APPLICATION OF ARTIFICIAL NEURAL NETWORK FOR SPEED CONTROL OF DC SHUNT MOTOR

[1]Subir Bhattacharyya, [2]Poulomi Chatterjee, [3]Chiranjib Mukherjee
[1]Associate Professor, [2]Assistanat Professor, [3]Assistanat Professor
[1]Department of Electrical Engineering,
[1]NSHM Knowledge Campus GOI, Durgapur, India

Abstract: This paper provide an idea about the Artificial Neural Network (ANN) and also to have some knowledge about its application in determining the speed control of a DC motor. In order to provide the speed control of DC motor ANN interpolate no load speed data of a shunt motor at different field current and applied armature voltage in a training data set. The network is trained with C program and MATLAB program. The network predicts the speed corresponding to different sets of field current and armature voltages. In addition to it the network also predict whether the speed is above rated speed.

Index Terms **- Artificial Neural Network, Single layer perceptron, Multi layer perceptron**

I. INTRODUCTION

An **artificial neural network** (ANN) or commonly just **neural network** (NN) is an interconnected group of artificial neurons that uses a mathematical model or computational model for information processing based on a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network

II. HISTORY OF ARTIFICIAL NEURAL NETWORK

The study of the human brain is thousands of years old with the advent of modern electronics. The first step toward ANN come in 1943 when Warren Mc Culloch, a neurophysiologist, and a young mathematician, Walter Pitts, wrote a paper on how neurons might work.Reinforcing this concept of neurons and how they work was a book written by Donald Heb. "The organization of behavior" was written in 1949.Nathanial Rochester from the IBM research laboratories led the first effort to simulate a neural network.In 1956 Dartmouth Summer research project on Artificial Intelligence (AI) provided boos to both artificial intelligence and neural network. One of the outcomes of this process was to simulate research in both the intelligence side. AI, as it is known through out the industry, and in the much lower level neural processing part of the brain. In 1959, Bernard Widrow & Marcian Hoff of Stanford developed models They call ADALINE & MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter, which eliminates echoes on phone line.

III. ANALOGY OF THE BRAIN

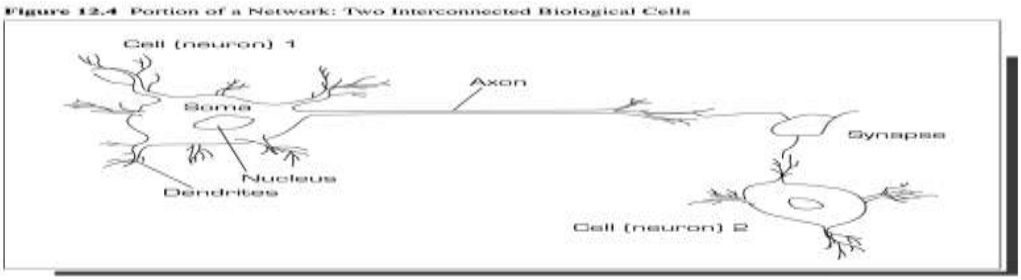


Fig 1: Two Interconnected Biological Cell

Artificial neural networks are electronic model based on the neutral structure of the brain. The exact workings of the human brain are still a mystery. The most basic elements of the human brain are a specific type of cells. It is assumed that these cells provide us our abilities to remember, think & apply previous experiences to our every action. These cells are known, as Neurons & numerically they are 100 billion for human brain. Each of these neurons can connect with up to 200,000 other neurons; although 1,000 to 10,000 are typical. There are over one hundred different classes of neurons, depending on the classification method used together these neurons and their connections from a process, which is not binary, not stable and not

synchronous. the power of the human mind comes from the number of these basic components and the multiple connections between them. it also comes from genetic programming and learning. the individual neurons are complicated. they convey information via electromechanical pathways. the artificial neural networks try to replicate only the most basic elements of this complicated powerful organism. they do it in a primitive

IV. DETAILED DESCRIPTION OF NEURAL NETWORK COMPONENTS & HOW THEY WORK

There is a general understanding of artificial neural network; it is appropriate to explore them in greater deal. But before jumping into the various networks, a more complete understanding of the inner networks is a large class of parallel processing architectures, which are useful in specific type of complex problems. These architectures should not be confused with common parallel processing configurations, which apply many sequential processing units to standard computing topologies. Instead, neural networks are radically different than conventional Von Neumann computers in that they crudely mimic the fundamental properties of human brain.Researchers are working in both the biological & engineering fields to further reveal the key mechanisms for how man learns & reacts to everyday experiences. Improved knowledge in neural processing helps create better artificial networks.Kunihiko Fukushima, a senior research scientist in Japan, describes the give & take of building a neural network model; "We try to follow a physiological evidence as faithfully as possible. For parts not yet clear, however, we construct a hypothesis & build a model that follows hypothesis. We then analyze or simulate the behavior of the model & compare it with that of the brain. If we find any discrepancy in the behaviors between the model & the brain, we change the initial hypothesis & modify the model. We repeat this procedure until the model behaves in the same way as the brain. "This common process has created thousands of network topologies.

