Feature Extraction from web data using Artificial Neural Networks (ANN)

[1]Manoj Kumar Sharma, [2]Vishal Shrivastav
[1] Research Scholar, M.Tech Arya College Of Engineering and IT
[2] Associate Professor, M.Tech, Arya College Of Engineering and IT

Abstract- The main ability of neural network is to learn from its environment and to improve its performance through learning. For this purpose there are two types of learning **supervised or active learning** – learning with an external 'teacher' or a supervisor who present a training set to the network. But another type of learning also exists : **unsupervised learning[1] .** Unsupervised learning is **self organized learning** doesn't require an external teacher. During training session neural network receives a number of input patterns , discovers significant features in these patterns and learns how to classify input data into appropriate categories. It follows the neuro - biological organization of the brain. These algorithms aim to learn rapidly so learn much faster than back-propagation networks and thus can be used in real time. Unsupervised NN are effective in dealing with unexpected and changing conditions[3].

There are basically two major self – organising networks based learning : Hebbian and competitive learning. We will use Hebbian learning in this paper to visualize that how it can help in feature extraction from any data. We will use input vector weight matrix examples and denote presence of a feature by 1 and absence of a feature by 0. In this method we will see that how features are identified and we can discover patterns in given data. We can use this method for classification and clustering purpose also. Then we will apply this learning rule on web data or content for discovers pattern & extraction of features. Weight increases when same pattern repeats and decrease when it doesn't repeat. The network associates some input x_i with some outputs y_i and y_j because input x_i and x_j coupled during training. But it cannot associate some input x with some output y because that input didn't appear during training and our network has lost the ability to recognize it. Thus, a neural network really can learn to associate stimuli commonly presented together and most important, the network can learn without a teacher. After that we will study applications and limitation of this method. Although it has some limitations but still it is a very useful and fast method that can be used in real time systems where accuracy is bit low but speed in finding association between data is very high for discover patterns and classification of data according to these features[4].

1.ARTIFICIAL NEURAL NETWORKS

One type of network sees the nodes as 'artificial neurons'. These are called artificial neural networks (ANNs). An artificial neuron is a computational model inspired in the natural neurons. Natural neurons receive signals through synapses located on the dendrites or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal though the axon. This signal might be sent to another synapse, and might activate other neurons

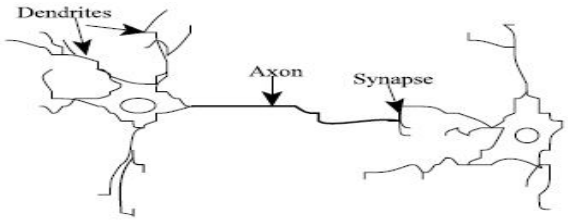


Fig.1 Natural neurons

The complexity of real neurons is highly abstracted when modelling artificial neurons. These basically consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and then computed by a mathematical function which determines the activation of the neuron. Another function (which may be the identity) computes the output of the artificial neuron (sometimes in dependance of a certain threshold). ANNs combine artificial neurons in order to process information[6].

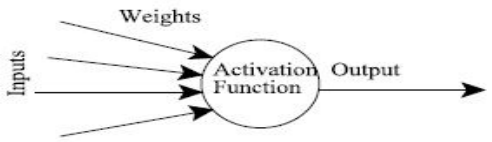


Fig.2 An artificial neuron

The higher a weight of an artificial neuron is, the stronger the input which is multiplied by it will be. Weights can also be negative, so we can say that the signal is inhibited by the negative weight. Depending on the weights, the computation of the neuron will be different. By adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. But when we have an ANN of hundreds or thousands of neurons, it would be quite complicated to find by hand all the necessary weights. But we can find algorithms which can adjust the weights of the ANN in order to obtain the desired output from the network. This process of adjusting the weights is called learning or training.

