***ABSTRACT :***

Heterogeneous types of gene expressions may provide a better insight into the biological role of gene interaction with the environment, disease development and drug effect at the molecular level. In this paper for both exploring and prediction

purposes a Time Lagged Recurrent Neural Network with trajectory learning is proposed for identifying and classifying the gene functional patterns from the heterogeneous nonlinear time series microarray experiments. The proposed procedures identify gene functional patterns from the dynamics of a state-trajectory learned in the heterogeneous time series and the gradient information over time. Also, the trajectory learning with Backpropagation through time algorithm can recognize gene expression patterns vary over time. This may reveal much more information about the regulatory network underlying gene expressions.

The analyzed data were extracted from spotted DNA microarrays in the budding yeast expression measurements, produced by Eisen et al. The gene matrix contained 79 experiments over a variety of heterogeneous experiment conditions. The number of recognized gene patterns in our study ranged from two to ten and were divided into three cases.

Optimal network architectures with different memory structures were selected based on Akaike and Bayesian information statistical criteria using two-way factorial design. The optimal model performance was compared to other popular gene classification algorithms such as Nearest Neighbor, Support Vector Machine, and Self-Organized Map. The reliability of the performance was verified with multiple iterated runs.

## CHAPTER 1

## introduction

**1.1 INTRODUCTION TO NEURAL NETWORK:**

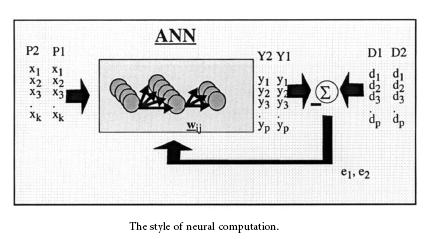
An artificial neural network is a system based on the operation of biological neural networks, in other words, is an emulation of biological neural system. Why would be necessary the implementation of artificial neural networks? Although computing these days is truly advanced, there are certain tasks that a program made for a common microprocessor is unable to perform; even so a software implementation of a neural network can be made with their advantages and disadvantages.

Another aspect of the artificial neural networks is that there are different architectures, which consequently requires different types of algorithms, but despite to be an apparently complex system, a neural network is relatively simple.

Artificial neural networks are among the newest signal processing technologies nowadays. The field of work is very interdisciplinary, but the explanation I will give you here will be restricted to an engineering perspective.

In the world of engineering, neural networks have two main functions: Pattern classifiers and as non linear adaptive filters. As its biological predecessor, an artificial neural network is an adaptive system. By adaptive, it means that each parameter is changed during its operation and it is deployed for solving the problem in matter. This is called the training phase.

A artificial neural network is developed with a systematic step-by-step procedure which optimizes a criterion commonly known as the learning rule. The input/output training data is fundamental for these networks as it conveys the information which is necessary to discover the optimal operating point. In addition, a non linear nature make neural network processing elements a very flexible system.



Basically, an artificial neural network is a system. A system is a structure that receives an input, process the data, and provides an output. Commonly, the input consists in a data array which can be anything such as data from an image file, a WAVE sound or any kind of data that can be represented in an array. Once an input is presented to the neural network, and a corresponding desired or target response is set at the output, an error is composed from the difference of the desired response and the real system output.

The error information is fed back to the system which makes all adjustments to their parameters in a systematic fashion (commonly known as the learning rule). This process is repeated until the desired output is acceptable. It is important to notice that the performance hinges heavily on the data. Hence, this is why this data should pre-process with third party algorithms such as DSP algorithms.

In neural network design, the engineer or designer chooses the network topology, the trigger function or performance function, learning rule and the criteria for stopping the training phase. So, it is pretty difficult determining the size and parameters of the network as there is no rule or formula to do it. The best we can do for having success with our design is playing with it. The problem with this method is when the system does not work properly it is hard to refine the solution. Despite this issue, neural networks based solution is very efficient in terms of development, time and resources. By experience, I can tell that artificial neural networks provide real solutions that are difficult to match with other technologies.

Fifteen years ago, Denker said: “artificial neural networks are the second best way to implement a solution” this motivated by their simplicity, design and universality. Nowadays, neural network technologies are emerging as the technology choice for many applications, such as patter recognition, prediction, system identification and control.

**1.2 INTRODUCTION TO BAYESIAN APPROACH:**

A Bayesian network, belief network or directed acyclic graphical model is a [probabilistic graphical model](http://en.wikipedia.org/wiki/Graphical_model) that represents a set of [random variables](http://en.wikipedia.org/wiki/Random_variables) and their [conditional dependencies](http://en.wikipedia.org/wiki/Conditional_independence) via a [directed acyclic graph](http://en.wikipedia.org/wiki/Directed_acyclic_graph) (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are [directed acyclic graphs](http://en.wikipedia.org/wiki/Directed_acyclic_graph) whose nodes represent [random variables](http://en.wikipedia.org/wiki/Random_variables) in the [Bayesian](http://en.wikipedia.org/wiki/Bayesian_probability) sense: they may be observable quantities, [latent variables](http://en.wikipedia.org/wiki/Latent_variable), unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a [probability function](http://en.wikipedia.org/wiki/Probability_function) that takes as input a particular set of values for the node's [parent](http://en.wikipedia.org/wiki/Glossary_of_graph_theory#Directed_acyclic_graphs) variables and gives the probability of the variable represented by the node. For example, if the parents are *m* Boolean variables then the probability function could be represented by a table of 2*m* entries, one entry for each of the 2*m* possible combinations of its parents being true or false.

Efficient algorithms exist that perform [inference](http://en.wikipedia.org/wiki/Inference) and [learning](http://en.wikipedia.org/wiki/Machine_learning) in Bayesian networks. Bayesian networks that model sequences of variables (*e.g.* [speech signals](http://en.wikipedia.org/wiki/Speech_recognition) or [protein sequences](http://en.wikipedia.org/wiki/Peptide_sequence)) are called [dynamic Bayesian networks](http://en.wikipedia.org/wiki/Dynamic_Bayesian_network). Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called [influence diagrams](http://en.wikipedia.org/wiki/Influence_diagrams).

**1.3 INTRODUCTION TO SELF ORGANISING NEURAL NETWORK :**

SOMs generally present a simplified, relational view of a highly complex data set.Once map objects or nodes are organized, all the data associated with a given node may be made available via that node. However this does not mean that all this data participated in the process of self-organization. A data set of nations might self-organize by annual rainfall, and once organized provide additional information such as color-coding by GNP.

A Self-organizing Map is a data visualization technique developed by Professor Teuvo Kohonen in the early 1980's. SOMs map multidimensional data onto lower dimensional subspaces where geometric relationships between points indicate their similarity. The reduction in dimensionality that SOMs provide allows people to visualize and interpret what would otherwise be, for all intents and purposes, indecipherable data. SOMs generate subspaces with an unsupervised learning neural network trained with a competitive learning algorithm. Neuron weights are adjusted based on their proximity to "winning" neurons (i.e. neurons that most closely resemble a sample input). Training over several iterations of input data sets results in similar neurons grouping together and vice versa. The components of the input data and details on the neural network itself are described in the "Basics" section. The process of training the neural network itself is presented in the "Algorithm" section. Optimizations used in training are discussed in the "Optimizations" section.

SOMs have been applied to several problems. The simple yet powerful algorithm has been able to reduce incredibly complex problems down to easily interpreted data mappings. The main drawback of the SOM is that it requires neuron weights be necessary and sufficient to cluster inputs. When an SOM is provided too little information or too much extraneous information in the weights, the groupings found in the map may not be entirely accurate or informative. This shortcoming, along with some other problems with SOMs are addressed in the "Conclusions" section.

# CHAPTER 2:

# BRIEF DISCRIPTION OF HUMAN NERVOUS SYSTEM:

The human nervous system may be viewed as a three stage system, as depicted in the block diagram of figure 2.central to the system is the brain, represented by the neural net, which continually receives information, perceives it, and makes appropriate decisions. Two sets of arrows are shown in the figure. Those pointing from left to right indicate the forward transmission of information bearing signals through the system. The arrows pointing right to left signify the presence of feed back in the system. The receptors convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net. The effectors convert electrical impulses generated by the neural net into discernible responses as system outputs.

**RECE**

**P**

**TOR**

**S**

**N**

**E**

**U**

**R**

**A**

**L**

**NET**

**E**

**F**

**F**

**E**

**C**

**T**

**OR**

**S**

**RESPONSE**

**STIMULUS**

FIGURE 2: Block Diagram Representation Of Nervous System.