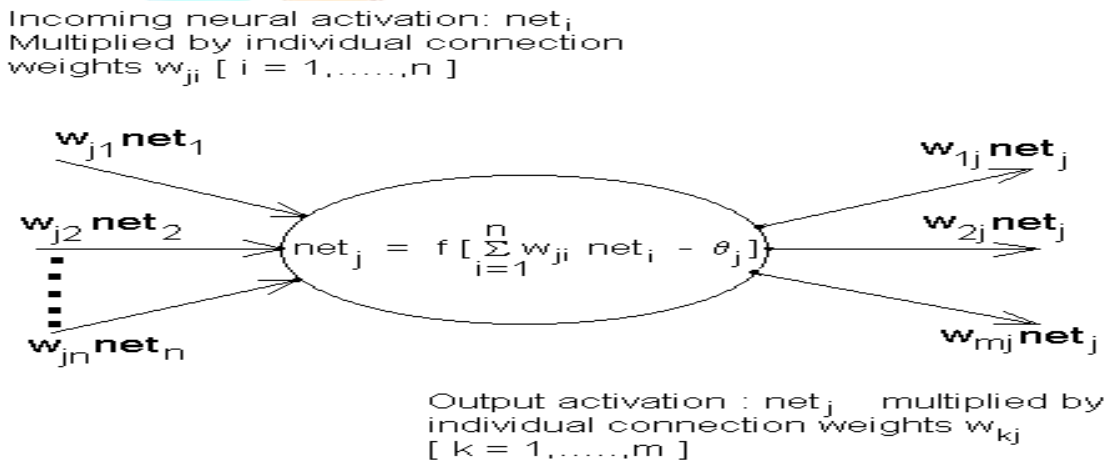


Fig 2:Neural Network and its Components

Inputs, net_i:

Typically, these values are external stimuli from the environment or come from the outputs of other artificial neurons. They can be discrete values from a set, such as {0,1}, or real-valued

Weights, w_i:

These are real-valued numbers that determine the contribution of each input to the neuron's weighted sum and eventually its output. The goal of neural network training algorithms is to determine the best possible set of weight values for the problem under consideration. Finding the optimal set is often a trade-off between computation time and minimizing the network error.

Threshold Θ*:*

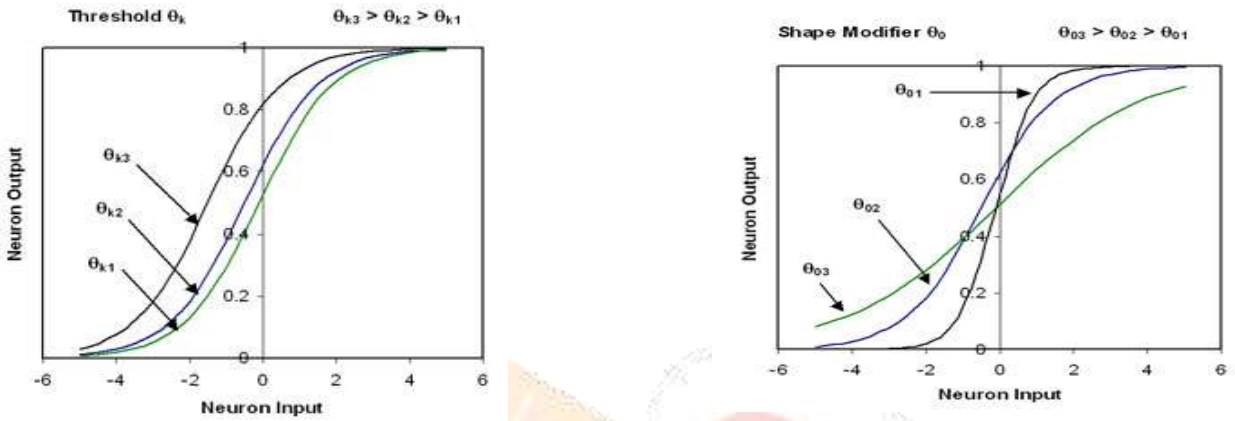
The threshold is a real number that is added algebraically to the weighted sum of the input values. Sometimes the threshold is referred to as a bias value. For simplicity, the threshold can be regarded as another input / weight pair, where $w_i = \Theta$ and net_i=-1.

Activation Function*:*

The activation function for the original McCulloch-Pitts neuron was the unit step function. However, the artificial neuron model has been expanded to include other functions such as the sigmoid, piecewise linear, Gaussian etc, the most popular of which is the sigmoidal function[4].

Sigmoidal Activation Function:

$$o_k = \frac{1}{1+e^{-(net_k+\theta_k)/\theta_O}}$$



(A) (B)

Fig 3: (A)Effect of Threshold modifier ,(B) Effect of Shape Modifier

4.1 Major Components Of Artificial Neural

This section describes the seven major components, which make up an artificial neuron. These components are valid whether the neuron is used for input, output, or is in one of the hidden layers.