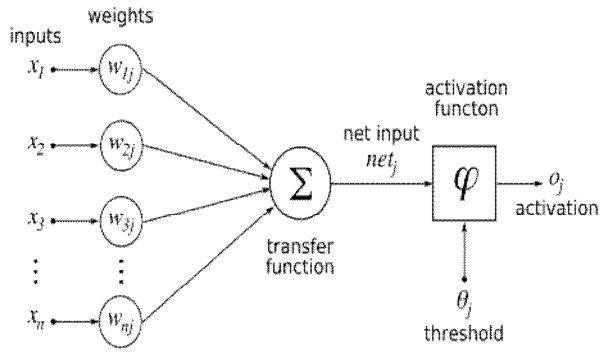


Fig.3 functioning of ANN

The number of types of ANNs and their uses is very high. Since the first neural model by McCulloch and Pitts (1943) there have been developed hundreds of different models considered as ANNs[11].

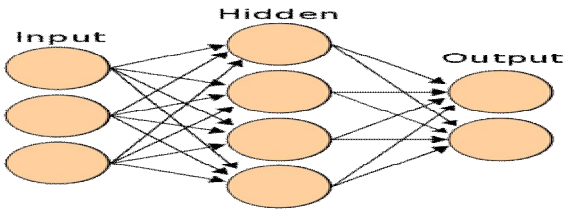


Fig.4 ANN model

The differences in them might be the functions, the accepted values, the topology, the learning algorithms, etc. Also there are many hybrid models where each neuron has more properties than the ones we are reviewing here. Because of matters of space, we will present only an ANN which learns using the backpropagation algorithm (Rumelhart and McClelland, 1986) for learning the appropriate weights, since it is one of the most common models used in ANNs, and many others are based on it. Since the function of ANNs is to process information, they are used mainly in fields related with it. There are a wide variety of ANNs that are used to model real neural networks, and study behaviour and control in animals and machines, but also there are ANNs which are used for engineering purposes, such as pattern recognition, forecasting, and data compression.

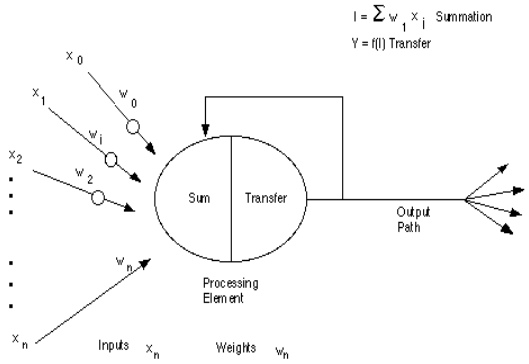


Fig.5 A computational neuron

2. SUPERVISED LEARNING

The Backpropagation Algorithm

It is a supervised learning method, and is a generalization of the delta rule. It requires a dataset of the desired output for many inputs, making up the training set. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backward propagation of errors". Backpropagation requires that theactivation function used by the artificial neurons (or "nodes") be differentiable.

For better understanding, the backpropagation learning algorithm can be divided into two phases: propagation and weight update.

Phase 1: Propagation

Each propagation involves the following steps:
Forward propagation of a training pattern's input through the neural network in order to generate

the propagation's output activations.Backward propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

Multiply its output delta and input activation to get the gradient of the weight.Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.This ratio influences the speed and quality of learning; it is called the *learning rate*. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.Repeat phase 1 and 2 until the performance of the network is satisfactory.

Modes of learning

There are two modes of learning to choose from: One is on-line(incremental) learning and the other is batch learning. In on-line(incremental) learning, each propagation is followed immediately by a weight update. In batch learning, many propagations occur before weight updating occurs. Batch learning requires more memory capacity, but on-line learning requires more updates.

Algorithm

Actual algorithm for a 3-layer network (only one hidden layer):

Initialize the weights in the network (often randomly)

Do

For each example e in the training set

O = neural-net-output(network, e) ; forward pass

T = teacher output for e

Calculate error (T - O) at the output units

Compute delta_wh for all weights from hidden layer to output layer ; backward pass

Compute delta_wi for all weights from input layer to hidden layer ; backward pass continued

Update the weights in the network

Until all examples classified correctly or stopping criterion satisfied

Return the network

As the algorithm's name implies, the errors propagate backwards from the output nodes to the inner nodes. Technically speaking, backpropagation calculates the gradient of the error of the network regarding the network's modifiable weights.[6] This gradient is almost always used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network. Non-linear activation functions that are commonly used include the logistic function, the softmax function, and the gaussian function.

