xilinx [3]).

A. Typical Application Synthesis Flow

In order to program an FPGA board, a user uses a Hardware Description Language (HDL), such as Verilog or VHDL. This design is transferred to the FPGA board, using the following three step procedure [4]:

1. *Technology Mapping*. In this stage, the functionality of primitive logic gates is restructured into sets of logic blocks. For example, an AND gate might be converted to one logic block, programming the logic block's LUT to have the truth table of the AND operation. Of course a logic gate may need to spread along several Logic blocks (a 5 gate NAND for example).

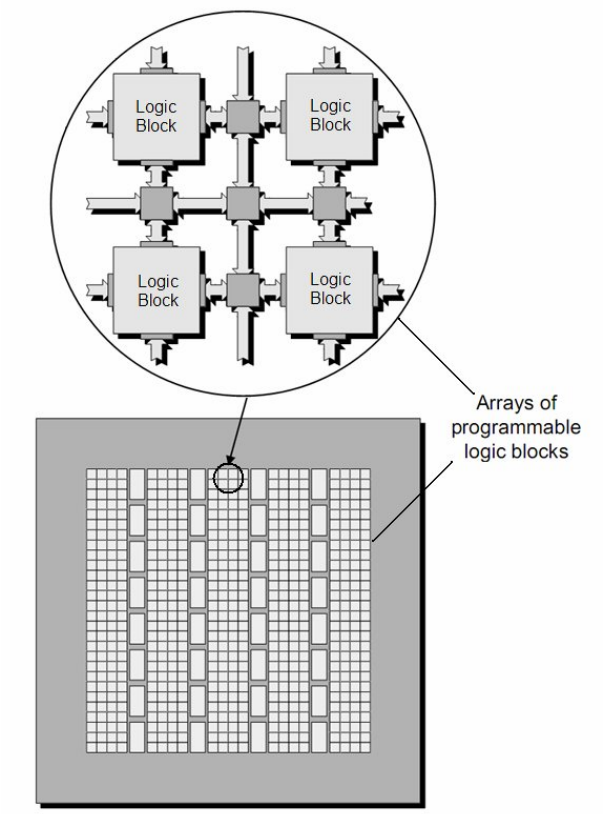


Fig 1. FPGA board structure showing Logic Blocks and interconnections

2. *Placement*. In this step, all packed blocks of logic have to be assigned to specific block locations in the prefabricated two-dimensional array of the FPGA board. Ideally, perfecting localized routability in each subsection of the board would yield the best placement, but given the distributed nature of interconnect and dependencies caused by segmentation this approach becomes infeasible. So a metric to evaluate an algorithm performance is the wiring length of the placement. Placement is an NP-complete problem.

3. *Routing*. After placing the logic blocks in specific places, these blocks must be connected using routing segments and switches to create a connecting path through FPGA's routing tracks. This is likewise an NP-complete problem, because an FPGA has a limited number of wiring tracks running around the board. The most commonly used algorithms for routing are the

maze-routing algorithms as presented in [5]. These algorithms are based on Dijkstra's shortest path algorithm. The placement and routing processes must not be separated; a specific placement may not be routable at all, while a slightly different one may yield a good quality routing.

B. Constraints

An FPGA design usually has specific constraints to operate correctly under given specifications. A straightforward example of a constraint is a logic block that receives its input from outside the board, so it has to be placed on the I/O blocks (usually at the edges of the board). So the route and place algorithms must take into consideration the special nature of this logic block. Another example of a constraint comes from switch pins that exist in specific places on the board where the corresponding logic block must be placed appropriately to receive the switch state.

Besides such constraints, there are some constraints that affect design's performance and not design correctness. For example, in an FPGA design all clocked elements (such as Flip-Flops) share the same global clock; so the clock signal must arrive at clocked elements simultaneously and as fast as possible (for better performance). Thus, these elements must be placed near the clock buffers that produce the clock signal.

IV. MODIFIED SOM ALGORITHM

We devise an algorithm to achieve good placement and routing on the FPGA board. Good placement yields good routing and vice versa.

We modified part of the Kohonen SOM algorithm in order to handle possible constraints of the FPGA design. The lattice we use for the SOM algorithm corresponds to the real layout of the design we want to place and route. So, a neuron in the lattice represents a logic block that has to be placed in a logic block of the FPGA, while synapses represent the connections between logic blocks (a logic block can have up to four connections). For example, for the simple design shown in Fig. 2, the lattice that would be produced is show in Fig. 3.

Constraints that can be applied in the algorithm fall in two different categories:

Strict constraints: Constraints in this category are defined as conditions that *must* hold in order for the design to work properly. For example, a specific logic block must be placed on a specific block (e.g. an I/O pin).

Relaxed constraints: Violation of constraints in this category will not affect the correctness of the design but its performance. The modified algorithm may violate some relaxed constrains in order to achieve a better placement and routing. For example a logic block should be placed as near as possible to an FPGA resource. The final distance between the logic block and the resource affects the performance of the design (e.g. if the logic block is placed too far, the design will be slower). Generally, we can regard these types of constraint as *resource race constraints*, because multiple elements have to be placed close to a specific resource.