1) Weighting Factor: A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input impact that it needs on the processing element's summation function. These weights perform the same function as biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural responds.

2) Summation Function: The first step in a processing element's is to compute the weighted sum of all the inputs. Mathematically, the inputs and their corresponding weights are vectors which can be represent as (i*1*, i*2*…in) and (w*1*, w*2*…wn). The total input signal is the dot or inner product of these two vectors. This summation function can be found by multiplying each components of the *i* vector by corresponding *w* vector and then adding up all the products. Input1= i1 * w1, input2= i2 * w2, etc., are added as input*1* + input*2* + …….input*n*. The result is a single number, not a multy element vector. The summation function can be more complex than the simple input and the weight sum of the products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. Some summation functions have an additional process applied to the result before it is passed on to the transfer function.

3) Transfer Function: The result of the summation function, almost always the weighted sum, is transformed to a working output through an algorithmic process known as the Transfer Function. In transfer function the summation can be compared with some threshold to determine the neural output. If the some is greater than the threshold value, the processing element generates a signal. If the some is less than the threshold value, no signal is generated. Both types of response are significant. The threshold or transfer function is generally non-linear. Linear functions are limited because the output is proportional to the input.

are shown in the fig. Below:

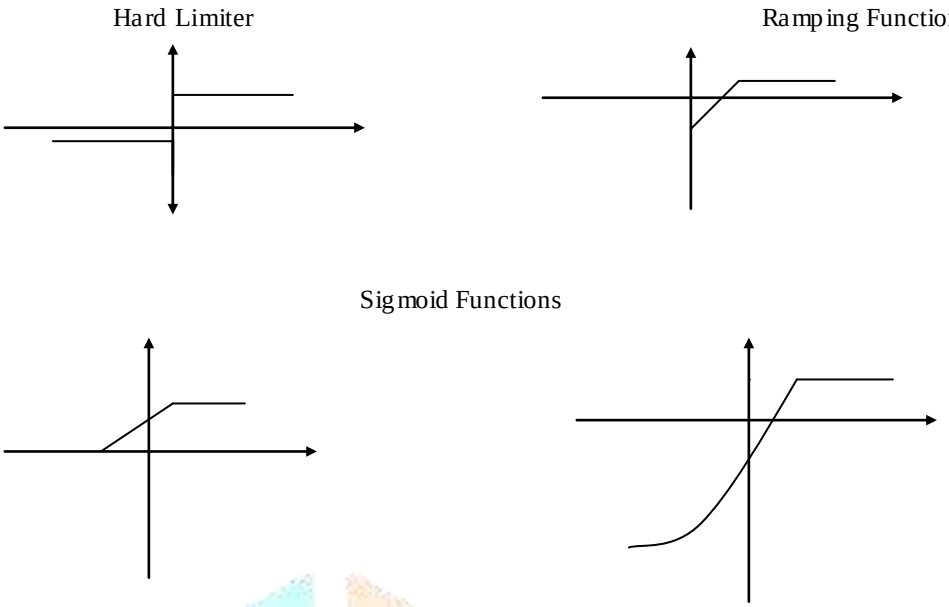


Fig 4: Different types of transfer functions

4) Scaling and Limiting: After processing element's transfer function, the result can pass through additional process, which scale and limit. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism, which, ensure that the scaled result doesn't exceed an upper, or lower bound.

5) Output Function (Competition): Each processing element is allowed one output signal, which, it may output to hundreds of other neurons. Normally the output is directly equivalent to the transfer function's result. Competition can occur at one or both of the two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning pr adaptation process.

6) Error Function and Back Propagated Value: In most learning networks the difference between the current output and the desired output is calculated. This error then transferred by the error function to match particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, cube the error to fit their specific function. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error.

4.2 Training Of Artificial Neural Network

Once a network has been structured for a particular application, that network is ready to be trained. To start this process the initial weights are chosen randomly, then the training or learning begins. There are two approaches training - "Supervised" and "Unsupervised". Supervised training involves a mechanism of providing the network with the desired output either by manually grading the network's performance or by providing the desired outputs with the inputs. Unsupervised training is where the network has to make sense of the inputs without outside. Most of networks utilize supervised training. Unsupervised training is used to perform some initial characterization on inputs.

1) **Supervised Training:**

In supervised training, both the inputs and outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated[3] back through the system, causing the system to adjust the weights, which control the network. This process occurs over and over as the weights are continually tweaked. The set of data which enables this training of a network the same set of data is processed many times as the connection weights are ever refined. In project we have made use of supervised training. The current commercial network development packages provide tools to monitor how well an artificial neural is converging on the ability to predict the right answer. These tools allow the training process to go on for days, stopping only when the system reaches some desired point, or accuracy. However, some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also do not converge if there is not enough data to enable complete learning. Ideally there should be enough data so that part of the data can be held back as a test.