The backpropagation algorithm for calculating a gradient has been rediscovered a number of times, and is a special case of a more general technique called automatic differentiation in the reverse accumulation mode.

It is also closely related to the Gauss–Newton algorithm, and is also part of continuing research in neural backpropagation.

Multithreaded backpropagation

Backpropagation is an iterative process that can often take a great deal of time to complete. When multicore computers are used multithreaded techniques can greatly decrease the amount of time that backpropagation takes to converge. If batching is being used, it is relatively simple to adapt the backpropagation algorithm to operate in a multithreaded manner.

The training data is broken up into equally large batches for each of the threads. Each thread executes the forward and backward propagations. The weight and threshold deltas are summed for each of the threads. At the end of each iteration all threads must pause briefly for the weight and threshold deltas to be summed and applied to the neural network. This process continues for each iteration. This multithreaded approach to backpropagation is used by the Encog Neural Network Framework.

Limitations

The convergence obtained from backpropagation learning is very slow.The convergence in backpropagation learning is not guaranteed.The result may generally converge to any local minimum on the error surface, since stochastic gradient descent exists on a surface which is not flat.Backpropagation learning requires input scaling or normalization.for supervised learning process a teacher is required.

•To train neural network: adjust weights of each unit such that the error between the desired output and the actual output is reduced

•Process requires that the neural network compute the error derivative of the weights (EW)

•it must calculate how the error changes as each weight is increased or decreased slightly[12].

3 .UNSUPERVISED LEARNING

HEBBIAN'S LEARNING FOR FEATURE DETECTION IN GIVEN DATA

We will follow self organising neural networks or unsupervised learning & use Hebbian learning for this purpose. According to Hebb's law if neuron i is near enough to excite neuron j and repeatedly participates in its activation the synaptic connection between these two neurons is strengthened and neuron j becomes more sensitive to stimuli from neuron i. We can represent Hebb's law in the form of two rules as follows :-

i.)If two neurons on either side of connection are activated synchronously then the weight of that connection is increased.

ii.)If two nerons on either side of connection are activated asynchronously, then the weight of that connection is decreased.

Fig 6

We can express the adjustment applied to the weight w_{ij} at iteration p in the following form :

$$\Delta w_{ij}(p) = F[y_j(p), x_i(p)]$$

$y_j(p)$ output from layer j in iteration p

$x_i(p)$ input to layer i in iteration p

where $F[y_j(p),x_i(p)]$ is a function of both postsynaptic and presynaptic activities. As a special case we can represent Hebb's law as follows(Haykin 1999) :

$$\Delta w_{ij}(p) = \alpha y_j(p) x_i(p)$$

Where α is learning rate parameter.

This equation is called activity product rule. It shows how a change in the in the weight of the connection between a pair of neurons is related to a product of incoming and outgoing signals[4].

Hebbian learning implies that weight can only increase. In other words Hebb's law allows the strength of connection to increase, but it doesn't provide a mean to decrease the strength. Thus repeated application of the input signal may drive the weight w_{ij} into saturation. To resolve this problem , we might impose a limit on the growth of weights. It can be done by introducing a non-linear forgetting factor into Hebb's law as follows :

$$\Delta w_{ij}(p) = \alpha y_j(p) x_i(p) - \square y_j(\mathrm{p}) w_{ij}(p)$$

Where □ is forgetting factor

Forgetting factor □ specifies the weight decay in a single learning cycle. It usually falls in the interval between 0 & 1. If the forgetting factor is 0, the neural network is capable only of strengthening its weights, and as a result these weights grow towards infinity. On the other hand, if the forgetting factor is close to 1, the network remembers very little of what it learns. Therefore, a rather small forgetting factor should be chosen, typically between 0.01 and 0.1, to allow only a little forgetting while limiting the weight growth[1].