A neuron is a an excitable cell in the nervous system that processes and transmits information by electrochemical signaling .In neuroscience, a neural network describes a population of physically interconnected neurons or a group of disparate neurons whose inputs or signalling targets define a recognizable circuit. Communication between neurons often involves an electrochemical process. The interface through which they interact with surrounding neurons usually consists of several dendrites (input connections), which are connected via synapses to other neurons, and one axon (output connection). If the sum of the input signals surpasses a certain threshold, the neuron sends an action potential (AP) at the axon hillock and transmits this electrical signal along the axon. The neural of nervous system is as shown figure 3

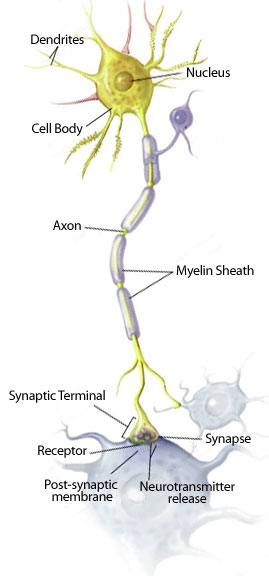


Figure3:Biological Neuron

A neuron accumulates chemical signals from its dendrites, and if the total chemical accumulation exceeds a threshold within a period of time, the neuron "fires", sending its own signal through its axon. Some neurons are capable of firing pulses on the order of 100 Hz. The signals passing through neurons involve accumulations of Sodium (Na), Potassium (K), and Chlorine (Cl) ions, and a resulting electrochemical potential (i.e., voltage).The resting voltage (-70 mV) and firing voltage (+30 mV) can be measured or even influenced by conventional electrical circuitry. The human brain has approximately **10^11** (100 billion) neurons. Each neuron in the cerebellum receives input from as many as **10^4** (10000) synapses. Although the axon and dendrites of a neuron often extend only a few micrometers away from the cell body, some axons are on the order of a meter in length. A brain has neurons with relatively short axons grouped in areas or clusters. A brain also has bundles of neurons with relatively long axons to link areas separated by centimeters. Thus a hierarchical network of processing elements is formed .

* 1. **MODELS OF NEURON:**

A neuron is an information processing unit that is fundamental to the operation of a neural network. the block diagram of figure 4 shows the model of a neuron, which forms the basis for designing neural networks. Here we identify three basic elements of the neuronal model:

1.A set of synapses or connecting links, each of which is characterized by a weight or strength of its own. Specifically ,a signal Xj the input of synapse j connected to neuron k is multiplied by the synaptic weight *wkj*. It is important to make a note of the manner in which the subscripts of synaptic weight *wkj* are written. The first subscript refers to the neuron in question and the second subscript refers to the input end to the synapse to which the weight refers. Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.

2.An adder for summing the input signals, weighted by the respective synapses of the neuron; the operation described here constitute a linear combiner.

3.An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function in that it squashes the permissible amplitude range of the output signal to some finite value.

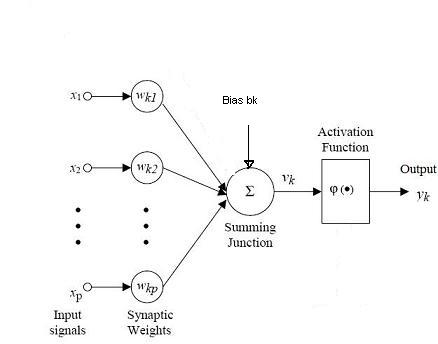


Figure 4:nonlinear model of a neuron.

Typically,the normalized amplitude ramge of the output of a neuron is written as the closed unit interval[0,1] or alternatively[1,-1].The neuron model of fig4 also includes an externally applied bias, denoted by bk. the bias bk has the effect of increasing or lowering the net input of the activation function, depending on whether it is positive or negative respectively.

In mathematical terms, we may describe a neuron k by writing the following pair of eqations:

untitled………….1

and yk=uk+bk)………………………..2

Where x1,x2,.,.,.,.,xm are the input signals;wk1,wk2,.,.,.,wkm are the synaptic weights of neuron k; uk is the linear combiner output due to the input signals; bk is the bias; is the activation function ;and yk is the output signal of the neuron .the use of bias bk has the effect of applying an affine transformation of the output uk of the linear combiner in the model of figure 5,as shown by

Vk=uk+bk……………………………………3In particular, depending on whether the bias bk is positive or negative ,the relationship between the induced local field or activation potential vk of the neuron k and the linear combiner output uk is modified in the manner illustrated in the figure 6.hereafter the term “induced local field” is used. Here we can note that as a result of this affine transformation ,the graph of vk versus uk no longer passes through the origin.

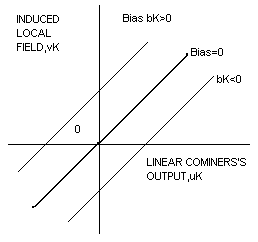


Figure 5:affine transformation produced by the presence of bias;note that vk=bk at uk=0.

The bias is an external parameter of artificial neuron k.

We may account for this presence as in equation 1.Equivalently, we may formulate the combination of eqs 1 to 3 as follows:

m

j=0wkxj

vk**=**

……………………………4

and yk=(Vk)……………………………………….5

in Eq(4) we have added a new synapses .its input is

x0=+1………………………………………………………………6

an its weight is

wk0=bk…………………………………………………………….7

we may therefore reformulate the model of neuron *k*  as in figure 6.In this figure, the effect of bias is accounted for by doing two things:

1.adding anew input signal fixed at +1

2.adding a new synaptic weight equal to the bias bk. Although models of figure 4 and 6 are different in appearance, they are mathematically equivalent.

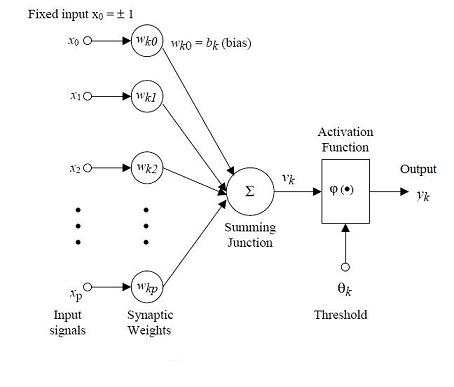


Figure 6:another non linear model of neuron.

# CHAPTER 3:

# NETWORK ARCHITECURES

The three fundamental different classes of network architectures are:

1.Single Layer Feedforward Networks.

2.Multilayer Feedforward Networks.

3.Recurrent networks

**3.1 SINGLE LAYER FEEDFORWARD NETWORKS**

In a layered neural network the neurons are organized in the form of layers. In its simplest form of a layered network, we have an input layer of source nodes that projects onto an output layer of neurons(computation nodes), but not vice versa .In other words, this network is strictly a feed forward or acyclic type .It is illustrated in the figure 7 for the case of four nodes in both input and output layers .Such a network is called a single layer network ,with the designation” single layer” referring to the output layer of computations nodes(neurons).We do not count the input layer of source node because no computations is preformed there.

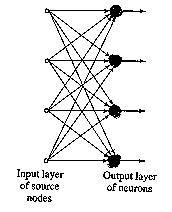
****

Figure 7: feedforward or the acyclic network with a single layer of neurons.

**3.2 MULTILAYER FEEDFORWARD NETWORKS:**

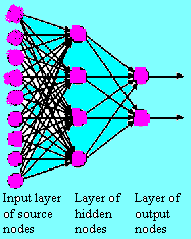
A typical multi-layer feed-forward network (as seen in Figure 8) has input layers, hidden layers, and output layers The input layers have the main function of receiving inputs and then buffering the input signals. The signals from the input layers are then transmitted to hidden layers, particularly hidden neurons or hidden units. The function of hidden neurons is to intervene between the external input and the network output in some useful manner. The processing elements in each layer are called nodes or units. Each of the nodes is connected to the nodes of neighboring layers. The parameters associated with each of these connections are called weights.Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down. This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of.

Figure 8 : Fully Connected Feedforwards Or Acyclic Network With One Hidden Layer And One Output Layer.

**3.3 RECURRENT NETWORKS:**

Recurrency nodes connect back to other nodes or themselves, information flow is multidirectional, sense of time and memory of previous state(s). Recurrent networks, in contrast to feed-forward networks, do have feedback elements that enable signals from one layer to be fed back to a previous layer.

A basic recurrent network is shown in figure 9. A simple recurrent network is one with three layers, an input, an output, and a hidden layer. A set of additional context units are added to the input layer that receive input from the hidden layer neurons. The feedback paths from the hidden layer to the context units have a fixed weight of unity.

A fully recurrent network is one where every neuron receives input from all other neurons in the system. Such networks cannot be easily arranged into layers. A small subset of neurons receives external input, and another small subset produce system output.

Recurrent networks often have what are called attractor states. This means that signals passing through the recurrent net are fed back and changed until they fall into a repeating pattern, which is then stable (i.e. it repeats itself indefinitely as it rattles round the loop). This is a little like a ball being placed on a slope and released - it rolls downhill until it reaches the bottom of a valley, and then stops. The input signals change until they reach one of these attractor states, and then they remain stable. The secret with recurrent networks is to train the weights so that the attractor states are the ones that you want. Recurrent network has input nodes, hidden neurons, output neurons, just as feedforward networks, what distinguish themselves from feedforward networks is that they have at least one *feedback* loop, i.e. when a neurons output is fed back into the network as input. A special type of feedback loop are called self-feedback loops, which refers to a situation in which the output of a neuron is fed back to its own input. Another type of feedback loop introduces a new type of nodes called *context* nodes [[fully connected recurrent network](http://rslab.movsom.com/paper/somrs/html/chapter3.html#figure:fully_connected_recurrent_network)] . These nodes receive connections from the hidden neurons or the output layers of the network and have output connections that travel back to the hidden neurons. Context nodes are required when learning patterns over time (i.e. when the past value of the network influences the present processing). Recurrent networks can therefore be seen as an attempt of incorporate time and memory into a neural network Two examples of recurrent networks that are simple extensions of feedforward networks are *Jordan network* (feedback from output layer to input layer) and *Elman network* (feedback from hidden layer to input layer). Also, a widely known recurrent network is the *Hopfield network* in which all the connections are symmetric.