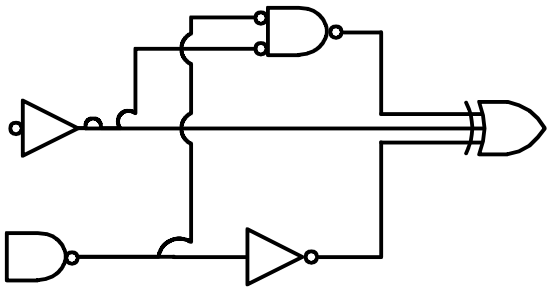


Fig. 2. Simple design to be placed

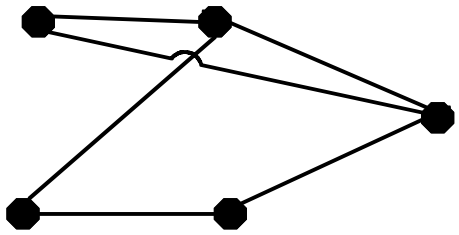


Fig. 3. Lattice for sample circuit of Fig. 1

A constraint has an *importance* property, graded from 1-10, where 10 implies great importance. Strict constrains are automatically assigned a value of 10. This property helps the algorithm evaluate constraint importance and drives the algorithm to prioritize the constraints to be considered, optimizing the quality of the final placement.

The ultimate goal of this algorithm is to minimize distances between interconnecting logic blocks, so as to maximize design performance.

A. Initialization of the algorithm

We first define the lattice of the circuit to be placed (as described in Fig. 3). A logic block (LB) is represented by a neuron and synapses between neurons represent the connections between the logic blocks. So if the design has M logic blocks to be placed we have the following set of neurons in the lattice labeled arbitrarily 1, …, M.

These M neurons connect through the MxM connection matrix C where

$$C_{ij} = \begin{cases} 1, if\ LBs\ i\ and\ j\ are\ connected \\ 0, otherwise \end{cases} \quad (5)$$

A strict constraint is defined by three values [s_1, s_2, s_3], where s_1 and s_2 define the coordinates where the logic block must be placed, and s_3 is the index of the logic block that has to be placed in the position [s_1, s_2] on the FPGA board. For example, the set [1, 1, 5] specifies that the logic block 5 has to

be placed in the [1, 1] position of the FPGA board.

Similarly, a relaxed constraint is defined by two values [c_1, c_2], where c_1 is the index of the logic block and c_2 is the index of the resource. For example, the set [2, 3] implies that logic block 2 must be placed close to resource 3.

A resource is defined by three values [r_1, r_2, r_3], where r_1 and r_2 are the X-Y coordinates of the resource and r_3 specifies its importance property.

Finally, N denotes the dimension of the squared FPGA board where the logic blocks must be placed (for example for N=20 we have a board with 20x20 = 400 positions for logic blocks).

Initialization of the algorithm consists of placing the lattice randomly on the FPGA board, employing a uniform distribution. The algorithm itself consists of four phases. A brief description of each phase follows:

Phase 1 (Constraints set): In this phase, we satisfy the strict constraints by placing the neurons (logic blocks) appropriately. These neurons will remain fixed as the lattice self-organizes.

Phase 2 (Resource Competition): During resource race phase 2, we restrict the set of potential winning neurons to the subset of neurons that compete for a specific resource. This phase is repeated for every resource.

Phase 3 (Ordering and Convergence): In the 3rd phase the lattice self-organizes with no further restrictions, except that constrained neurons are neither allowed to win nor be updated.

Phase 4 (Quantization): Because the coordinates of the neurons won't have integer values, we quantize the coordinate matrix..

An analytical description of the above phases follows.

B. Phase 1: Constraints set

During this phase we place the strict constrained logic blocks in the exact coordinates defined by design constraints. During following phases, these neurons are not involved in the competitive or cooperative process of the self-organizing map (they remain fixed throughout the whole process of ordering and convergence).

In case of a conflict, we move the logic block to the nearest unoccupied block.

C. Phase 2 (Resource Competition)

In the second phase the algorithm considers the relaxed constrains set by the FPGA design. During this phase, the algorithm's effort is to move the logic blocks (neurons) close to the resources in order to optimize placement quality.

An SOM self-organizes based on input vectors, so we have to generate these vectors. For each resource, one input vector is generated. For example, if we have a resource in [2, 2] position on the board, then a two-dimensional input vector is generated in the same position ([2, 2]). Fig. 4 presents the feature map using a resource placed at the bottom left of the FPGA board.