So the former equation may also be written in the form reffered to as a generalized activity product rule :

$$\Delta w_{ij}(p) = \Phi y_j(p)[\lambda x_i(p) - w_{ij}(p)]$$

Where $\lambda = \alpha / \square$

This rule implies that if the input(log) of neuron(machine) i at iteration p, $x_i(p)$ is less than $w_{ij}(p) / \lambda$, then the modified weight at iteration (p+1) , $w_{ij}(p+1)$, will decrease by an amount proportional to the output of neuron j at iteration p, $y_j(p)$. On the other hand , if $x_i(p)$ is greater than $w_{ij}(p) / \lambda$, then the modified weight at iteration (p+1), $w_{ij}(p+1)$, will increase also in proportion to the output of neuron j, $y_j(p)$ in other words, we can determine the activity balance point for modifying the weights as a variable equal to $w_{ij}(p) / \lambda$. This approach solves the problem of an infinite increase of the weights. So the generalized Hebbian learning algo. Is[8] :

Step 1 : Initialisation

Set initial weights & threshold to small random values, say in an interval [0,1]. Also assign small positive values to the learning rate parameter £ & forgetting factor Ø.

Step 2 : Activation

Compute the neuron output at iteration p

$$y_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j$$

where n is the no. of neurons inputs and θ_j is the threshold value of neuron j.

Step 3 : Learning

Update the weights in network :

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

where $\Delta w_{ij}(p)$ is the weight correction in iteration p. Weight correction is :

$$\Delta w_{ij}(p) = \Phi\, y_j(p)[\lambda x_i(p) - w_{ij}(p)]$$

Step 4 : Iteration

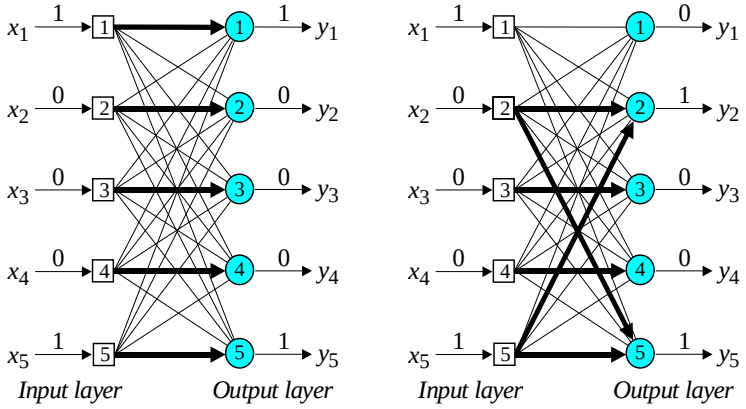
Increase iteration p by 1, go back to step 2 and continue until the weights reach their steady state value.

Hebbian learning example

To illustrate Hebbian learning, consider a fully connected feedforward network with a single layer of five computation neurons. Each neuron is represented by a McCulloch and Pitts model with the sign activation function. The network is trained on the following set of input vectors:

$$\mathbf{X}_1 = \begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix} \quad \mathbf{X}_2 = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \quad \mathbf{X}_3 = \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \quad \mathbf{X}_4 = \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \quad \mathbf{X}_5 = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}$$

Initial and final states of the network



Initial and final weight matrices

Input layer \ Output layer	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Input layer \ Output layer	1	2	3	4	5
1	0	0	0	0	0
2	0	2.0204	0	0	2.0204
3	0	0	1.0200	0	0
4	0	0	0	0.9996	0
5	0	2.0204	0	0	2.0204

A test input vector, or probe, is defined as

$$\mathbf{X} = \begin{bmatrix}1\\0\\0\\0\\1\end{bmatrix}$$

When this probe is presented to the network, we obtain:

$$\mathbf{Y} = sign\left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \begin{bmatrix}1\\0\\0\\0\\1\end{bmatrix} - \begin{bmatrix}0.4940\\0.2661\\0.0907\\0.9478\\0.0737\end{bmatrix} \right\} = \begin{bmatrix}0\\1\\0\\0\\1\end{bmatrix}$$

Learning is often made possible through some notion of which features in the input set are important. But often we don't know in advance which features are important and asking a learning system to deal with raw input data can be computationally expensive. Unsupervised learning can be used as "feature discovery" module that preceds supervised learning.

Consider the above example. The group of ten animals each described by its own set of features, breaks down naturally into three groups : mammals, reptiles and birds. We would like to buid the network that can learn which group a particular animal belongs to, and to generalize so that it can identify animals it has not yet seen.