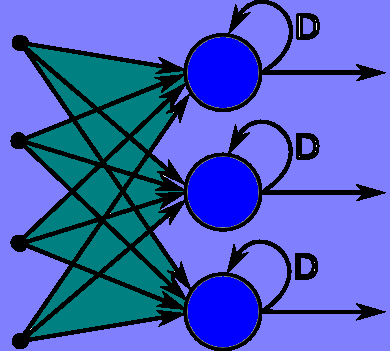


Figure 9:*Simple Recurrent Networks*

# CHAPTER 4:

# LEARNING PROCESS:

One of the most important aspects of Neural Network is the learning process. To describe that process I am going to use a nice analogy from Thomas Lahore: The learning process of a Neural Network can be viewed as reshaping a sheet of metal, which represents the output (range) of the function being mapped. The training set (domain) acts as energy required to bend the sheet of metal such that it passes through predefined points. However, the metal, by its nature, will resist such reshaping. So the network will attempt to find a low energy configuration (i.e. a flat/non-wrinkled shape)that satisfies the constraints (training data). Learning can be done in supervised or unsupervised manner. In supervised training, both the inputs and the outputs are provided.  
The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then calculated, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are continually tweaked.

In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization or adaption.

**4.1 ERROR-CORRECTION LEARNING:**

Error correction is, as the name indicates, about correcting the synaptical strengths according to the error in the neurons output.

Mathematically the definition of a neurons output error is given by this equation:  
  
errorsignal

In the equation, time should also be involved, so it looks like this.  
  
errorwithtime  
  
Here *n* is an abbreviation for time. The reason why time is involved in the equation is, that the [algorithm](JavaScript:void(null);), as mentioned, gradually adapts the synaptical strengths, so you can regard *n* as the number of times the neuron has been updated, or how many training examples that has been run trough. The actual learning process doesn't take place before the calculation of the neurons' error-signal is completed, and the changing of the weights is made.

If we look at neuron k we can set up the equation for the update of its synaptical weights. The change of the neuron k´s synaptic weight is determined by the error of the neurons output, the associated synapses and a learning rate. This rule is named The Widrow-Hoff-rule, or simply The Delta-rule.  
deltarule  
The error signal ek is determining the rate of learning. The larger error the larger change.  
The input signals (synapses) xj also determines the rate of learning. If an input signal is small, it haven’t got much influence in the neurons error. The associated synaptic weight, should likewise not be changed much.  
The learning rate etais an auxiliary value determining the rate of learning. It can be set by the teacher to control the updates of the synaptic weights. The update of neuron k's synaptical strengths is now very simple.  
The future strength of synapse j in neuron k has to be equal to the present plus the given alteration of the synapse.  
err-c-update  
The synapses of neuron k are now altered and the result of the same input should now be closer to the wanted than before.  
We will now go through an example to visualize our theory. To make the example understandable, we will use a single neuron, with only three synapses. Our neuron will be a very small, [binary calculator](http://library.thinkquest.org/C007009/introduction/programming/programming-4.html) as the one we have seen already. The only difference is, that the synaptical strengths are now no longer predefined and that the neuron must be trained to work.

**4.2 HEBBIAN LEARNING:**

When the axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased'' . As a result, a learning method evolved for artificial neural networks, called Hebbian learning, which is an instance of an unsupervised learning procedure. In Hebbian learning, weights between learning nodes are adjusted so that each weight better represents the relationship between the nodes. Nodes which tend to be positive or negative at the same time will have strong positive weights while those which tend to be opposite will have strong negative weights. Nodes that are uncorrelated will have weights near zero. For example, if two nodes A and B are often simultaneously active, Hebbian learning will increase the connection strength between the two so that excitation of either one tends to cause excitation of the other. On the other hand, if nodes A and C were of opposite activations at all times, then Hebbian learning would gradually decrease the connection in between below zero so that an excited A or C would inhibit the other. Hebbian learning has four features interesting to the cognitive scientist: first it is unsupervised; second it is a local learning rule, meaning that it can be applied to a network in parallel; third it is simple and therefore requires very little computation; fourth it is biologically plausible .

Formally, Hebb's rule  for the modification of a weight $w_{ij}$from neuron $i$to $j$with a learning rate $\eta$is defined as \begin{displaymath}{\Delta}w_{ij} = {\eta}ij.\end{displaymath}

This rule is not entirely unproblematic, because there exists no bound on the weight of the connection. A common solution to this problem is the introduction of a decay function for each connection. However, given the controlled usage of Hebbian learning in the neural optimal controller constructed here, this step is not necessary.

**4.3 HEBB’A HYPOTHESIS:**

Hebbian theory describes a basic mechanism for [synaptic plasticity](http://en.wikipedia.org/wiki/Synaptic_plasticity) wherein an increase in [synaptic](http://en.wikipedia.org/wiki/Synapse) efficacy arises from the [presynaptic](http://en.wikipedia.org/wiki/Presynaptic) cell's repeated and persistent stimulation of the [postsynaptic](http://en.wikipedia.org/wiki/Postsynaptic) cell. Introduced by [Donald Hebb](http://en.wikipedia.org/wiki/Donald_Olding_Hebb) in 1949, it is also called Hebb's rule, Hebb's postulate, and cell assembly theory, and states:Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.… When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.The theory is often summarized as "cells that fire together, wire together", although this is an oversimplification of the [nervous system](http://en.wikipedia.org/wiki/Nervous_system) not to be taken literally, as well as not accurately representing Hebb's original statement on cell connectivity strength changes. The theory is commonly evoked to explain some types of associative learning in which simultaneous activation of cells leads to pronounced increases in [synaptic strength](http://en.wikipedia.org/wiki/Chemical_synapse#Synaptic_strength). Such learning is known as Hebbian learning.

When the axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased'' . As a result, a learning method evolved for artificial neural networks, called Hebbian learning, which is an instance of an unsupervised learning procedure. In Hebbian learning, weights between learning nodes are adjusted so that each weight better represents the relationship between the nodes. Nodes which tend to be positive or negative at the same time will have strong positive weights while those which tend to be opposite will have strong negative weights. Nodes that are uncorrelated will have weights near zero. For example, if two nodes A and B are often simultaneously active, Hebbian learning will increase the connection strength between the two so that excitation of either one tends to cause excitation of the other. On the other hand, if nodes A and C were of opposite activations at all times, then Hebbian learning would gradually decrease the connection in between below zero so that an excited A or C would inhibit the other.

Hebbian learning has four features interesting to the cognitive scientist: first it is unsupervised; second it is a local learning rule, meaning that it can be applied to a network in parallel; third it is simple and therefore requires very little computation; fourth it is biologically plausible .

Formally, Hebb's rule for the modification of a weight $w_{ij}$from neuron $i$to $j$with a learning rate $\eta$is defined as

\begin{displaymath}{\Delta}w_{ij} = {\eta}ij.\end{displaymath}

This rule is not entirely unproblematic, because there exists no bound on the weight of the connection. A common solution to this problem is the introduction of a decay function for each connection. However, given the controlled usage of Hebbian learning in the neural optimal controller constructed here, this step is not necessary.

## 4.4COMPETITIVE LEARNING:

## *Competitive learning is a rule based on the idea that only one neuron from a given iteration in a given layer will fire at a time. Weights are adjusted such that only one neuron in a layer, for instance the output layer, fire. Competitive learning is useful for classification of input patterns into a discrete set of output classes. The “winner” of each iteration, element i\* , is the element whose total weighted input is the largest. Using this notation, one example of a competitive learning rule can be defined mathematically as:*

wij[n + 1] = wij[n] + Δwij[n]

\Delta w_{ij}[n] = \left\{ \begin{matrix} \eta
 (x_i - w_{ij}) & \mbox{ if } i = j \\ 0 & \mbox{ 
otherwise}\end{matrix}\right.

Competitive learning is a simple learning algorithm, which is very useful for letter-recognition. In competitive learning there is only one active neuron at a time, and it's only the active neuron that is trained.  
Every neuron has its own synaptical strength so there is no one that gives the same output at any time. The reason for this is that it is only the neuron that makes the largest output that receives training, this neuron becomes a so-called [winner-takes-it-all](JavaScript:%7bvoid(null);%7d) neuron. It can be set up mathematically like [this](JavaScript:%7bvoid(null);%7d).

Actually this just means that all outputs is run through and the neuron with the largest output, has an output at 1. The rest of the neurons will have 0 as output.