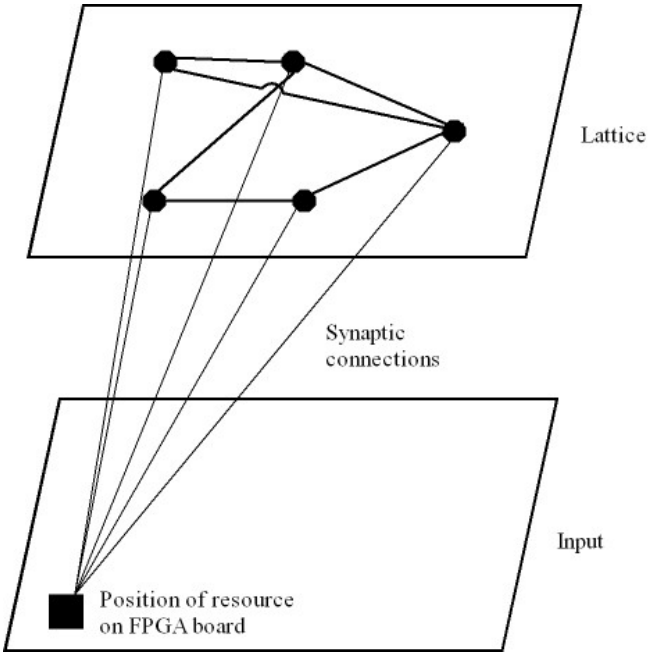


Fig. 4. Self-organized feature space

Using this method, we drive winning neurons closer to resources. Only constrained neurons are allowed to win, so they will gradually move closer to the resources. The probability that an input vector will be selected during sampling is directly proportional to the importance property defined for each resource. So the modified algorithm for this phase is the following:

1. *Initialization*. The values of the initial weight vectors are copied from the weight vectors of phase 1 (e.g. we continue using the same feature map)
2. *Sampling*. We choose a sample x from the input space, with probability directly proportional to the importance of this vector. Thus, more important resources will be sampled more often than others.
3. *Similarity matching*. We find the best-matching neuron i(x), where i is a neuron that competes for the resource. Therefore, only racing neurons are allowed to win. The best matching criterion is Euclidean distance:

$$i(x) = \arg\min_k \|x(n) - w_k\|, i, k \in R_L \qquad (6)$$

where R_L is the subset on neurons that compete for Resource L.

4. *Updating*. We adjust the synaptic weight vectors of all neurons using the following formula:

$$w_j(n+1) = w_j(n) + \eta_1(n) h_1(x(n) - w_j(n)) \qquad (7)$$

where learning rate parameter $\eta_1(n)$ gradually decreases over time, and h_1 is the neighborhood constant; so neighborhood radius remains constant throughout the algorithm. We set the neighborhood radius to a constant small value to prevent resource competition from modifying a large portion of the lattice, allowing only competing neurons and some of their neighbors to move towards the resources.

5. *Continuation*. These steps are repeated until no noticeable changes in the map are observed.

At the end of this phase the constrained neurons will be closer to the desired resources, while the rest of the map will still be unordered. The synaptic weights of these neurons, like strict constrained neurons presented in Phase 1, won't be updated during subsequent phases. Phase 2 is repeated for every resource defined in the FPGA design.

D. Phase 3 (Ordering and Convergence)

During this phase, placement is finalized using the Kohonen SOM algorithm presented in Section II. Again, the input data vectors must be defined. Similarly to Phase 2, one input vector is created for every resource and for every strict constrained neuron (placed during Phase 1). So the final input vector set consists of two dimensional vectors that represent the position of the resources and strict constrained neurons.

When this phase completes, a good geometric approximation of the design to be placed will be produced. The result is not guaranteed to be optimal; placement and routing are NP-complete problems.

Also, due to the mathematical nature of the SOM algorithm, logic blocks' coordinates will have real values; this is not allowed, because logic blocks should be placed on distinct logic block places. This misalignment defect is targeted in the next and final phase.

E. Phase 4 (Quantization)

During this final phase logic blocks are moved to the nearest logic block location. If during this quantization phase more than one logic block is to be placed in a single location, only the nearest logic block is allowed to move there; in case of a tie, the first in the list is moved there. Then the rest of the blocks are placed in the nearest unoccupied block using Euclidean distance.

After this final phase we get a design that fulfills the constraints of the FPGA design while achieving good placement and routing. In most of the cases the modified SOM algorithm manages to reduce the distances of the initial random placement up to 10 times. A specific example of using the algorithm follows.

V. CASE STUDY

In this section the modified SOM algorithm is demonstrated. Assuming we have an FPGA board of 10x10 logic block locations, we will attempt to place and route 30 randomly connected logic blocks (keeping the limit of up to 4 connections though). Every phase of the algorithm is presented in Figure 5.

We add three strict constraints in our case study:

```
[ 1  4  3]
[10  2 26]
[ 5  5 15]
```

The first two values are the X-Y coordinates of position that the logic block must be placed in, while the third value specifies which logic block will be placed there.

We also add one resource located in four different places with different importance properties:

```
[ 2  2 10]
[ 2  7 10]
[ 7  2  5]
[ 7  7  5]
```

In the above matrix the first two values in a row specify the X and Y coordinate of the resource while the third specifies its importance.