If a network simply cannot solve the problem, the designer then has to review the inputs and outputs, the number of layers, the summation, transfer and training functions and even the initial weights themselves. Those changes required to create a successful network constitute a process wherein the art of neural networking occurs.

Another part of the designer's creativity governs the rules of training. There are many laws (algorithms) used to implement the adaptive feedback required to adjust the weights during training. The most common technique is backward-error propagation, more commonly known as back-propagation. Yet, training is not just a technique. It involves a feel and conscious analysis to

ensure that the network is not over trained. Initially, an artificial neural network configures itself with general statistical trends of the data. Later, it continues to learn about other aspects of the data, which may be spurious from a general viewpoint.
When finally the system has been correctly trained, and no further learning is needed, the weights can, if desired, be frozen. In some systems this finalized network is then turned into hardware so that it can be fast. Other systems do not lock themselves in but continue to learn while in production use.

2) **Unsupervised or Adaptive Training:**
In unsupervised training the network is provided with inputs but not with desired outputs. The system itself must then decide what feature it will use to group the input data. This is often referred to as self-organization or adaptation. At the present time, unsupervised training is not well understood. This adaptation to the environment is the promise, which would enable science fiction types of robots to continually learn on their own as they encounter new situations and new environments.

Life is filled with situations where exact training sets do not exist. Some of these situations involve military actions where new combat techniques and weapons might be encountered. Because of these unexpected aspects to life and the human desire to be prepared, there continues to be research into, and hope for, this field. Yet, at the present time, the vast bulk of neural network work is in systems with supervised learning. Supervised learning is achieving results. One of the leading researches into supervised learning is Tuevo Kohonen, an electrical engineer at the Helsinki University of Technology. He developed a self-organizing network, sometimes called an auto-associator, that learns without the benefit of knowing the right answer. It is an unusual looking network in that it contains one single layer with many connections. The weights for those connections have to be initialized and the input has to be normalized. The neurons are set up to compete in a winner-take-all fashion. Kohonen continues his research into networks that are structured differently than standard feed forward back-propagation approaches[2]. Kohonen's work deals with the grouping of neurons into field. Neurons within a field are topologically ordered. Topology is a branch of mathematics that studies how to map from one space to another without changing the geometric configuration. The three-dimensional groupings, often found in mammalian brains, are examples of topological ordering.

Kohonen has pointed out that the lack of technology in neural network models makes today's neural networks just simple abstractions of the real neural networks within the brain. As this research continues more powerful self-learning networks may possible. But currently this field remains one that is still in the laboratory.

4.2.1 Single Layer Perceptrons

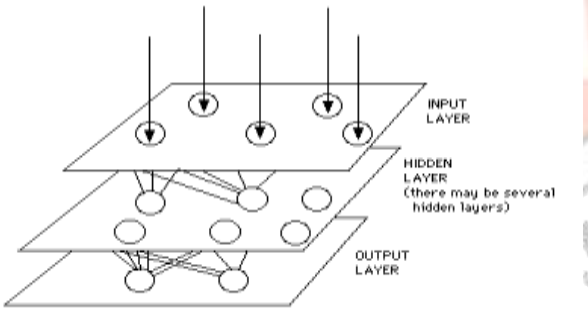


Fig 5: Single Layer Perceptron

The perceptron is the simplest form of a neutral network used for the classification of patterns said to be linearly separable. Basically it consists of a single neutron with adjustable synaptic weights & bias. The algorithm used to adjust the free parameters of this neural network first appeared in a Learning procedure developed by Rosenblatt (1958, 1962) for his perceptron brain model. Indeed Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable class, then the perceptron algorithm converges & positions the decision surface in theform of a hyper plane between the two classes. The proof of convergence of the algorithmic known as the Perceptron convergence theorem[1] .The perceptron built around a single neutron is limited to performing pattern classification with only two classes. By expanding the output layer of the perceptron to include more than one neuron, We may correspondingly from classification with more than two classes. However, the classes have to bilinear separable for the perceptron to work properly. The important point is that insofar as the basis theory of the perceptron as a pattern classifier is concerned ,we need consider only the case of a Single neuron. The extension of the theory tothe case of more than one neuron is trivial. The single neuron also forms the basis of an adaptive filter, a functional block that is the basic to the ever-expanding subject of signal processing.