As said before it is only the active neuron that receives training. When the active neuron is to be trained, the [vector](JavaScript:void(null);) of synaptical strength comes to look like the vector of input. This is done according to the rule about competitive learning as [this.](JavaScript:%7bvoid(null);%7d) A really good image of how competitive learning performs shows the network [before and after training](JavaScript:%7bvoid(null);%7d). [This program](http://library.thinkquest.org/C007009/download/pREC.zip) is an implementation of a neural network with competitive learning here you can see how the network is able to generalize by itself, and divide certain patterns into the right categories.

At the bottom of this program there is a graph that shows how large alterations has been made on the synaptical strengths. It shows that the network learns less the closer it comes to an actual classification of the pattern itself.

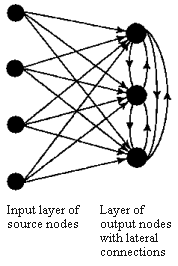


Figure 10:Competitive Learning Network

The network may include lateral connections among the neurons, as shown in [[competitive neural networks](http://rslab.movsom.com/paper/somrs/html/chapter3.html#figure:competitive_neural_networks)] , and thereby perform lateral inhibition, with each neuron tending to inhibit the neuron to which it is laterally connected. A neuron k is the winning neuron if its net internal activity level Sk for a given input vector x is the largest one among all neurons in the network, where Sk is the combination of all the forward and feedback inputs for neuron k . The output signal yk of winning neuron k is set to one and all the others to zero. Each neuron is allowed a fixed amount of synaptic weight (typically, all synaptic weights are positive), which is distributed among its input nodes; that is, we have \sum_i w_{ik} = 1for all k and where wik denotes the synaptic weight between input node i and neuron k . A neuron learns by shifting synaptic weights from its inactive input nodes to the active ones, no learning takes place if the neuron don't respond to a particular input vector x . The change \Delta w_{ik}applied to synaptic weight wik is defined by the standard competitive learning rule:

\Delta w_{ik} =
\left\{
  \begin{array}{ll}
     \eta (x_i - w_{ik}) & \mbox{if neruon k wins}\\
     0 & \mbox{if neruon k loses}\\
  \end{array}
\right.

where η is the learning parameter. The effect of this rule is that the synaptic weight vector wk of the winning neuron k is moved toward the input vector x .

**4.5 LEARNING WITH A TEACHER:**

Supervised learning is a [machine learning](http://en.wikipedia.org/wiki/Machine_learning) technique for deducing a function from training data. The [training data](http://en.wikipedia.org/wiki/Training_set) consist of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (called [regression](http://en.wikipedia.org/wiki/Regression_analysis)), or can predict a class label of the input object (called [classification](http://en.wikipedia.org/wiki/Classification_%28machine_learning%29)). The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). To achieve this, the learner has to generalize from the presented data to unseen situations in a "reasonable" way (see [inductive bias](http://en.wikipedia.org/wiki/Inductive_bias)). (Compare with [unsupervised learning](http://en.wikipedia.org/wiki/Unsupervised_learning).) The parallel task in human and animal psychology is often referred to as [concept learning](http://en.wikipedia.org/wiki/Concept_learning).

Supervised learning can generate models of two types. Most commonly, supervised learning generates a global model that maps input objects to desired outputs. In some cases, however, the map is implemented as a set of local models (such as in [case-based reasoning](http://en.wikipedia.org/wiki/Case-based_reasoning) or the [nearest neighbor algorithm](http://en.wikipedia.org/wiki/Nearest_neighbor_%28pattern_recognition%29)).

In order to solve a given problem of supervised learning (e.g. learning to [recognize handwriting](http://en.wikipedia.org/wiki/Handwriting_recognition)) one has to consider various steps:

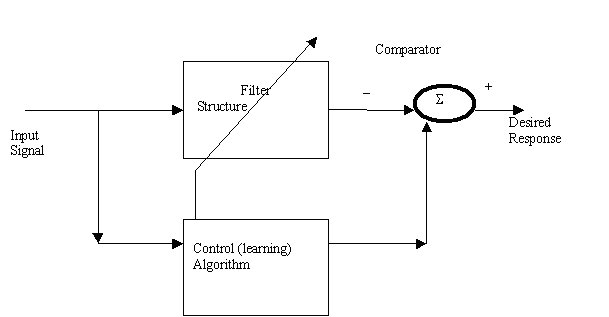
1.Determine the type of training examples. Before doing anything else, the engineer should decide what kind of data is to be used as an example. For instance, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.

2.Gathering a training set. The training set needs to be characteristic of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.

3.Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the [curse of dimensionality](http://en.wikipedia.org/wiki/Curse_of_dimensionality); but should be large enough to accurately predict the output.

4.Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use [artificial neural networks](http://en.wikipedia.org/wiki/Artificial_neural_network) or [decision trees](http://en.wikipedia.org/wiki/Decision_tree_learning).

5.Complete the design. The engineer then runs the learning algorithm on the gathered training set. Parameters of the learning algorithm may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via [cross-validation](http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29). After parameter adjustment and learning, the performance of the algorithm may be measured on a test set that is separate from the training set.



**Figure 11:Supervised Learning**

***4.6* UNSUPERVISED LEARNING:**

Unsupervised learningThe network is trained using input signals only. In response, the network organises internally to produce outputs that are consistent with a particular stimulus or group of similar stimuli. Inputs form clusters in the input space, where each cluster represents a set of elements of the real world with some common features.

In both cases once the network has reached the desired performance, the learning stage is over and the associated weights are *frozen*. The final state of the network is preserved and it can be used to classify new, previously unseen inputs. At the testing stage, the network receives an input signal and processes it to produce an output. If the network has correctly learnt, it should be able to generalize, and the actual output produced by the network should be almost as good as the ones produced in the learning stage for similar inputs.

# CHAPTER 5

# BAYESIAN APPROACH:

**5.1 Bayesian Methodology:**

The concept behind the Bayesian modelling framework is Bayes’ theorem, which states that any prior beliefs regarding an uncertain quantity are updated,based on new information, to yield a posteriorprobability of the unknown quantity. In terms of an ANN, Bayes’ theorem can be used to estimatethe posterior distribution of the network weights

w = {w1, . . . ,wd} given a set of N target data

y = {y1, . . . , yN} and an assumed model structure H

as follows:

p(w|y,H) = p(y|w,H)p(w|H)

p(y|H) =Rp(y|w,H)p(w|H)dwN(1)

In this equation, p(w|H) is the *prior* distribution, which describes any knowledge of the weight values before observing the data; p(y|w,H) isknown as the *likelihood* function and is obtainedby comparing the observed data y to the model outputs ˆy. This is the function through which the prior knowledge of w is updated by the data. The denominator p(y|H) is a normalizing constant known as the marginal likelihood, or *evidence*, of the model. When estimating the posterior of the weights, it is common to ignore this term, instead writing (1) as the proportionality

p(w|y,H) / p(y|w,H)p(w|H). However, when using Bayesian methods for model selection, the model evidence becomes very important.

**5.2 Bayesian Model Selection (BMS)**

Given a set of H competing models, Bayes’ theorem can be rewritten to infer the posterior probability that each model Hi, where i = 1, . . . ,H, is the “true” model of the system given the observed data, asfollows:

p(Hi|y) = p(y|Hi)p(Hi)

p(y) =PHj=1 p(y|Hj)p(Hj)(2)

where p(Hi) is the prior probability assigned to Hiand p(y|Hi) is the evidence of the model, which is the denominator in (1). It is unlikely that any model willactually be the “true” model of the system; however,the Bayes’ approach enables the relative merits ofthe competing models to be compared in an objective

manner.It is generally assumed that the prior probabilitiesassigned to the different models are approximately equal, as a model thought to be highly implausiblewould not even be considered in the comparison.

Furthermore, even without a prior preference forsimple models, the evidence of a model automaticallyfavours simple theories, as discussed in MacKay(1995). Therefore, (2) can be simplified to:

p(Hi|y) = p(y|Hi)

PHj=1 p(y|Hj)/ p(y|Hi) (3)

which states that the relative probabilities of thecompeting models can be compared based on their evidence. The ratio of two models’ posteriorprobabilities is called the *Bayes’ factor* BF, which,when assuming equal prior probabilities, is defined by:

BF2,1 = p(H2|y)p(H1|y)= p(y|H2)p(y|H1)(4)

In order to interpret the information provided byBF2,1 in terms of the evidence against modelH1 in favour of model H2, Kass and Raftery(1995) suggest using the interpretive scale given inTable 1. The problem of BMS then becomes one of**Table 1.** Bayes’ factor interpretive scale2 loge BF2,1 Evidence against H10 to 2 Weak2 to 6 Positive

6 to 10 Strong

> 10 Very strong

estimating the evidence of each competing model andranking the models according to their Bayes’ factors.Nevertheless, this task is far from trivial. As shownin (1), the evidence can be evaluated by the integral

p(y|H) =Rp(y|w,H)p(w|H)dw; however, exceptfor the simplest of models, this integral is analytically

intractable. Therefore, alternative methods are neededto estimate p(y|H).

