**MCS 105**

**Computational Intelligence**

**Report**

**on**

**TWO LAYER NEURAL NETWORK**

**Submitted By :**

**Abhishek Sen**

**Ruman Saleem**

**Anju**

**INTRODUCTION**

As the name suggests, ANNs have a biological motivation, and we briefly look at that first. Following this, we look in detail at how information is represented in ANNs, then we look at the multi-layer networks. Let us begin this with defining ANN.

1. **ARTIFICIAL NEURAL NETWORK**

Neuro-scientists have told us that the brain is made up of architectures of networks of neurons. At the most basic level, neurons can be seen as functions which, when given some input, will either fire or not fire, depending on the nature of the input. Artificial Neural Networks (ANNs) are designed to mimic the behavior of the brain. Some ANNs are built into hardware, but the vast majority are simulated in software, and we concentrate on these.

Neuron in ANNs tends to have fewer connections than biological neurons. Each neuron in ANN receives a number of inputs.

**WHY USE NEURAL NETWORK ?**

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. Other advantages include:

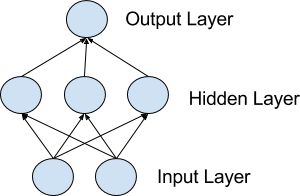
1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.
5. System has got property of continuous learning.

he weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and strength of the output signal. Historically simple step activation functions were used where if the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0.

Artificial Neural Networks consist of a number of **units** which are mini calculation devices. They take in **real-valued** input from multiple other nodes and they produce a single real valued output.

## **NETWORKS OF NEURONS**

Neurons are arranged into networks of neurons. A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology. The architecture of ANNs is as follows:



1. A set of **input units** which take in information about the example to be **propagated** through the network. The set of input units forms what is known as the **input layer**.
2. A set of **hidden units** which take input from the input layer. The hidden units collectively form the **hidden layer**. Many ANNs have multiple hidden layers, with the output from one hidden layer forming the input to another hidden layer. Also, ANNs with no hidden layer - where the input units are connected directly to the output units - are possible. These tend to be too simple to use for real world learning problems, but they are useful to study for illustrative purposes.
3. A set of **output units** which, in learning tasks, dictate the category assigned to an example propagated through the network. The output units form the **output layer**. A weighted sum of the output from the hidden units forms the input to every output unit.
4. The connections between one unit and another are represented by a number called a **weight**, which can be either positive (if one unit excites another) or negative (if one unit suppresses or inhibits another). The higher the weight, the more influence one unit has on another.

Hence ANNs look like this in the general case:

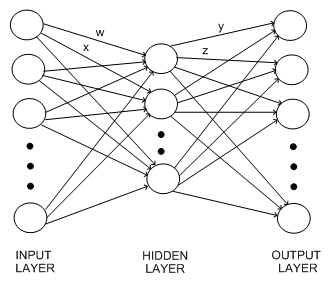


Fig: Basic structure of ANN

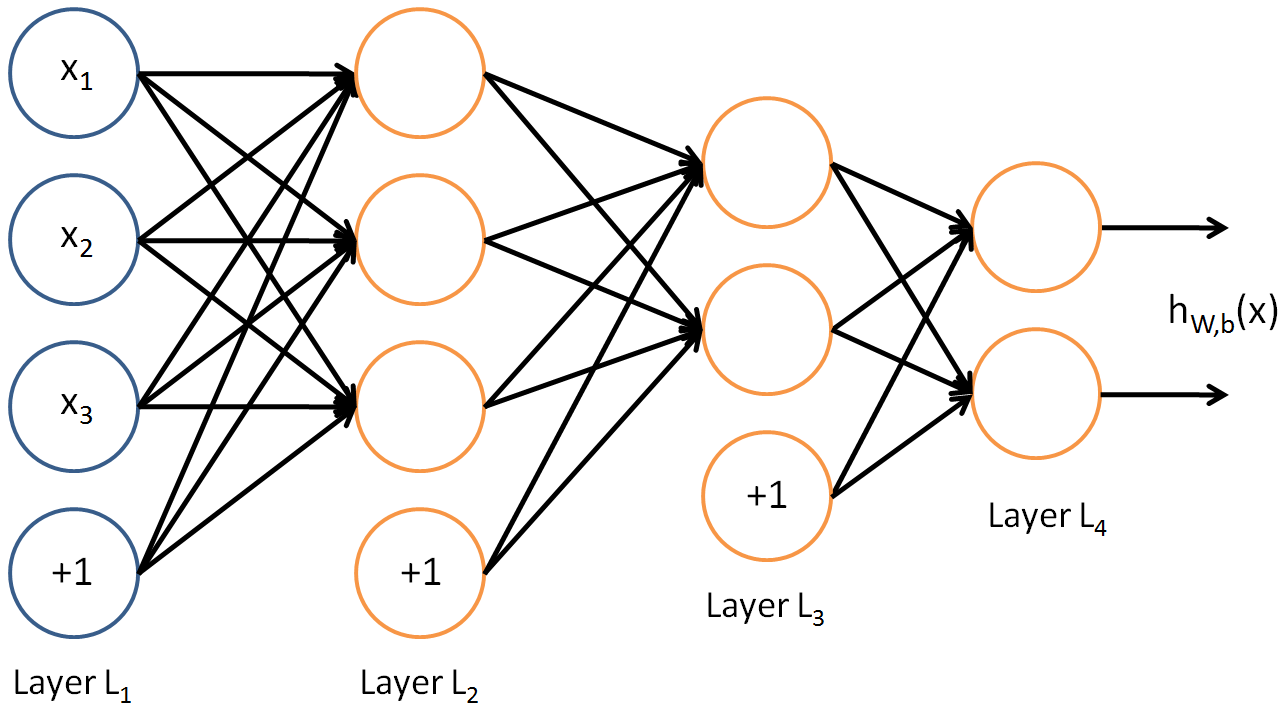
**HOW DOES IT WORK IN PRACTICE?**

Once the network has been trained with enough learning examples, it reaches a point where you can present it with an entirely new set of inputs it's never seen before and see how it responds. For example, suppose you've been teaching a network by showing it lots of pictures of chairs and tables, represented in some appropriate way it can understand, and telling it whether each one is a chair or a table. After showing it, let's say, 25 different chairs and 25 different tables, you feed it a picture of some new design it's not encountered before. Depending on how you've trained it, it'll attempt to categorize the new example as either a chair or a table, generalizing on the basis of its past experience—just like a human.

**MULTI-LAYER ARTIFICIAL NEURAL NETWORKS**

We can now look at more sophisticated ANNs, which are known as multi-layer artificial neural networks because they have several hidden layers.

Multilayer networks solve the classification problem for non linear sets by employing *hidden layers*, whose neurons are not directly connected to the output. The additional hidden layers can be interpreted geometrically as additional hyper-planes, which enhance the separation capacity of the network. To get the final value for the hidden layer, we need to apply the [activation function](https://en.wikipedia.org/wiki/Activation_function).



For one example *x*(*i*):

***z*[1](*i*) = *W*[1]*x*(*i*) + *b*[1](*i*)**

***a*[1](*i*) = tanh(*z*[1](*i*)))**

***z*[2](*i*) =*W*[2]*a*[1](*i*) +*b*[2](*i*)**

***y*̂ (*i*) = *a*[2](*i*) = *σ*(*z*[2](*i*))**

***y*(*i*)*prediction* = { 1 if *a*[2](*i*) >0.5**

**0 otherwise**

Given the predictions on all the examples, you can also compute the cost J as follows:

***J*=−1*m*∑*i*=0*m*(*y*(*i*)log(*a*[2](*i*))+(1−*y*(*i*))log(1−*a*[2](*i*)))**

**ACTIVATION FUNCTION**

The “neuron” is a computational unit that takes as input *x*1,*x*2,*x*3 (and +1 intercept term), and outputs *hW*,*b*(*x*)=*f*(*WTx*)=*f*(∑3*i*=1*Wixi*+*b*), where *f*:R↦R is called the **activation function**.

Their main purpose is to convert a input signal of a node in a ANN to an output signaland to introduce nonlinearity. An advantage of this is that the output is mapped from a range of 0 and 1, making it easier to alter weights in the future.

Common choices for activation function are :

1. **Sigmoid or Logistic** : It is a activation function of form

**f(x)=1/(1+exp(−x))**

Its Range is between 0 and 1. Sigmoids have slow convergence.