4.2.1 Multilayer Perceptron

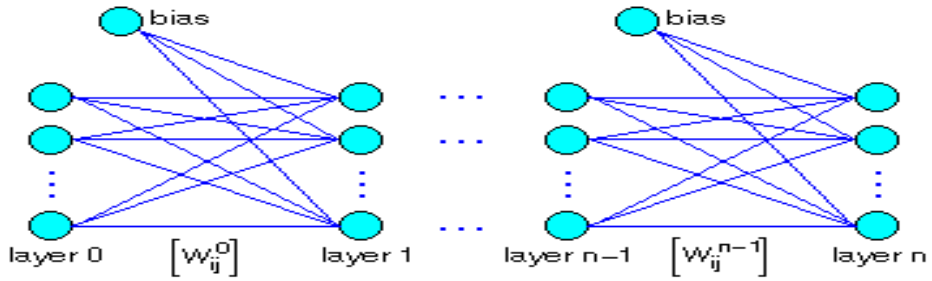


Fig 6: Multi Layer Perceptron

Multiplayer perceptron network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers of computation nodes & an output layer of computation nodes. The input signal propagates through the network in forward direction, on a layer-by-layer basis. The neural networks are commonly referred to as multiplayer perceptrons (MLPs), which represents a generalization of the single-layer perceptron. Multiplayer perceptrons have been applied successfully to solve some difficult & diverse problems by training them in a supervised manner with a highly popular algorithm known as the error back-propagation algorithm. This algorithm is based on the error correction-learning rule. Basically, error back-propagation learning consists of two passes through the different layers of the network: a forward pass & backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network & its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error –correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network, against the direction of synaptic connections-hence the name "error back propagation". The synaptic weights are adjusted to make the actual response of the network move closer to the desired response in a statistical sense. The error back-propagation algorithm is also referred to in the literature as the back-propagation algorithm, or simply back-prop. Henceforth we will refer to it as the back-propagation algorithm. The learning process performed with the algorithm is called back-propagation learning. A mutilayer perceptron has three distinctive characteristics:

(1) The model of each neuron in the network includes a nonlinear activation function. The important point to emphasize here is that the nonlinearity is smooth (i.e. differentiable everywhere), as opposed to hard limiting used in Rosenblatt's perceptron. A commonly used form of nonlinearity that satisfy this requirement is a Sigmoidal nonlinearity-defined by the logistic function:

$$Yj = (1/(1 + \exp(-vj)))$$

Where, vj = Induced local field (i.e. the weighted sum of all synaptic inputs + bias) of neuron j, and Yj is the output of the neuron. The presence of non-linearities is important because otherwise the input-output relation of the network could be reduced to that of a single-layer perceptron. Moreover, the use of the logistic function is biologically motivated, since it attempts to account for the refractory of real neurons.

(2) The neurons contain one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns (vector).

(3) The network exhibits a high degree of connectivity, determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

V.TYPES OF NEURAL NETWORK

Feed forward Network

The feedforward neural networks are the first and arguably simplest type of artificial neural networks devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

Recurrent Network

Contrary to feedforward networks, recurrent neural networks (RNs) are models with bi-directional data flow. While a feedforward network propagates data linearly from input to output, RNs also propagate data from later processing stages to earlier stages.

Simple recurrent network

A *simple recurrent network* (SRN) is a variation on the multi-layer perceptron, sometimes called an "Elman network" due to its invention by Jeff Elman A three-layer network is used, with the addition of a set of "context units" in the input layer. There are connections from the middle (hidden) layer to these context units fixed with a weight of one. At each time step, the input is propagated in a standard feed-forward fashion, and then a learning rule (usually back-propagation) is applied. The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction that are beyond the power of a standard multi-layer perceptron.

In a *fully recurrent network*, every neuron receives inputs from every other neuron in the network. These networks are not arranged in layers. Usually only a subset of the neurons receive external inputs in addition to the inputs from all the other neurons, and another disjunction subset of neurons report their output externally as well as sending it to all the neurons. These distinctive inputs and outputs perform the function of the input and output layers of a feed-forward or simple recurrent network, and also join all the other neurons in the recurrent processing.

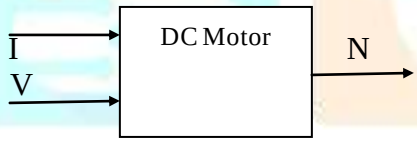
Hopfield network

The Hopfield network is a recurrent neural network in which all connections are symmetric. Invented by John Hopfield in 1982, this network guarantees that its dynamics will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration. Hopfield networks can be created with the function newhop.

V. RESEARCH METHDOLOGY

SPEED CONTROL OF DC SHUNT MOTOR BY ARTIFICIAL NEURAL NETWORK

The DC shunt motor is an electromechanical device which transfer electric energy into mechanical energy. In shunt motor the field is connected across the armature .The circuit diagram is shown below



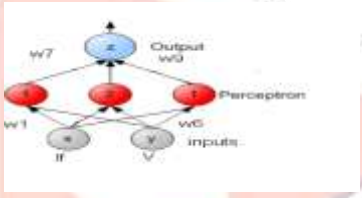


Fig 6 : Speed control of Dc motor by ANN

From the above circuit diagram we get

V=I(a)R(a)+Eb

Eb=V-I(a)R(a) Where Eb is the back emf.