**5.3 Proposed BMS Framework**

In order to estimate p(y|H), (1) can be rearranged as follows:

p(y|H) = p(y|w,H)p(w|H)p(w|y,H)(5)

However, for ANNs (and other complex models),direct evaluation of this equation is impossible, as the posterior weight distribution p(w|y,H) is analyticallyintractable. Recently, Markov chain Monte Carlo(MCMC) methods for simulating observations fromposterior distributions have increased in popularity.As discussed in Kingston et al. (2005), thereare numerous benefits to estimating the posteriordistribution of ANN weights p(w|y,H) with MCMCmethods. Therefore, the proposed BMS approachis based on approximating p(y|H) using MCMCposterior simulations, as this only requires a simpleadditional step after sampling from p(w|y,H).

Furthermore, the weight distributions obtained fromthe MCMC simulation may provide a useful check for the accuracy to the approximated Bayes’ factors.

**5.4 MCMC Sampling of the Posterior Weights**

The first step in the proposed framework involves selecting an appropriate likelihood function and prior weight distribution. Assuming that the residuals between the observed data and the model outputsare normally and independently distributed with zero mean and constant variance \_2, the likelihood

function is given by:

In this study, a wide uniform prior on the range[-100,100] was assumed for each weight in order to

specify an equal probability of a weight taking onpositive or negative values, but an otherwise lack of

prior knowledge about the weights.As the likelihood function given by (6) depends notonly on the value of w, but also on the value ofthe variance \_2, a two-step MCMC procedure wasused in this study to sample both w and \_2 from theposterior distribution. This involved the use of thetwo simplest MCMC algorithms: the Gibbs sampler

and the Metropolis algorithm. In the first step ofthis procedure, the variance parameter \_2 is heldconstant while the weights w are sampled from thedistribution:

p(w|\_2, y,H) / p(y|w, \_2,H)p(w,H) (7)using a Metropolis sampling step. As it isgenerally difficult to sample from p(w|\_2, y,H)directly, the Metropolis algorithm makes use of asimpler, symmetrical distribution Q(w\_|wt) (often

a multinormal distribution with mean wt), knownas the ‘proposal’ distribution, to generate candidate

weight vectors w\_ based on the current weight vectorwt, thus forming a random walk Markov chainwithin the weight space. An adaptive acceptancerejectioncriterion is employed such that this sequencecontinually adapts to the posterior distribution of theweights. This works by only accepting the candidateweight state according to theprobability \_, given by:

\_(w\_|wt) = min\_p(y|w\_,H)p(w\_|H)p(y|wt,H)p(wt|H) , 1(8)

If w\_ is accepted, wt+1 is set equal to w\_, otherwisewt+1 = wt and the process is repeated. In this study,rather than using the straight Metropolis algorithm, avariation developed by Haario et al. (2001) called theadaptive Metropolis (AM) algorithm was used, as it has been found to have a number of advantages overother variants of the Metropolis algorithm in terms

of efficiency and ease of use (Marshall et al., 2004).The AM algorithm was developed to overcome the problems experienced using the straight Metropolisalgorithm associated with selecting an appropriatecovariance for the proposal distribution. In thisalgorithm, the covariance of the proposal distributionis updated at each iteration based on all previousstates of the weight vector, ensuring that information gained about the proposal distribution throughout thesimulation is used to increase the efficiency of the algorithm and improve the convergence rate.In the second step of the MCMC procedure, theweights are held constant while \_2 is sampled fromthe full conditional distribution:

p(\_2|w, y,H) / p(y|w, \_2,H)p(\_2) (9)using the Gibbs sampler. To enable straightforward sampling from p(\_2|w, y), a noninformative conjugateinverse chi-squared prior \_2 \_ \_−2 (0.1, 0.01) was assumed. Given sufficient iterations, the sampledsequences should converge to a stationary distribution. From this point onwards, it can be consideredthat the sampled parameters are generated from theposterior distribution and can be used to a predictivedistribution for each given input pattern.

|  |
| --- |
|  |

**5.5 Computation of Evidence:**

There are a number of methods available forapproximating the evidence of a model using posterior simulations (see DiCiccio et al. (1997)). In thisstudy, the framework proposed by Chib and Jeliazkov (2001) was used due to its simplicity and ease ofprogramming. By taking the logarithm of (5) at some fixed point ˆw, the following expression is obtained



Thus, if ˆw is a sampled weight vector obtained using the above MCMC procedure, the only unknown in this equation is log p( ˆw|y,H). Therefore, estimation of the evidence is reduced to estimating the posterior weight density at a single point ˆw. Chib and Jeliazkov



where wi are sampled draws from the posterior weight distribution, wj are sampled draws from the proposal distribution Q and \_(·) is given by (8). Chib and Jeliazkov (2001) note that while thechoice of ˆw is arbitrary, fores is appropriate to choose a point that has high posterior density. Therefore, in this study, the median of the posterior distribution was chosen. Furthermore, they state that although J and K may be different, inpractice they are set to be equal.timation efficiency it

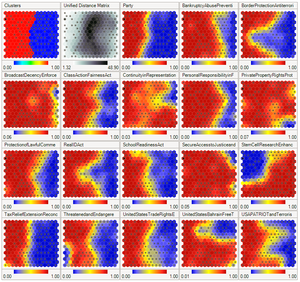
**5.6** **Checking Bayes Factors with Posterior Weight Distributions:**

There may be a number of problems associated with estimates of p(y|H) based on posterior simulations, for reasons discussed in DiCiccio et al. (1997). Therefore, in this framework, it is proposed that the Bayes’ factors calculated using the approximated evidence values be used as a guide for model selection, but a final check of the model rankings be carried out using the posterior weight distributions. If the marginal posterior distribution of a hiddento- output layer weight includes the value zero, this suggests that the associated hidden node may be pruned from the network without affecting model performance. If there are more than one hiddento- output layer weights with marginal posterior distributions that include zero, scatter plots of pairs of these weights should be inspected to determine whether the joint distribution of the weights passes through the origin (0,0), which would indicate that both weights in the pair may be pruned.

# CHAPTER 6

# SELF ORGANISING NEURAL NETWORK

A self-organizing map (SOM) or self-organizing feature map (SOFM) is a type of [artificial neural network](http://en.wikipedia.org/wiki/Artificial_neural_network) that is trained using [unsupervised learning](http://en.wikipedia.org/wiki/Unsupervised_learning) to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map. Self-organizing maps are different from other artificial neural networks in the sense that they use a neighborhood function to preserve the [topological](http://en.wikipedia.org/wiki/Topology) properties of the input space.

[](http://en.wikipedia.org/wiki/File:Synapse_Self-Organizing_Map.png)

[http://bits.wikimedia.org/skins-1.17/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:Synapse_Self-Organizing_Map.png)

A self-organizing map showing [U.S. Congress](http://en.wikipedia.org/wiki/United_States_Congress) voting patterns visualized in [Synapse](http://en.wikipedia.org/wiki/Peltarion_Synapse). The first two boxes show clustering and distances while the remaining ones show the component planes. Red means a yes vote while blue means a no vote in the component planes (except the party component where red is [Republican](http://en.wikipedia.org/wiki/Republican_Party_%28United_States%29) and blue is [Democrat](http://en.wikipedia.org/wiki/Democratic_Party_%28United_States%29)).

This makes SOMs useful for [visualizing](http://en.wikipedia.org/wiki/Scientific_visualization) low-dimensional views of high-dimensional data, akin to [multidimensional scaling](http://en.wikipedia.org/wiki/Multidimensional_scaling). The model was first described as an artificial neural network by the [Finnish](http://en.wikipedia.org/wiki/Finland) professor [Teuvo Kohonen](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Vector_quantization). Mapping automatically classifies a new input vector.

A self-organizing map consists of components called nodes or neurons. Associated with each node is a weight vector of the same dimension as the input data vectors and a position in the map space. The usual arrangement of nodes is a regular spacing in a [hexagonal](http://en.wikipedia.org/wiki/Hexagonal) or rectangular grid. The self-organizing map describes a mapping from a higher dimensional input space to a lower dimensional map space. The procedure for placing a vector from data space onto the map is to first find the node with the closest weight vector to the vector taken from data space. Once the closest node is located it is assigned the values from the vector taken from the data space.

While it is typical to consider this type of network structure as related to [feedforward networks](http://en.wikipedia.org/wiki/Feedforward_neural_networks) where the nodes are visualized as being attached, this type of architecture is fundamentally different in arrangement and motivation.

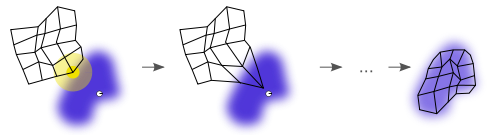
Useful extensions include using [toroidal](http://en.wikipedia.org/wiki/Torus) grids where opposite edges are connected and using large numbers of nodes. It has been shown that while self-organizing maps with a small number of nodes behave in a way that is similar to [K-means](http://en.wikipedia.org/wiki/K-means_algorithm), larger self-organizing maps rearrange data in a way that is fundamentally topological in character.