In this example fig 8 , in input vector X2 and X6 input values x8,x9 and x10 always come together. In input vector X3,X4 and X6 input values x5,x6 and x7 always come together so.they can make an association and can be classified into two different groups. According to features of concerned group they can be classified into reptiles and birds & the rest one is mammals. So according to types and their contents or class and their features network can make an auto association and hence when a new input vector applied to this network, it tries to find association of this data into network and by Hebb's law it extract features from this given data and discover pattern in this data. When some association is found it classify data[13].

.	has hair	has scales	has feathers	flies	lives in water	lay eggs
dog	1	0	0	0	0	0
cat	1	0	0	0	0	0
bat	1	0	0	1	0	0
whale	1	0	0	0	1	0
canary	0	0	1	1	0	1
robin	0	0	1	1	0	1
ostrich	0	0	1	1	0	1
snake	0	1	0	0	0	1
lizard	0	1	0	0	0	1
alligator	0	1	0	0	1	1

Data for unsupervised learning

Fig 8

An *associative memory* is a brain-like distributed memory that learns *associations.* Association is a known and prominent feature of human memory.

Association takes two forms:

Auto-association: Here the task of a network is to *store* a set of patterns (vectors) by repeatedly presenting them to the network. The network subsequently is presented with a partial description or distorted (noisy) version of the original pattern stored in it, and the task is to *retrieve* (*recall*) that particular pattern.

Hetero-association: In this task we want to *pair* an arbitrary set of input patterns to an arbitrary set of output patterns. Auto-association involves the use of unsupervised learning (Hebbian, Hopfield) while hetero-association involves the use of unsupervised (Hebbian) or supervised learning (e.g. MLP/BP) approaches.

Let $\mathbf{x}_k$ denote a *key pattern* applied to an associative memory and $\mathbf{y}_k$ denote a *memorised pattern.* The pattern association performed by the network is described by:

$$\mathbf{x}_k \rightarrow \mathbf{y}_k , \quad k=1,2,\ldots,q$$

Where q is the number of patterns stored in the network. The key pattern $\mathbf{x}_k$ acts as a stimulus that not only determines the storage location of memorised pattern $\mathbf{y}_k$ but also holds the key for its retrieval. In an auto-associative memory, $\mathbf{y}_k = \mathbf{x}_k$, so the input and output spaces have the same dimensionality. In a hetero-associative memory, $\mathbf{y}_k \neq \mathbf{x}_k$, hence in this case the dimensionality of the output space may or may not equal the dimensionality of the input space.

There are two phases involved in the operation of the associative memory:

Storage phase: which refers to the training of the network in accordance with a suitable rule;

Recall phase: which involves the retrieval of a memorised pattern in response to the presentation of a noisy version of a key pattern to the network. Let the stimulus **x** (input) represent a noisy version of a key pattern $\mathbf{x}_i$. This stimulus produces a response **y** (output). For perfect recall, we should find that $\mathbf{y} = \mathbf{y}_i$ where $\mathbf{y}_i$ is the memorised pattern associated with the key pattern $\mathbf{x}_i$. When $\mathbf{y} \neq \mathbf{y}_i$ for $\mathbf{x} = \mathbf{x}_i$, the associative memory is said to have made an *error* in recall.

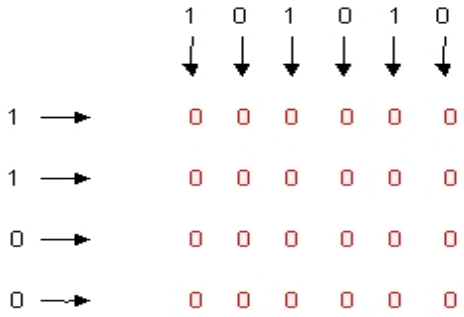
The number q of patterns stored in an associative memory provides a direct measure of the *storage capacity* of the network. In designing an associative memory, we want to make the storage capacity *q* (expressed as a percentage of the total number N of neurons) as large as possible and yet insist that a large fraction of the patterns is recalled correctly.