The logic blocks that will compete for this resource are defined in the following matrix:

```
[10 22 8 20]
```

The above matrix specifies that the logic blocks with index 10, 22, 8 and 20 will compete for the resource.

During initialization of the algorithm, we randomly place the logic blocks on the board. The initial summed distance between all logic blocks is 1345 units.

In phase 1 the strict constrained logic blocks are moved to their specified positions. In this case study the summed distance is slightly increased, but it could also be slightly decreased or remain the same. The constrained logic blocks are represented with red dots.

Phase 2 is the resource competition phase. Blue-dotted logic blocks compete for the blue-crossed resources. Even though they are not extremely close to the resource, they are pretty close, and they are evenly spaced among the resources. The summed distance in this stage was decreased to 273.44 units, which is very good considering the fact that the algorithm actually hasn't started final ordering and convergence.

In phase 3 the final ordering and convergence of the algorithm is performed. After 600 iterations the map converged to the lattice shown in Fig. 5. The distance is much smaller (126.04) compared to the initial one (1345.00), which ensures us that the algorithm has greatly improved the placement and routing of the design.

During the 4^{th} and final phase, we quantize the final

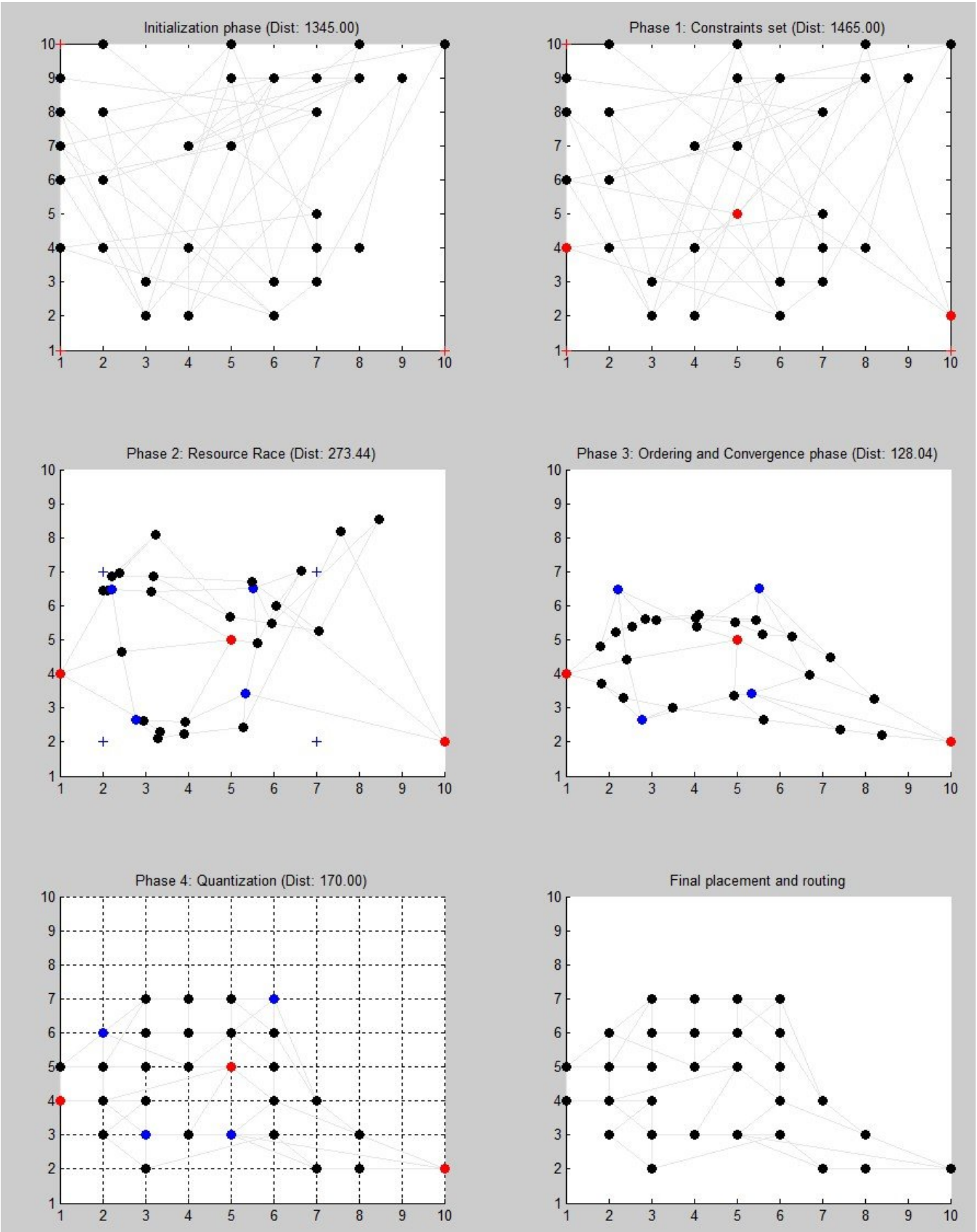


Fig. 5. Case study for a 10x10 FPGA with 30 logic blocks to be placed

positions of the logic blocks. This quantization yields increased distances (170 units) This is expected because close to resources there are more logic blocks, so conflicts will occur because of limited logic block locations. While conflicting logic blocks move to nearby places, summed distance increases.