It is also common to use the [U-Matrix](http://en.wikipedia.org/wiki/U-Matrix). The U-Matrix value of a particular node is the average distance between the node and its closest neighbors (ref. 9). In a square grid for instance, we might consider the closest 4 or 8 nodes, or six nodes in a hexagonal grid.

Large SOMs display properties which are emergent. In maps consisting of thousands of nodes, it is possible to perform cluster operations on the map itself.

**6.1 Learning algorithm:**

The goal of learning in the self-organizing map is to cause different parts of the network to respond similarly to certain input patterns. This is partly motivated by how visual, auditory or other [sensory](http://en.wikipedia.org/wiki/Sense) information is handled in separate parts of the [cerebral cortex](http://en.wikipedia.org/wiki/Cerebral_cortex) in the [human brain](http://en.wikipedia.org/wiki/Human_brain).

[](http://en.wikipedia.org/wiki/File:Somtraining.svg)

[http://bits.wikimedia.org/skins-1.17/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:Somtraining.svg)

An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training sample drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node nearest to the training node (highlighted in yellow) is selected, and is moved towards the training datum, as (to a lesser extent) are its neighbours on the grid. After many iterations the grid tends to approximate the data distribution (right).

The weights of the neurons are initialized either to small random values or sampled evenly from the subspace spanned by the two largest [principal component](http://en.wikipedia.org/wiki/Principal_component) [eigenvectors](http://en.wikipedia.org/wiki/Eigenvectors). With the latter alternative, learning is much faster because the initial weights already give good approximation of SOM weights.

The network must be fed a large number of example vectors that represent, as close as possible, the kinds of vectors expected during mapping. The examples are usually administered several times as iterations.

The training utilizes [competitive learning](http://en.wikipedia.org/wiki/Competitive_learning). When a training example is fed to the network, its [Euclidean distance](http://en.wikipedia.org/wiki/Euclidean_distance) to all weight vectors is computed. The neuron with weight vector most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and with distance from the BMU. The update formula for a neuron with weight vector **Wv**(t) is

**Wv**(t + 1) = **Wv**(t) + Θ (v, t) α(t)(**D**(t) - **Wv**(t)),

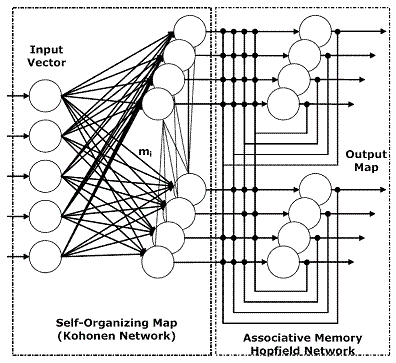
where α(t) is a [monotonically decreasing](http://en.wikipedia.org/wiki/Monotonically_decreasing) learning coefficient and **D**(t) is the input vector. The neighborhood function Θ (v, t) depends on the lattice distance between the BMU and neuron *v*. In the simplest form it is one for all neurons close enough to BMU and zero for others, but a [gaussian function](http://en.wikipedia.org/wiki/Gaussian_function) is a common choice, too. Regardless of the functional form, the neighborhood function shrinks with time.[[3]](http://en.wikipedia.org/wiki/Self-organizing_map#cite_note-Haykin-2) At the beginning when the neighborhood is broad, the self-organizing takes place on the global scale. When the neighborhood has shrunk to just a couple of neurons the weights are converging to local estimates.

### 6.2 Algorithm

1. Randomize the map's nodes' weight vectors
2. Grab an input vector
3. Traverse each node in the map
   1. Use [Euclidean distance](http://en.wikipedia.org/wiki/Euclidean_distance) formula to find similarity between the input vector and the map's node's weight vector
   2. Track the node that produces the smallest distance (this node is the best matching unit, BMU)
4. Update the nodes in the neighbourhood of BMU by pulling them closer to the input vector
   1. **Wv**(t + 1) = **Wv**(t) + Θ(t)α(t)(**D**(t) - **Wv**(t))
5. Increase t and repeat from 2 while *t* < λ

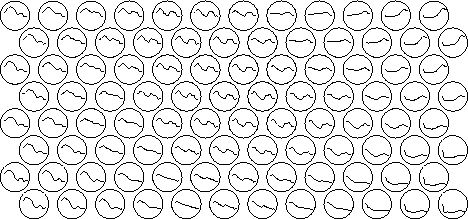
The combination of a SOM neural network with a Hopfield network. The self-organizing map (SOM) as proposed by Kohonen is a biologically motivated neural network that applies Hebbian learning in order to generate a topological representation of the data distribution from examples. In general, it defines a nonlinear mapping from the input data space R*n* onto a lower-dimensional array of nodes, usually organized in two dimensions. To each neuron a weight vector is assigned with the same dimensionality *n* as in the input space. A given input pattern is compared to the weight vector of each neuron and the neuron with the closest Euclidean distance is declared as the winner3. As result, the activation of the winning neuron is spread around its neighborhood, and therefore, spatially close neurons become sensitive to similar patterns. The neighborhood of the winner is determined by the topology of the lattice, that means, it is a function of the number of lateral connections to the winning neuron. The size of the neighborhood is initially large in order to preserve topology, but shrinks over iterations in order to allow neurons to specialize in the latter stages of training.

A Hopfield net is a type of recurrent neural network. Its state of the network at any time is given by the vector of the node outputs. Started in any initial state, the state of the system evolves to a final state that is a local minimum . Novelty detection is related with the discovery of stimuli not perceived before and the concept has been used in different areas like fault detection, learning of temporal signals and learning in mobile robotics. We follow Crook and Hayes who proposed to use a Hopfield network by detecting the presence or absence of a pattern by computing its energy. The energy of the net is calculated by ( 4.3) for the detection of already stored patterns. If a stored pattern is presented to the net, the energy goes down in the first iteration step, which does not occur when a new pattern is presented.



 The SOM is a new, effective software tool for the visualization of high-dimensional data. It converts complex, nonlinear statistical relationships between high-dimensional data items into simple geometric relationships on a low-dimensional display. As it thereby compresses information while preserving the most important topological and metric relationships of the primary data items on the display, it may also be thought to produce some kind of abstractions. These two aspects, visualization and abstraction, can be utilized in a number of ways in complex tasks such as process analysis, machine perception, control, and communication.

The SOM usually consists of a two-dimensional regular grid of nodes. A model of some observation is associated with each node (cf. Fig. 1).

  
*Figure 1:* In this exemplary application, each processing element in the hexagonal grid holds a model of a short-time spectrum of natural speech (Finnish). Notice that neighboring models are mutually similar.

The SOM algorithm computes the models so that they optimally describe the domain of (discrete or continuously distributed) observations.

The models are organized into a meaningful two-dimensional order in which similar models are closer to each other in the grid than the more dissimilar ones. In this sense the SOM is a similarity graph, and a clustering diagram, too. Its computation is a nonparametric, recursive regression process.

# 6.3 SORN: a self-organizing recurrent neural network

Understanding the dynamics of recurrent neural networks is crucial for explaining how the brain processes information. In the neocortex, a range of different plasticity mechanisms are shaping recurrent networks into effective information processing circuits that learn appropriate representations for time-varying sensory stimuli. However, it has been difficult to mimic these abilities in artificial neural network models. Here we introduce SORN, a self-organizing recurrent network. It combines three distinct forms of local plasticity to learn spatio-temporal patterns in its input while maintaining its dynamics in a healthy regime suitable for learning. The SORN learns to encode information in the form of trajectories through its high-dimensional state space reminiscent of recent biological findings on cortical coding. All three forms of plasticity are shown to be essential for the network’s success.

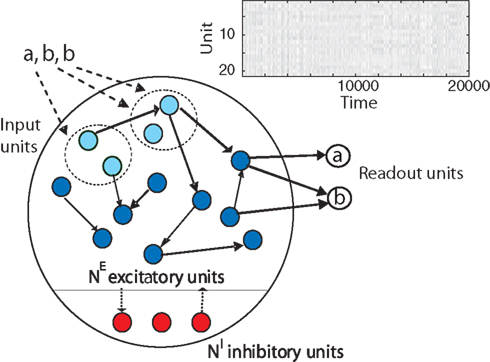
The mammalian neocortex is the seat of our highest cognitive functions. Despite much effort, a detailed characterization of its complex neural dynamics and an understanding of the relationship between these dynamics and cognitive processes remain elusive. Cortical networks present an astonishing ability to learn and adapt via a number of plasticity mechanisms which affect both their synaptic and neuronal properties. These mechanisms allow the recurrent networks in the cortex to learn representations of complex spatio-temporal stimuli. Interestingly, neuronal responses are highly dynamic in time (even when the stimulus is static) ([Broome et al., 2006](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B4) ) and contain a rich amount of information about past events ([Brosch and Schreiner, 2000](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B5) ; [Bartlett and Wang, 2005](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B1) ; [Broome et al., 2006](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B4) ; [Nikolic et al., 2006](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B20) ).