PNZϕ/60a=V-I(a)R(a)

Speed Equation Of DC machine

N=V-I(a)*R(a)/Kϕ Where K=60a/PZ

Thus the speed of the DC motor can be controlled by controlling one of the two parameters If & V or both. To calculate the speed of the machine, we need to know the unknown machine parameters as indicated in the speed equation. But this is a laborious process & not suitable for quick calculation. So to calculate the speed in an easy manner we can introduce the idea of Artificial Neural Network.

VI. Process Of Speed Control By Ann

Algorithm

Step -1 : Define $\mathbf{V_i}$, $\mathbf{I_i}$, and corresponding $\mathbf{N_i}$.
(I = 1 , 2 , 3… 25)

Step - 2: Define all W_j (j = 1 , 2…9)

Step - 3: Find $\mathbf{W_j' = W_j + pW}$ (where initially $\mathbf{pW = 0}$)

Step - 4: Using output formula to find N_i' corresponding to V_i and I_i and calculate

$$\mathbf{e_i^2 = (N_i' - N_i)^2}$$

Define $\mathbf{e^2 = \sum_{i=1}^{15} e_i^2}$

Step-5: Check whether $\mathbf{e}^2$ <= **0.02** , if it is , break the loop , and go to step - 1 .

Step-6: Find pW using formula for all weights .

Setp-7: Go to step - 3 .

Step-8: **Final weights** are **Wj'** (j = 1 , 2 ,… 9) .

6.1 Training set of data

Table 1: Training Data

Serial No	Field Current(IF) Ampere	Armature Voltage Va (Volt)	Speed(n) RPM	N = n/2000
1.	0.25	150	838	0.419
		160	925	0.4625
		180	1058	0.529
		200	1170	0.585
		230	1320	0.66
2.	0.22	150	955	04775
		160	1002	0.501
		180	1116	0.555
		200	1202	0.601
		220	1302	0.651
3.	0.20	150	1022	0.511
		160	1072	0.536
		180	1220	0.61
		200	1340	0.67
		220	1492	0.746
4.	0.19	150	1250	0.625
		160	1270	0.635
		180	1320	0.66
		200	1450	0.725
		220	1590	0.795
5.	0.18	150	1200	0.60
		160	1296	0.648
		180	1480	0.74
		200	1672	0.836
		220	1840	0.92

6.1.1 Training the data by C Programming and MATLAB

6.1.1.1C PROGRAM ON SINGLE LAYER PERCEPTRONS :

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 void main()
{
clrscr();
float c[35],v[35],n[35];
float vc=248.00,ic=0.4378,ip=0.2,vp=220;
 float a,b;
 int i=1;
c[1]=0.18;
c[2]=0.18;
c[3]=0.18;
c[4]=0.18;
c[5]=0.18;

c[6]=0.19;
c[7]=0.19;
c[8]=0.19;
c[9]=0.19;
c[10]=0.19;

c[11]=0.20;
c[12]=0.20;
c[13]=0.20;
c[14]=0.20;
c[15]=0.20;

c[16]=0.22;
c[17]=0.22;
c[18]=0.22;
c[19]=0.22;
c[20]=0.22;

c[21]=0.25;
c[21]=0.25;
 c[21]=0.25;
 c[21]=0.25;
 c[21]=0.25;

 v[1]=150;
 v[2]=160;
 v[3]=180;
 v[4]=200;
 v[5]=220;

 v[6]=150;
 v[7]=160;
 v[8]=180;
 v[9]=200;
 v[10]=220;

 v[11]=150;
 v[12]=160;
 v[13]=180;
 v[14]=200;
 v[15]=220;

 v[16]=150;
 v[17]=160;
```

```
v[18]=180;
v[19]=200;
v[20]=220;

v[21]=150;
v[22]=160;
v[23]=180;
v[24]=200;
v[25]=220;

n[1]=0.600;
n[2]=0.648;
n[3]=0.740;
n[4]=0.836;
n[5]=0.920;
n[6]=0.625;
n[7]=0.635;
n[8]=0.660;
n[9]=0.725;
n[10]=0.795;

n[11]=0.511;
n[12]=0.536;
n[13]=0.610;
n[14]=0.670;
n[15]=0.746;

n[16]=0.478;
n[17]=0.501;
n[18]=0.558;
n[19]=0.601;
n[20]=0.651;

n[21]=0.419;
n[22]=0.463;
n[23]=0.529;
n[24]=0.585;
n[25]=0.660;