This final placement shown in Fig. 5 can be considered good placement, because there aren't many crosses between routing channels while the summed distance is about ten times smaller than initially.

VI. Results and Discussion

A. *Performance Evaluation*

In this section we will evaluate algorithm performance. Generally, the best measurement for placement's quality is the summed distance of logic blocks. We could also use other measurements, such as the mean quantization and the topological error.

We first focus on the performance of the algorithm for different iterations, using summed distance as our quality measurement. The summed distance of logic blocks in each phase will be calculated. By performing 20 different runs of the algorithm we get the distances presented in Figure 6.

The first conclusion from this figure is that the algorithm exhibits similar performance for each run: great improvement during Phase 2, further improvement during final ordering and convergence during Phase 3, while during the Quantization Phase 4 summed distance are slightly increased. Also, Phase 3 has the least deviation compared to all other phases; so algorithm performance seems to be deterministic.

Another important conclusion is that the final placement *is independent of the initial random placement*; so no matter how good or bad the initial placement is, algorithm performance isn't affected.

Next consider evaluation of the average performance of the algorithm for many runs. We perform 100 iterations of the algorithm and calculate the average value of summed distance between phases. The results shown in Fig. 6 are clear; all iterations exhibit the same behavior as described previously.

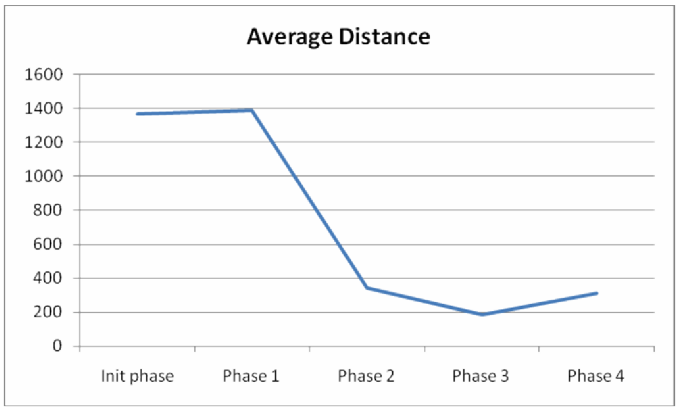


Fig. 7. Average distance between phases for 100 runs

Finally, we calculate the average quantization and topological error of the algorithm in each phase. The results are shown in Fig. 7.

By carefully examining the graph, it is obvious that the average topological error follows the average summed distance curve. This was expected because topological error is related to distances between neurons and possible crosses between their synapses.

However, average quantization's curve is slightly different than the previous curves. During the transition from initialization phase to phase 1, quantization error is improved. Since input data vectors are generated around constrained