The net input that arrives to every unit is calculated as:

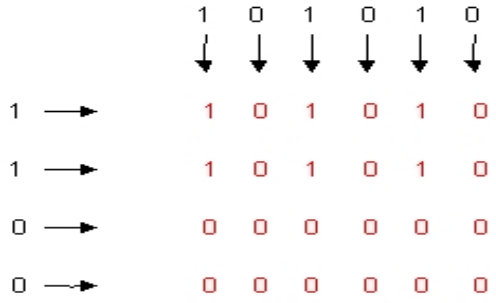
Where i is an output neuron and j an index of an input neuron. The dimensionality of the input space is N and of the output space is M. w_{ij} is the weight from neuron j to neuron i. a_j is the activation of a neuron j.

The activation of each neuron is produced by using a suitable threshold function and a threshold. For example we can assume that the activations are binary (i.e. either 0 or 1) and to achieve this we use the step function[8].

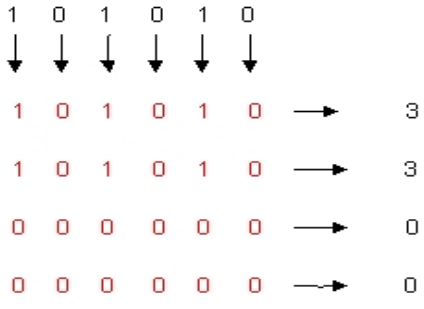
The training of the network takes place by using for example the Hebbian form. Thus what we have is a matrix of weights, with all of them zero initially, assuming an input pattern of (101010) and an output pattern (1100):



If we assume a learning rate η=1 and after a single learning step we get:



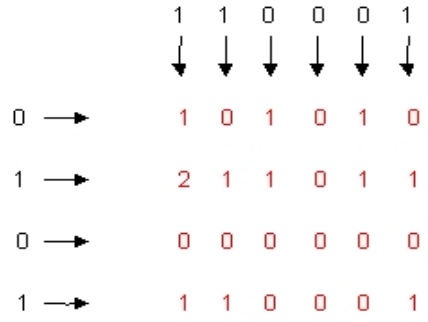
To recall from the matrix we simply apply the input pattern and we perform matrix multiplication of the weight matrix with the input vector. We get in our example:



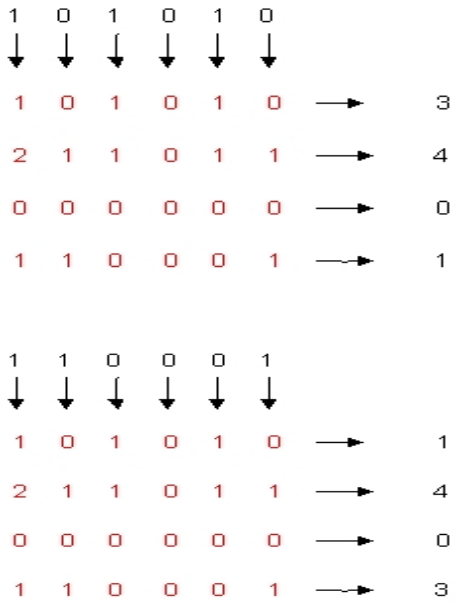
If we assume a threshold of 2 we can get the correct answer (1100) using a step function as activation function.

We can learn multiple associations using the same weight matrix. For

example assume that a new input vector (110001) is given with corresponding output vector as (0101). In this case after a single presentation (with η=1) we will get an updated weight matrix:



Again we can get the correct output vectors when we introduce the corresponding input vector:



Again by using the threshold of 2 and a step function we can get the correct answers of (1100) and (0101).

However, keep in mind that there is only a limited number of patterns which can be stored before perfect recall fails. Typical capacity of an associator network is 20% of the total number of neurons.