But mimicking these features in artificial neural networks has proven to be very difficult. The first models that could address temporal tasks have incorporated in their structure an explicit representation of time ([Elman and Zipser, 1988](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B11) ). Recurrent neural networks (RNNs) were the first models to represent time implicitly, through the effect that is has on processing ([Hopfield, 1982](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B14) ; [Elman, 1990](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B10) ). In the recently developed framework of ‘reservoir’ computing ([Jaeger, 2001](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B15) ; [Maass et al., 2002](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B17) ), a randomly structured RNN non-linearly transforms a time varying input signal into a spatial representation. At each time step, the network combines the incoming stimuli with a volley of recurrent signals containing a memory trace of recent inputs. For a network with *N* neurons, the resulting activation vector at a discrete time *t*, can be regarded as a point in a *N*-dimensional space. Over time, these points form a pathway through the state space also referred to as a *neural trajectory*. A separate read-out layer is trained, with supervised learning techniques, to map different parts of the state space to desired outputs. In real cortical networks, experimental evidence has shown that different stimuli elicit different trajectories while for a given stimuli the activity patterns evolve in time in a reproducible manner ([Broome et al., 2006](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B4) ; [Churchland et al., 2007](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B6) ). Furthermore, identical trials can present a high response variability, but the resulting trajectories are not dominated by noise ([Mazor and Laurent, 2005](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B19) ; [Broome et al., 2006](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B4) ; [Churchland et al., 2007](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B6) ). Reservoir networks do not require classical attractor states and are compatible with the view that cortical computation is based on transient dynamics ([Mazor and Laurent, 2005](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B19) ; [Durstewitz and Deco, 2008](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B9) ; [Rabinovich et al., 2008](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B21) ). It has been shown that neural systems may exhibit transients of long durations which carry more information about the stimulus then the steady states towards which the activity evolves ([Mazor and Laurent, 2005](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B19) ).

Attempts to endow RNNs with unsupervised learning abilities by incorporating biologically plausible local plasticity mechanisms such as spike-timing-dependent plasticity (STDP) ([Markram et al., 1997](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B18) ; [Bi and Poo, 1998](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B3) ) have remained largely unsuccessful (and often unpublished). The problem is most difficult, because structural changes induced by plasticity will impact the network’s dynamics giving rise to altered firing patterns between neurons. These altered firing patterns can further induce changes in connectivity through the plasticity mechanisms and so forth. Understanding and controlling the ensuing self-organization of network structure and dynamics as a function of the network’s inputs is a formidable challenge.

The key to the brain’s solution to this problem may be the synergistic combination of multiple forms of neuronal plasticity. There has been extensive evidence that synaptic learning is accompanied by homeostatic mechanisms. Synaptic scaling regulates the total synaptic drive received by a neuron but maintains the relative strength of synapses established during learning ([Turrigiano et al., 1998](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B24) ). At the same time, intrinsic plasticity (IP) was shown to directly regulate neuronal excitability ([Desai et al., 1999](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B7) ; [Zhang and Linden, 2003](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B27) ). In a RNN, IP induced robust homeostatic effects on the network dynamics ([Steil, 2007](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B23) ; [Schrauwen et al., 2008](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B22) ). But there is only little work combining several forms of plasticity in RNNs ([Lazar et al., 2007](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#B16) ).

In the following, we present a RNN of threshold units combining three different forms of plasticity that learns to efficiently represent and “understand” the spatio-temporal patterns in its input. The SORN model (self-organizing recurrent network) consists of a population of excitatory cells and a smaller population of inhibitory cells (Figure [1](http://www.frontiersin.org/computational_neuroscience/10.3389/neuro.10.023.2009/full#F1) ). The connectivity among excitatory units is sparse and subject to a simple STDP rule. Additionally, synaptic normalization (SN) keeps the sum of an excitatory neuron’s afferent weights constant, while IP regulates a neuron’s firing threshold to maintain a low average activity level. The network receives input sequences composed of different symbols and learns the structure embedded in these sequences in an unsupervised manner. The three types of plasticity mechanisms induce changes in network dynamics which we assess via hierarchical clustering and principal component analysis (PCA). In addition, we train a separate readout layer with supervised learning techniques and compare the performance of our network with that of fixed random networks constructed in the spirit of reservoir computing.

****

**Figure 1. The self-organizing recurrent neural network (SORN) comprises populations of excitatory (blue) and inhibitory (red) cells.**

Directed connections with variable strength between neurons are indicated by black arrows. Some of the excitatory cells also receive external input (light blue). Three forms of plasticity interact to shape the dynamics of the network keeping them in a healthy regime and allowing the network to discover structure in its inputs. A population of readout units is trained with supervised learning methods.

# CHAPTER 7

# WORKING WITH NEURAL NETWORK TOOL BOX

Like its counterpart in the biological nervous system, a neural network can learn and therefore can be trained to find solutions, recognize patterns, classify data, and forecast future events. The behavior of a neural network is defined by the way its individual computing elements are connected and by the strength of those connections, or weights. The weights are automatically adjusted by training the network according to a specified learning rule until it performs the desired task correctly.

Neural Network Toolbox includes command-line functions and graphical tools for creating, training, and simulating neural networks. Graphical tools make it easy to develop neural networks for tasks such as data fitting (including time-series data), pattern recognition, and clustering. After creating your networks in these tools, you can automatically generate [MATLAB](http://www.mathworks.com/products/matlab/)® code to capture your work and automate tasks.

## 7.1 Network Architectures TOOL BOX:

Neural Network Toolbox supports a variety of supervised and unsupervised network architectures. With the toolbox’s modular approach to building networks, you can develop custom architectures for your specific problem. You can view the network architecture including all inputs, layers, outputs, and interconnections.

### Supervised Networks

Supervised neural networks are trained to produce desired outputs in response to sample inputs, making them particularly well-suited to modeling and controlling dynamic systems, classifying noisy data, and predicting future events.

Neural Network Toolbox supports four types of supervised networks:

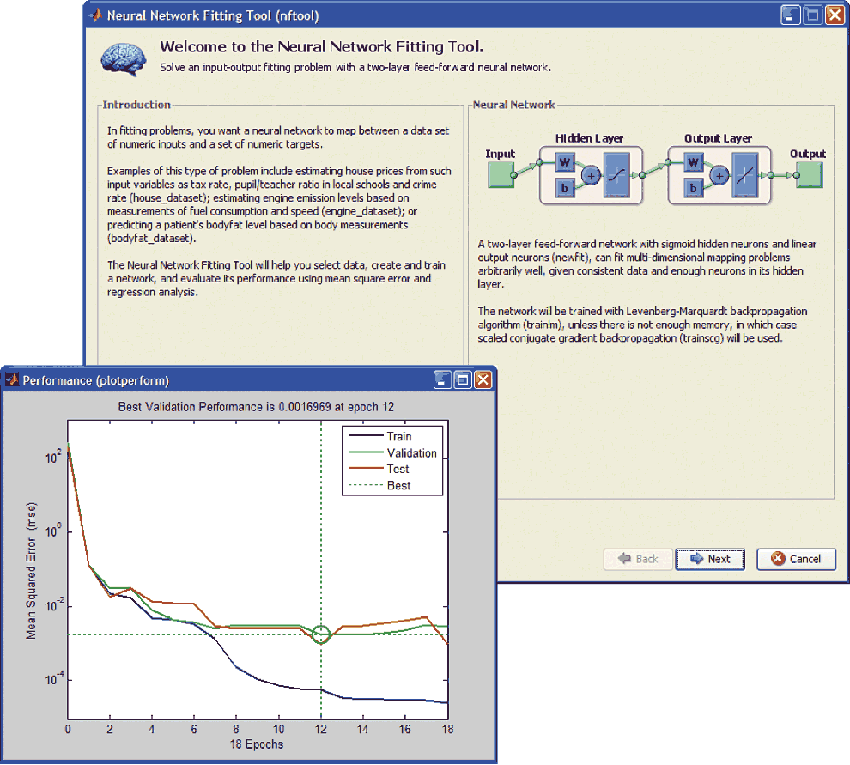
* **Feedforward networks** have one-way connections from input to output layers. They are most commonly used for prediction, pattern recognition, and nonlinear function fitting. Supported feedforward networks include feedforward backpropagation, cascade-forward backpropagation, feedforward input-delay backpropagation, linear, and perceptron networks.
* **Radial basis networks** provide an alternative, fast method for designing nonlinear feedforward networks. Supported variations include generalized regression and probabilistic neural networks.
* **Dynamic networks** use memory and recurrent feedback connections to recognize spatial and temporal patterns in data. They are commonly used for time-series prediction, nonlinear dynamic system modeling, and control systems applications. Prebuilt dynamic networks in the toolbox include focused and distributed time-delay, nonlinear autoregressive (NARX), layer-recurrent, Elman, and Hopfield networks. The toolbox also supports dynamic training of custom networks with arbitrary connections.
* **Learning vector quantization (LVQ)** is a powerful method for classifying patterns that are not linearly separable. LVQ lets you specify class boundaries and the granularity of classification.