b=((vp*vc)+(ip*ic));
for(i=1;i<=25;i++)
{
   a=(((v[i]*vc)+(c[i]*ic))-b);
   if(a>0.0)
   printf("\n\tTHE VALUE IS ABOVE RATED SPEED");
   else
   printf("\n\tTHE VALUE IS BELOW RATED SPEED");
   }
    getch();
    }
```

6.1.1.2 C PROGRAM ON MULTILAYER PERCEPTRONS:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 void main()
{
clrscr();
float c[35],v[35],n[35],s[35],y[35], w[35],e[35],t[35];
int i,kk=0;
float f=0.0;
float p,ep=0.0,k=0.01;
c[1]=0.18;
c[2]=0.18;
```

```
c[3]=0.18;
c[4]=0.18;
c[5]=0.18;

c[6]=0.19;
c[7]=0.19;
c[8]=0.19;
c[9]=0.19;
c[10]=0.19;

c[11]=0.20;
c[12]=0.20;
c[13]=0.20;
c[14]=0.20;
c[15]=0.20;

c[16]=0.22;
c[17]=0.22;
c[18]=0.22;
c[19]=0.22;
c[20]=0.22;

c[21]=0.25;
c[21]=0.25;
 c[21]=0.25;
 c[21]=0.25;
 c[21]=0.25;

 v[1]=150;
 v[2]=160;
 v[3]=180;
 v[4]=200;
 v[5]=220;

 v[6]=150;
 v[7]=160;
 v[8]=180;
 v[9]=200;
 v[10]=220;

 v[11]=150;
 v[12]=160;
 v[13]=180;
 v[14]=200;
 v[15]=220;

 v[16]=150;
 v[17]=160;
 v[18]=180;
 v[19]=200;
 v[20]=220;

 v[21]=150;
 v[22]=160;
 v[23]=180;
 v[24]=200;
 v[25]=220;

 n[1]=0.600;
 n[2]=0.648;
 n[3]=0.740;
 n[4]=0.836;
 n[5]=0.920;

 n[6]=0.625;
 n[7]=0.635;
```

```
n[8]=0.660;
n[9]=0.725;
n[10]=0.795;

n[11]=0.511;
n[12]=0.536;
n[13]=0.610;
n[14]=0.670;
n[15]=0.746;

n[16]=0.478;
n[17]=0.501;
n[18]=0.558;
n[19]=0.601;
n[20]=0.651;

n[21]=0.419;
n[22]=0.463;
n[23]=0.529;
n[24]=0.585;
n[25]=0.660;
for(i=1;i<=9;i++)
{
   f=0.1;
    w[i]=f;
   }
   f=0.0;
while(f==0.0)
 {
for(i=1;i<25;i++)
{
s[1]=c[i]*w[1];
s[2]=v[i]*w[4];
s[3]=s[1]+s[2];
y[1]=(1/(1+exp(-s[3])));

s[4]=c[i]*w[2];
s[5]=v[i]*w[5];
s[6]=s[4]+s[5];
y[2]=(1/(1+exp(-s[6])));

s[7]=c[i]*w[3];
s[8]=v[i]*w[6];
s[9]=s[7]+s[8];
y[3]=(1/(1+exp(-s[9])));

s[10]=y[1]*w[7];
s[11]=y[1]*w[8];
s[12]=y[3]*w[9];
s[13]=s[11]+s[12]+s[13];
t[i]=(1/(1+exp(-s[13])));
p=(k*(t[i]-n[i])*(1-t[i])*t[i]);

w[7]=(w[7]-(p*(y[1])));
w[8]=(w[8]-(p*(y[2])));
w[9]=(w[9]-(p*(y[3])));
w[1]=((w[1])-(p*(w[7])*(y[1])*(1-(y[1]))*(v[i])));
w[2]=((w[2])-(p*(w[8])*(y[2])*(1-(y[2]))*(v[i])));
w[3]=((w[3])-(p*(w[9])*(y[3])*(1-(y[3]))*(v[i])));
w[4]=((w[4])-(p*(w[7])*(y[1])*(1-(y[1]))*(c[i])));
w[5]=((w[5])-(p*(w[8])*(y[2])*(1-(y[2]))*(c[i])));
w[6]=((w[6])-(p*(w[9])*(y[3])*(1-(y[3]))*(c[i])));

e[i]=((t[i]-n[i])*(t[i]-n[i]));
ep=(ep+e[i]);
 }
```

```
   kk++;
   if(kk%3000==0)
   printf("\n\n\tThe value of ep is%f\n",ep);

   if(ep<4.0)
  f=1.0;
  else
    ep=0;
}

 printf(" \nthe error is%4f",ep);

 printf("\nthe weights are w[1]=%f",w[1]);
 printf("\nw[2]=%f",w[2]);
  printf("\nw[2]=%f",w[2]);
  printf("\nw[3]=%f",w[3]);
  printf("\nw[4]=%f",w[4]);
  printf("\nw[5]=%f",w[5]);
  printf("\nw[6]=%f",w[6]);
  printf("\nw[7]=%f",w[7]);
  printf("\nw[9]=%f",w[9]);
     getch();
     }
```