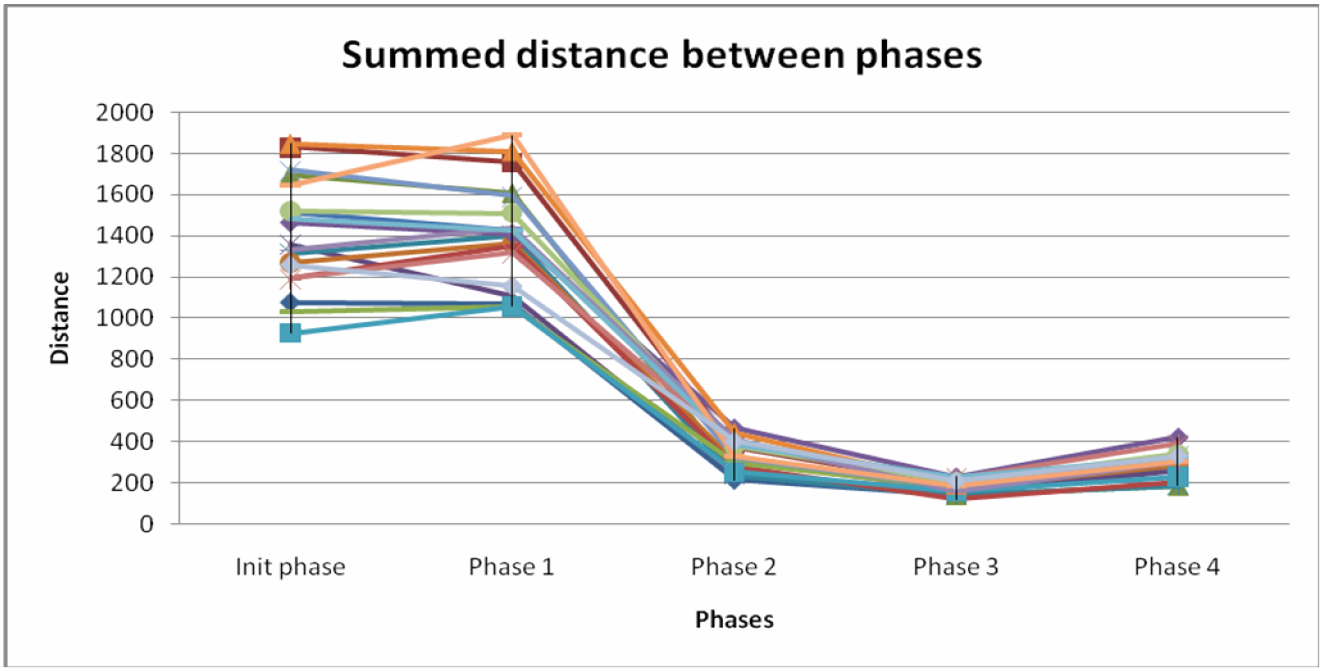


Fig. 6. Summed distance between all phases of the algorithm for 20 runs

neurons, this improvement was expected; when we move neurons during phase 1 to their constrained positions, we move them closer to the input data, so the quantization error decreases. The same explanation covers Phase 1 to Phase 2 improvements. However, during phase 3, quantization error is increased. We can attribute this increase to the fact that during final ordering and convergence neuron distribution is denser close to the important resources and sparser close to less important resources. After final quantization, neurons are placed at discrete locations, decreasing once more the quantization error.

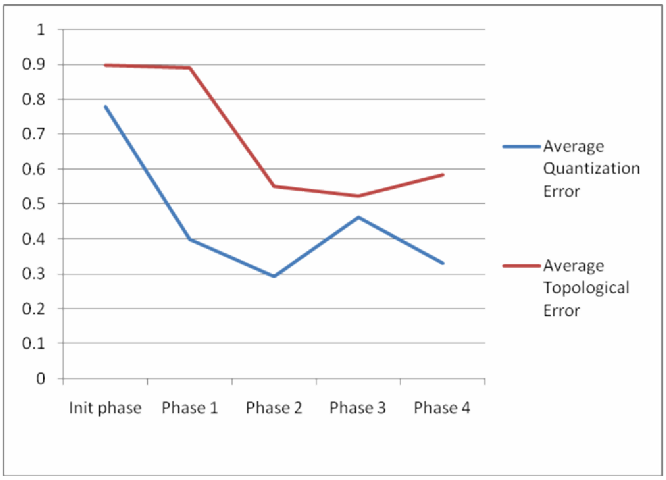


Fig. 8. Average Quantization and Topological error for 100 runs

B. *Discussion*

The proposed methodology is an approach to placement and routing problems using a self-organizing neural network. An FPGA design can be viewed as a lattice that has to be organized optimally.

The modified algorithm produces a good topological approach of the design to be placed; it decreases distances between logic blocks and avoids intersecting wires by decreasing the topology error. It is possible that this layout may be fine-tuned locally (for example by using single logic block swaps and recalculating distances) but the overall design will adhere to the same topology.

We should also mention that the algorithm is slow, and for large designs it would probably require a large number of iterations to converge. Also the resource race phase results are not deterministic; so we can't predict with certainty the specifications of the placed layout. However, exploring all possible parameters for each training phase of the algorithm could yield greater certainty, trading-off an optimized placement.

REFERENCES

[1] Kohonen T., The Self Organizing Map, Proceedings of the IEEE Vol. 78 No. 9 (1990) pp. 1464-1480
[2] S. Haykin, Neural Networks: A comprehensive foundation (Upper Saddle River, NJ: Prentice Hall, 1999)
[3] Field-Programmable Gate Arrays Data Book, Xilinx Corporation, 1996
[4] Tessier R., Fast Place and Route Approaches for FPGAs, PhD thesis, MIT, 1999
[5] C. Lee, An algorithm for path Connections and its Applications, IRE Transactions on Electronic Computers, Sept 1961

Neural Network based Web Search Query Result Summarizer

Mohd Fahadullah

Abstract

This paper proposes an Ensemble Neural Network to create text summary of web search query results. A search engine returns the result to a web search query by matching the keywords entered by a user to the keywords contained in a web page. Thus, the results might not always be relevant to the user. Here, we propose a neural network model to generate text summary for each result to allow user to find the content without going through the whole webpage. Text summary is generated by extracting all those sentences which are of the most significance in the text. Each sentence in the text is encoded as a set of features and presented as an input to an ensemble neural network. The network is trained using the set of randomly selected texts from the internet. The network output is then used to classify a sentence as a summary sentence.

***Index Terms*—Ensemble Neural Network, Part of speech tagger, Search Query, Summarizer.**

1. Introduction

With so much data finding its way to the Internet, it almost seems impossible for a web user to find and utilize information relevant to his/her use. Search engines have made it possible for a web user to search for information that a user wants nevertheless a user is required to find the relevant information from the plethora of search query results returned by the search engine. A search engines response to a user search query consists of a list of matching web page titles and their links to actual sites hosting them. However it is still required on the part of the user to go through each search result to see which results are relevant. We propose a solution to this problem by generating an automatic summary for each search result. Then, a user can see whether a particular search result is relevant just by reading the summary.