Recall accuracy reflects the similarity of a key pattern with the stored patterns. The network can generalise in the sense when an input pattern is not exactly the same with any of the stored patterns, then it returns the (stored) patterns which more closely resembles the input[5].

	mail	jobs	chating	videos	news
Gmail	1	0	1	0	0
Hotmail	1	0	1	0	0
Naukri	0	1	0	0	0
Monster	0	1	0	0	0
Facebook	0	0	1	0	0
Orkut	0	0	1	0	0
Youtube	0	0	0	1	0
Metacafe	0	0	0	1	0
googlenews	0	0	0	1	1
msnindia	0	0	0	1	1

In above data table there are different kind of websites having different type of features and contents. So according to features provided by them they can be divided into various groups. When an input pattern is presented to the network and suppose our network is trined on above data. Now according to positions of input values and their corresponding vector feature and type we can map and associate new input pattern according to its class and identify features hidden in given input. In above example googlenews and msnindia can make a class of news because their last two input values are same so weight between these two increases. Again if an input who has 1 at last or second last position, we can associate with that to News group. Since during training last two inputs of both were coupled. In input vector X1 & X3, input values x1 and x2 comes together so same input repeated at same positions so weight will increase thus new input associate at those positions with highest probability with mail group[7].

Now we will compare artificial neural network with conventional network sysrem. Conventional system is consists of web clients , proxy servers and web servers while ANN consists of input layer, hidden layers and output layer corresponding to client, proxy and server respectively. So we can receive input at web clients corresponding to input layer, then we can do computation at

proxy servers corresponding to hidden layer and collect output at web servers corresponding to output layer. At web clients we can receive different input patterns and according to the features that we want to map, presence of a feature and absence of a feature can be taken 1 & 0 as input values corresponding to input vectors. Then we assign some weights randomly in the range of 0 to 1 to each input value. This calculation is performed between input and hidden layer corresponding to web client and proxy servers. Then we apply hebbian's learning for weight training purpose & use generalized activity product rule between hidden layer and output layer corresponding to proxy and web server. The output we get by this process is different from input in aspects of their pattern and features. So this is a very useful process for pattern discovery, recognition and feature extraction to make association[11].

A neuro-fuzzy system is based on a fuzzy system which is trained by a learning algorithm derived from neural network theory. The (heuristical) learning procedure operates on local information, and causes only local modifications in the underlying fuzzy system._A neuro-fuzzy system can be viewed as a 3-layer feedforward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as (fuzzy) connection weights. It is not necessary to represent a fuzzy system like this to apply a learning algorithm to it. However, it can be convenient, because it represents the data flow of input processing and learning within the model.A neuro-fuzzy system can be always (i.e.\ before, during and after learning) interpreted as a system of fuzzy rules. It is also possible to create the system out of training data from scratch, as it is possible to initialize it by prior knowledge in form of fuzzy rules._The learning procedure of a neuro-fuzzy system takes the semantical properties of the underlying fuzzy system into account. This results in constraints on the possible modifications applicable to the system parameters. A neuro-fuzzy system approximates an n-dimensional (unknown) function that is partially defined by the training data. The fuzzy rules encoded within the system represent vague samples, and can be viewed as prototypes of the training data. A neuro-fuzzy system should not be seen as a kind of (fuzzy) expert system, and it has nothing to do with fuzzy logic in the narrow sense. So the neuro fuzzy approach to solve feature extraction problem from web data will be a very good and efficient method as seen in fig 9.[4]

Properties of pattern associators:

- Generalisation;
- Fault Tolerance;
- Distributed representations are necessary for generalisation and fault tolerance;
- Prototype extraction and noise removal;
- Speed;
- Interference is not necessarily a bad thing (it is the basis of generalisation).