1. **Tanh — Hyperbolic tangent :** It’s mathamatical formula is

**f(x) = 1 — exp(-2x) / 1 + exp(-2x).**

Now it’s output is zero centered because its range in between -1 to 1

1. **ReLu -Rectified linear units :**

**R(x) = max(0,x)**

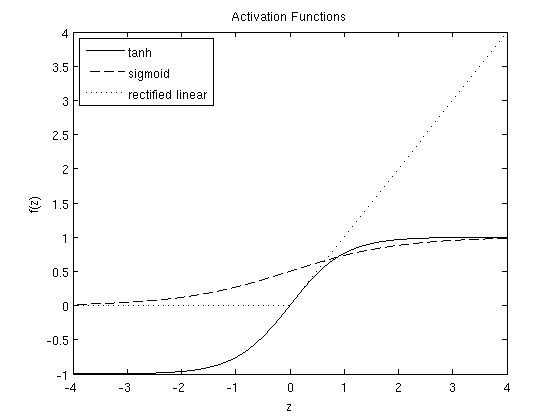


Fig: Plots of the sigmoid ,tanh and rectified linear functions.

In outline, the backpropagation method performs as:

We choose and fix our architecture for the network, which will contain input, hidden and output units.We randomly assign the weights between all the nodes.Each training example is used, one after another, to re-train the weights in the network. After each epoch (run through all the training examples), a termination condition is checked (also detailed below).

**WORKED EXAMPLE**

**DATASET USED :** olivetti faces

[**DATASET ABSTRACT**](https://archive.ics.uci.edu/ml/datasets/seeds)

There are ten different images of each of 40 distinct subjects. For some

subjects, the images were taken at different times, varying the lighting,

facial expressions (open / closed eyes, smiling / not smiling) and facial

details (glasses / no glasses). All the images were taken against a dark

homogeneous background with the subjects in an upright, frontal position (with

tolerance for some side movement).

**NUMBER OF INSTANCES :** 400

**NUMBER OF ATTRIBUTES :** 4096 (as images are 64\*64)

**OUR WORK :**

1. **Logistic Regression :** Data is fit into logistic regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable. This gave us :

**Accuracy = 92.5%**

1. Used our own multilayer model to train.
   1. **Different Activation Functions:** Used tanh and sigmoid activation function and compared the accuracy as follows(h1=10,h2=5):

|  |  |  |  |
| --- | --- | --- | --- |
| **A1** | **A2** | **A3** | **Accuracy** |
| Tanh | Tanh | Sigmoid | 92% |
| Tanh | Sigmoid | Sigmoid | 92% |
| relu | Relu | sigmoid | 95% |
| relu | Tanh | sigmoid | 95% |
| Tanht | Relu | sigmoid | 92% |

Table : Comparisons in various activation functions

* 1. **Different sizes of hidden layers :** Various combinations of sizes for hidden layer 1 and 2 were used and the results are as follows(relu,relu,sigmoid) :

|  |  |  |
| --- | --- | --- |
| **H1** | **H2** | **Accuracy** |
| 20 | 20 | 96% |
| 20 | 5 | 96% |
| 10 | 5 | 95% |
| 30 | 20 | 93% |
| 40 | 20 | 95% |
| 30 | 30 | 95% |
| 10 | 10 | 95% |

Table : Different hidden layer sizes

* 1. **Different values for learning rate(relu,relu,sigmoid, h1=20,h2=20) :** The **learning rate** is a common **parameter** in many of the **learning** algorithms, and affects the speed at which the ANN arrives at the minimum solution. We tried different (small) values for the learning rate.

|  |  |
| --- | --- |
| **No of iterations** | **Accuracy** |
| 1400 | 96% |
| 5000 | 96% |

Table : Impact on changing No of iterations used

**CONCLUSION**

We need to use Cross-validation to test the accuracy on the test set. The optimal number of hidden units could easily be smaller than the number of inputs, there is no rule like multiply the number of inputs with N... If you have a lot of training examples, you can use multiple hidden units, but sometimes just 2 hidden units works best with little data. So we have two hidden layers with sizes 8 and 5 resp . (from table 2).