### Unsupervised Networks

Unsupervised neural networks are trained by letting the network continually adjust itself to new inputs. They find relationships within data and can automatically define classification schemes.

Neural Network Toolbox supports two types of self-organizing, unsupervised networks:

* **Competitive layers** recognize and group similar input vectors, enabling them to automatically sort inputs into categories. Competitive layers are commonly used for classification and pattern recognition.
* **Self-organizing maps** learn to classify input vectors according to similarity. Like competitive layers, they are used for classification and pattern recognition tasks; however, they differ from competitive layers because they are able to preserve the topology of the input vectors, assigning nearby inputs to nearby categories.

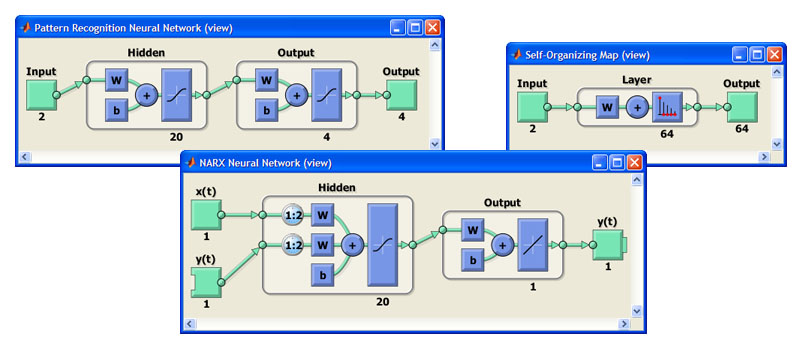


Self-organizing feature maps (SOFM) learn to classify input vectors according to how they are grouped in the input space. They differ from competitive layers in that neighboring neurons in the self-organizing map learn to recognize

neighboring sections of the input space. Thus, self-organizing maps learn both the distribution (as do competitive layers) and topology of the input vectors they are trained on. The neurons in the layer of an SOFM are arranged originally in physical positions according to a topology function. The functions gridtop, hextop or randtop can arrange the neurons in a grid, hexagonal, or random topology.Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist and mandist. Link distance is the most common. These topology and distance functions are described in detail later in this section. Here a self-organizing feature map network identifies a winning neuron using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons as follows. Or Here the *neighborhood* contains the indices for all of the neurons that lie within a radius of the winning neuron . Thus, when a vector is presented, the weights of the winning neuron *and* its close neighbors move toward . Consequently, after many presentations, neighboring neurons will have learned vectors similar to each other. To illustrate the concept of neighborhoods, consider the figure given below. The left diagram shows a two-dimensional neighborhood of radius around neuron . The right diagram shows a neighborhood of radius .

Self-organizing feature maps (SOFM) learn to classify input vectors accordingto how they are grouped in the input space. They differ from competitive layersin that neighboring neurons in the self-organizing map learn to recognize neighboring sections of the input space. Thus, self-organizing maps learn boththe distribution (as do competitive layers) and topology of the input vectors they are trained on. The neurons in the layer of an SOFM are arranged originally in physical positions according to a topology function. The functions gridtop, hextop orrandtop can arrange the neurons in a grid, hexagonal, or random topology.Distances between neurons are calculated from their positions with a distancefunction. There are four distance functions, dist, boxdist, linkdist andmandist. Link distance is the most common. These topology and distance functions are described in detail later in this section.Here a self-organizing feature map network identifies a winning neuronusing the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certainneighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons as follows.

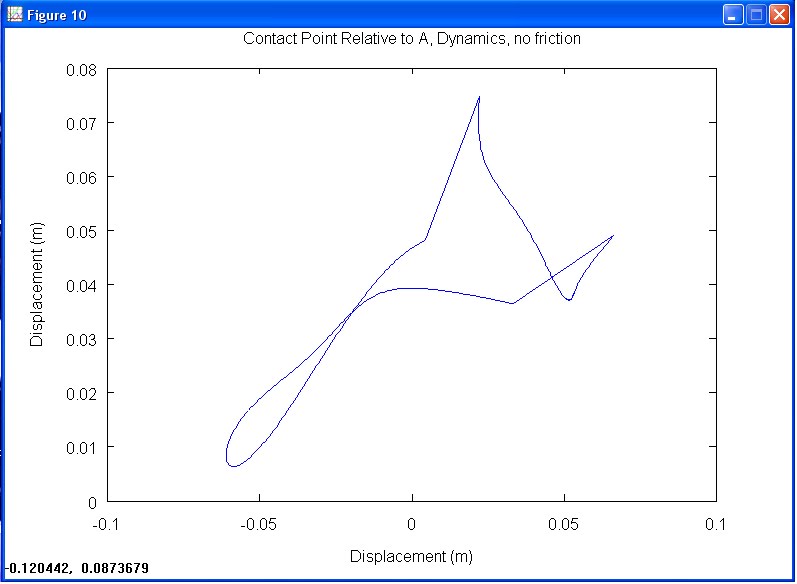
Or Here the *neighborhood* contains the indices for all of the neurons that lie within a radius of the winning neuron .Thus, when a vector is presented, the weights of the winning neuron *and* itsclose neighbors move toward . Consequently, after many presentations, neighboring neurons will have learned vectors similar to each other. To illustrate the concept of neighborhoods, consider the figure given below. Theleft diagram shows a two-dimensional neighborhood of radius aroundneuron . The right diagram shows a neighborhood of radius .



# CHAPTER 8

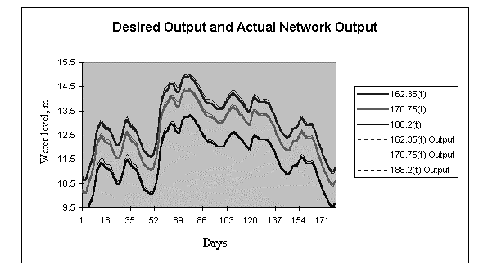
# SIMULATION RESULTS:

**8.1 self organizing neural network:**

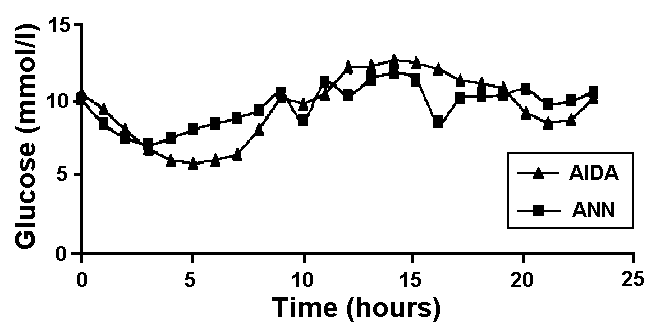
****

. The functions gridtop, hextop or randtop can arrange the neurons in a grid, hexagonal, or random topology.Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist and mandist. Link distance is the most common. These topology and distance functions are described in detail later in this section. Here a self-organizing feature map network identifies a winning neuron using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons as follows.

**8.2 BAYESIAN APPROACH:**

****

. The functions gridtop, hextop or randtop can arrange the neurons in a grid, hexagonal, or random topology.Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist and mandist. Link distance is the most common. These topology and distance functions are described in detail later in this section. Here a self-organizing feature map network identifies a winning neuron using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons as follows.

****

. The functions gridtop, hextop or randtop can arrange the neurons in a grid, hexagonal, or random topology.Distances between neurons are calculated from their positions with a distance function. There are four distance functions, dist, boxdist, linkdist and mandist. Link distance is the most common. These topology and distance functions are described in detail later in this section. Here a self-organizing feature map network identifies a winning neuron using the same procedure as employed by a competitive layer. However, instead of updating only the winning neuron, all neurons within a certain neighborhood of the winning neuron are updated using the Kohonen rule. Specifically, we adjust all such neurons as follows.

# CHAPTER 9

# CONCLUSION:

In this paper we proposed and explored the use of TLRNNs with dynamic trajectory learning for investigating the gene functional patterns with heterogeneous microarray experiments. Results show that the TLRNN works better than the nearest neighbor with Mahalanobis Distances, SVM and SOM. For SVM, this is a little surprise since most well known results usingSVM provided the highest performance, and it has properties of dealing with high level noise and large number of attributes, which both exist in the gene expression data. The possible reasons may be found in the heterogeneous time series gene expression data, since it involved different conditions with time information. Another reason is that

TLRNN can iteratively construct the network, train the weights and update the time information. Results show that the best generalization capability largely depends on the complexity of the patterns in which TLRNN can be monitored by the complexity of the trajectory with distinct types of states. With the increase in the number of gene functional patterns, the generalization performance decreased. However, with changing the number of trajectories and the number of hidden nodes, the performance of the model can be improved based on the statistical criteria for model selection, in which two or three way factorial design can be employed in order to search for the best network architecture for prediction and medical diagnosis.

9.1 REFERENCES:

[WWW.WIKIPEDIAN.COM](http://WWW.WIKIPEDIAN.COM)

ARTIFICIAL NEURAL NETWORKS BY RAO