The method for extracting the summary of a search query result returned by a search engine uses an ensemble of neural network to extract sentences from the search query results.

The neural network employed is explained in depth in Section II and discussion and future work are given in Section III. Words *text* and *document* are used interchangeably throughout and they both refer to a web page.

2. Neural Network based Web Search Query Result Summarizer

2.1 Introduction and Motivation

Text Summarization is an information retrieval method to automatically generate a summary of text. The technique has been studied for decades but today with the Internet and the World Wide Web it has become much more important. Most text summarizers are extraction based, i.e. they generate summary by extracting sentences from the text. Some summarization algorithms are capable of producing summaries that contain not only sentences that are present in the document but also new automatically constructed phrases that are added to the summary to make the latter more intelligible. In theory, this functionality makes the summarization algorithm more powerful by improving the comprehensibility of the output summary. In practice, the automatic construction of phrases is a quite difficult task and there is no guarantee that the new phrases will be really meaningful for the user. In this paper, we discuss the neural network to generate summary by extraction. We propose a neural network model which is an extension of the neural network proposed in [5]. We provide a method to extract all those sentences in the document which are related to the user specified search query. We tag the words in the sentence and the query with the part of speech and then compare them to extract related sentences. This help a user finding all those results which are related to his/her query.

Ensemble neural networks are the neural networks which consist of a cluster of neural networks each having the same form. Ensemble methods combine the outputs of several neural networks [2]. The output of an ensemble is a weighted average of the output of each network, with the ensemble weights determined as a function of the relative error of each network determined in training [2]. Individual networks in the ensemble are trained employing the same data but their synaptic weights are initialized with different values. In several papers and previous work it has been shown that the network ensemble has a generalization error generally smaller than

that obtained with a single network and moreover that the variance of the ensemble is less than the variance of a single network. Thus, ensemble neural network provide better generalization in classification problems than a single network. The output of a Basic ensemble method [2] consists of output of all the networks weighted equally.

$$F(n) = \frac{1}{M}\sum_{i=1}^{M} F_i(n)$$

where M is the number of the individual neural networks in the ensemble, $F_i(n)$ is the output of network i on the n^{th} training pattern, and $F(n)$ is the output of the ensemble on the nth training pattern.

2.2 Features

We extract a set of features for each sentence in the text. These features sets are extracted to represent the relevant information from each sentence without losing the overall meaning and context of the sentence in the document. These features are classified as the position of a sentence in the document like it's the first sentence or the last sentence in the document. It has been shown that summaries consisting of leading sentences outperform most other methods in this domain [7], and that sentences located at the beginning and end of paragraphs are likely to be good summary sentences [6]. The number of unique words in the sentence which are the non-stop and non-repetitive words in the sentence, the number of unique words that are above a threshold weight value which is determined by the weight of the words, the number of words that match the title of the document, the number of unique words that match the title of the document, the number of all the words that match the title and have the same part of speech as the words in the title, the weight of the sentence which is the measure of its closeness with other sentences based on number of words a sentence shares with other sentences, the number of unique words that match the search query, the number of all the words that match the search query and have the same part of speech as the words in the search query and the total sum of the weights of all the unique words in the sentence.

Words in a sentence are compared to the search query words in order to include those sentences in the summary which are more related to the user specified query words and hence more relevant. In case there isn't any sentence which matches to the user specified keywords, the document can be classified as not relevant to the user. This can be used to provide user a recommendation for the document to be relevant. These features are encoded as a vector and each sentence is assigned the vector of its extracted feature values.

2.3 Preprocessing

Preprocessing consists of a number of operations mainly stop word removal, case folding, unique word extraction and small sentence removal. Stop words are words like 'do', 'is' and 'will' etc. and have minor effect on the meaning of the text. Case folding consists of converting all the characters of a document into the same case format, either the upper-case or the lower-case format. Unique words extraction consists of finding all the non-stop and non-repetitive words in each sentence. Each unique word, w, in the document is assigned a weight, $weight_w$, which is calculated by its frequency in the document as follows -

$$weight_w = tF_w \times \log_{10}\left(\frac{S_{total}}{S_w}\right)$$

where tF_w is the frequency of word w in the document, S_{total} is the total number of sentences in the document and S_w is the number of sentences containing w. Thus, words that are highly frequent in few sentences have larger weights than the words that are highly frequent in many sentences. We associate a weight with each sentence of the document which is the measure of its closeness with all the other sentences in the document and its value is inversely proportional to the closeness. The weight of a sentence is the weighted average of its closeness with all the other sentences. Closeness of a sentence is defined as the number of unique words it shares with another sentence. Sentence which shares larger weight words with another sentence is more closure to it than the sentence to which it shares smaller weight words as they contain the same words and might convey the same meaning. Words in a sentence are tagged with the part of speech through a part of speech tagger and compared to the search query and the title of the document. We used a log-linear part-of-speech tagger available at *The Stanford Natural Language Processing Group* licensed under the GNU GPL Number of words matched and the corresponding weights of the words matched are used to find how close each sentence is to the search query and the title. We keep a lower limit on the number of words in a sentence and sentences with words below the threshold value are eliminated.