6.1.1.3 OUTPUT PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
 void main()
{
clrscr();
float c,v,t;
int a;
float s[25],w[25],y[25];
printf("Enter the value of current::");
scanf("%4f",&c);
printf("Enter the value of voltage::");
scanf("%4f",&v);
 w[1]=0.0;
 w[2]=0.01;
 w[3]=0.01;
 w[4]=0.01;
 w[5]=0.01;
 w[6]=0.01;
 w[7]=0.01;
 w[8]=0.01;
 w[9]=0.01;
 s[1]=c*w[1];
 s[2]=v*w[4];
 s[3]=s[1]+s[2];
 y[1]=(1/(1+exp(-s[3])));

 s[4]=c*w[2];
 s[5]=v*w[5];
 s[6]=s[4]+s[5];
 y[2]=(1/(1+exp(-s[6])));

 s[7]=c*w[3];
 s[8]=v*w[6];
 s[9]=s[7]+s[8];
 y[3]=(1/(1+exp(-s[9])));

 s[10]=y[1]*w[7];
```

```
 s[11]=y[1]*w[8];
 s[12]=y[3]*w[9];
 s[13]=s[11]+s[12]+s[13];
 t=(1/(1+exp(-s[13])));
 a=(t*2000);
 printf("the required speed is::%d",a);
getch();
    }
```

6.1.2 MATLAB PROGRAMMING

```
clear all
clc
net1 = newelm([23 66;-38 -0.05],[40 1],{'tansig','purelin'},'traingdx');
net1.trainparam.show=500;
net1.trainparam.lr=0.005;
net1.trainparam.epochs=2500;
net1.trainparam.goal=1e-5;
net1.trainparam.min_grad=1e-20;
net1.trainparam.mu_max=1e20;
load sawan_train.txt;
load sawan_test.txt;
p3=sawan_test(:,1:2);
t3=sawan_test(:,3);
p3=p3';
p3seq=con2seq(p3);
t3=t3';
t3seq=con2seq(t3);
p4=sawan_train(:,1:2);
t4=sawan_train(:,3);
p4=p4';
t4=t4';
p4seq=con2seq(p4);
net1=train(net1,p3seq,t3seq);
%net=newpnn(p1,t1);
a1=sim(net1,p4seq);
z1=seq2con(a1);
z1=z1';
z1{1,1};
a1=z1{1,1};
a1=a1';
b1=t4';
test_result1(:,1)=b1(:,1);
test_result1(:,2)=a1(:,1);
test_result1
```

VII. Result

Table 2: Result Data

Serial No	Field Current(IF) Ampere	Armature Voltage Va (Volt)	Speed(n) RPM	**N = n/2000**	**N (measured) By ANN[C program]**
1.	0.25	150	838	**0.419**	**0.42**
		160	925	**0.4625**	**0.453**
		180	1058	**0.529**	**0.512**
		200	1170	**0.585**	**0.555**
		230	1320	**0.66**	**0.62**
2.	0.22	150	955	**04775**	**0472**
		160	1002	**0.501**	**0.482**
		180	1116	**0.555**	**0.53**
		200	1202	**0.601**	**0.62**
		220	1302	**0.651**	**0.64**
3.	0.20	150	1022	**0.511**	**0.521**
		160	1072	**0.536**	**0.521**
		180	1220	**0.61**	**0.60**
		200	1340	**0.67**	**0.65**
		220	1492	**0.746**	**0.723**
4.	0.19	150	1250	**0.625**	**0.598**
		160	1270	**0.635**	**0.601**
		180	1320	**0.66**	**0.623**
		200	1450	**0.725**	**0.70**
		220	1590	**0.795**	**0.76**
5.	0.18	150	1200	**0.60**	**0.58**
		160	1296	**0.648**	**0.65**
		180	1480	**0.74**	**0.71**
		200	1672	**0.836**	**0.81**
		220	1840	**0.92**	**0.89**

VIII. ACKNOWLEDGEMENT

1. Speed measurements could have been done by ammeter and voltmeter readings, which is easy using ANN concept required basic software knowledge because the black box contains software program.

2. Since the training data is generated from the voltmeter and ammeter readings, it is itself not free from errors. Subsequently the predicted result, which relies on the training set, becomes erroneous.

3. If the network choosing is not proper then the error instead of converging would go on diverging.

4. If the program over learns itself with the training data given, then it spreads out over a wide range where a number of local minima exists at the cost of single global minima. Working with these numbers local minima gives erroneous result.

IX. REFERENCES

[1] Christophe Amerijckx et.al, "Image Compression by Self-organized Kohonen maps", IEEE Trans. on Neural Networks , Vol.9, No.3, May 1998

[2] A.K. Krishnamurthy et al., "Neural Networks for VQ of speech and Images", IEEE Jl. on selected areas in comm., Vol.8, No.8, Oct 1990

[3] R.P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, April 1987, pp 4-21

[4] P. V. Rao, Suhas Madhusudana, Nachiketh S S, and Kusuma Keerthi, "Image Compression using Artificial Neural Networks", Second International Conference on Machine Learning and Computing, ICMLC2010, Feb 2010, Bangalore, India

[5] Takahashi and Y. Ohmori, "High-performance direct torque control of induction motor," IEEE Trans. Ind. Appl., vol. 25, no. 2, pp. 257–264,1989

.