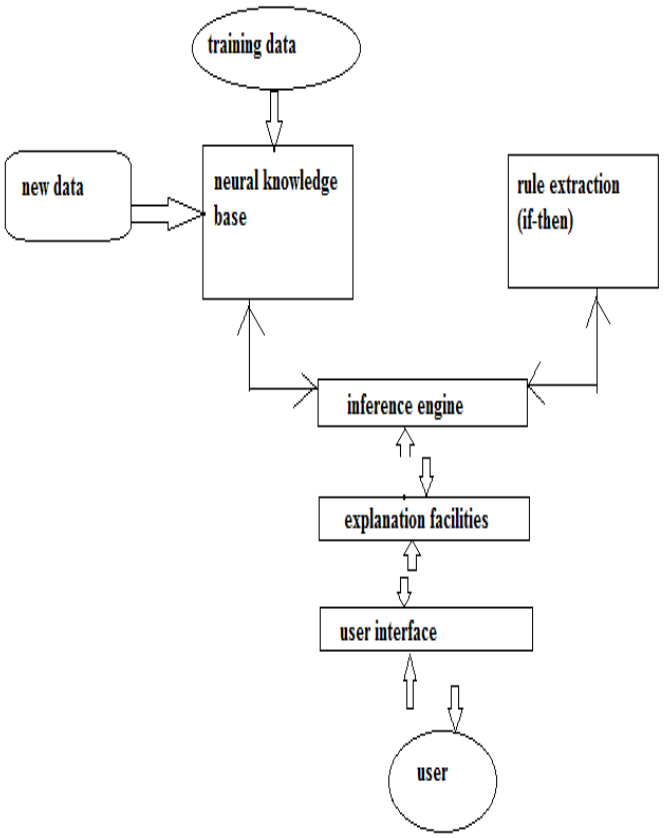


Fig.9 Neural fuzzy expert system

- Hebbian learning is the oldest learning law discovered in neural networks
- It is used mainly in order to build associators of patterns.
- The original Hebb rule creates unbounded weights. For this reason there are other forms which try to correct this problem. There are also temporal forms of the Hebbian rule. A hybrid case is the memory case presented before in the VisNet case.
- It has wide applications in pattern association problems and models of computational neuroscience & cognitive science.

4. CONCLUSION

- If the inputs to a system cause the same pattern of activity to occur repeatedly, the set of active elements constituting that pattern will become increasingly strongly interassociated. That is, each element will tend to turn on every other element and (with negative weights) to turn off the elements that do not form part of the pattern. To put it another way, the pattern as a whole will become 'auto-associated'. We may call a learned (auto-associated) pattern.
- For example, we have heard the word Nokia for many years, and are still hearing it. We are pretty used to hearing Nokia Mobile Phones, i.e. the word 'Nokia' has been associated with the word 'Mobile Phone' in our Mind. Every time we see a Nokia Mobile, the association between the two words 'Nokia' and 'Phone' gets strengthened in our mind. The association between 'Nokia' and 'Mobile Phone' is so strong that if someone tried to say Nokia is manufacturing Cars and Trucks, it would seem odd.
- It can provides us good mapping of information between proxy server and web server and we can maintain good cache as well at proxy level. So end user will be benefited at access of information will be fast.

5. REFERENCES

[1] Becker, S & Plumbley, M (1996). Unsupervised neural network learning procedures for feature extraction and classification. International Journal of Applied Intelligence, 6, 185-203

[2] Mumford, D (1994). Neuronal architectures for pattern-theoretic problems. In C Koch and J Davis, editors, Large-Scale Theories of the Cortex. Cambridge, MA: MIT Press, 125-152.

[3] Artificial neural networks for pattern recognition B YEGNANARAYANA Scidhanci, Vol. 19, P a r t 2, April 1994, pp. 189-238.

[4] Artificial Intelligence: A Guide to Intelligent Systems, 2/E, Michael Negnevitsky. ISBN-10: 0321204662 ISBN-13: 9780321204660. Publisher: Addison-Wesley.

[5] An Introduction to Feature Extraction Isabelle Guyon, and Andr´e Elisseeff

[6] S.Haykin, Neural Networks and Learning Machines,2010,PHI

[7] Xianjun Ni , Research of Data Mining Based on Neural Networks, World Academy of Science, Engineering and Technology ,39,2008, p 381-38

[8] Networks of Neural Computation WK7 – Hebbian Learning Dr. Stathis Kasderidis Dept. of Computer Science University of Crete Spring Semester, 2009.

[9] Leen, T. K. (1991). Dynamics of learning in linear feature-discovery networks. Network,2:85-105

[10] Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks, 2:53-58.

[11] P. Dayan and L. Abbott, Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems, The MIT Press, Cambridge, MA, 2001.

[12] Artificial Neural Networks, Sargur Srihari.

[13] Artificial Intelligence, third edition
Elaine Rich, Kevin Knight, B Nair.

[14] [Chang et al., 2000] H. Chang, D. Cohn, and A. K. McCallum. Learning to create customized authority lists. In Proceedings of the 17th International Conference on Machine Learning, pages 127–134. Morgan Kaufmann, 2000