2.4 Implementation

Each Sentence in the document is mapped as a vector of nine features defined in [B]. Values in the feature vector are row normalized to a unit vector. Neural

networks in the ensemble are implemented as three layer Feed-Forward neural network. Each Neural network consists of nine input units, twenty-one computational units in the hidden layer and one output neuron. The output of the ensemble neural network is the weighted average of the output of all neural networks in the ensemble. The feature vector of a sentence is applied to all the networks in the ensemble as an input and the output of the ensemble determines whether the sentence is to be included in the summary or not. The output of the ensemble for a sentence is in the range of [0-1], where a value close to 1 means the sentence is to be included in the summary and a value close to 0 means the sentence is not to be included in the summary. The number of sentences in the summary is taken as ten percent of the total number of sentences. Thus, we select ten percent of the total sentences for which ensemble output value is close to 1. Neural network is implemented in the Matlab Neural network toolkit.

2.5 Training

The neural network is trained on a set of text documents with Back-Propagation training algorithm. The supervised learning is performed using the summary of the training data set as the target values. For each document in the training set, feature vectors are computed for each sentence and applied to the ensemble neural network with the desired target output as 1 if that sentence is included in the summary or as 0 if it's not. In the neural network training, the keywords of the document are matched with the unique words in the sentences and the part of speech tagged words in the sentences. Thus we make use of keywords in the same way as we make use of the search query. Training the neural network on keywords makes it more robust since neural network learns to associate with the words that describe the context of the document and would be of interest to a user.

The main difficulty with training of neural network was the unavailability of any large text corpus with extraction based summaries. The dataset used to train and test the ensemble neural network consisted of 50 random web pages selected from *http://directory.google.com/* from different categories. Summary and keywords for each of the web pages was created by hand. The data set was divided between 35 documents in training set and 15 in test set. Each document consisted of 16 to 143 sentences with an average of 46 sentences. The neural network was trained using 35 documents using Back-Propagation algorithm with variable learning rate and momentum term. For each document in the training set, we provided the desired target output of 1 for each sentence that was included in the summary and a 0 for all the sentences that were not included in the summary.

2.6 Result

We compared the results of our neural network model against a public domain multi-document summarization system, MEAD, available at *http://tangra.si.umich.edu/clair/md/demo.cgi*. Our neural network when tested on documents from the test set correctly classified 60% of the sentences as summary sentences against 48% of the sentences correctly classified as summary sentences by MEAD. We compared the summaries created with the summary created by hand for each document in the test set.

The results that we received were not as expected as one issue was the proper text extraction from html web pages. In most of the cases there were some unwanted non-stop words that were included in the text extracted from the search result.

3. Discussion and Future Wwork

The neural network model proposed enables a web user to view the summary of each web page returned by the search engine such that the summary generated tends to consist of those sentences which are more proximate or relevant to a search query specified by the user. This model can be used when the summary to be generated depends on some interest or relevance parameter of the document i.e. the summary generated highlights those points in the document which are more prevalent and describe its content. The summary generation method can be further enhanced by incorporating semantics of the document while generating the summary for example considering meaning of the words in a sentence to find closeness between two sentences.

4. References

[1] I. Mani, *Automatic Summarization*, John Benjamins Publishing Company, pp. 129-165, 2004.

[2] Perrone, M. P., Cooper, L. N., "When networks disagree: ensemble methods for hybrid neural networks", Neural Networks for Speech and Image Processing by R.J. Mammone, ed., Chapman-Hall, 1993.

[3] Cavnar, W.B., Using An N-Gram-Based Document Representation With A Vector Processing Retrieval Model, *Proc. of TREC-3 (Third Text REtrieval Conference)*. Gaithersburg, Maryland, USA. 1994

[4] W. T. Chuang and J. Yang. Extracting sentence segments for text summarization: A machine learning

approach, *Proceedings.of the 23rd ACM SIGIR*, pages 152–159, 2000.

[5] Kaikhah, Khosrow , Automatic Text Summarization with Neural Network, *Second IEEE International Conference on Intelligent Systems*, 2004.

[6] R. Brandow, K. Mitre and L. Rau, "Automatic condensation of electronic publications by sentence selection", *lnfmrno1ion Processing ond Manogernen1,* vol. 31, *"0.5,* pp. 675-685, 1995.

[7] P.B. Baxendale, "Machine-Made Index for Technical Literature: *An* Experiment" IBM Journal of Research and Developmen< vol. *2,* no. 4, pp. 354361, 1953.