Artificial Neural Networks

Prof. Dr. A K M Akhtar Hossain
Dept. Of CSE, RU

Artificial Neural Network (ANN)

- **Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.**
- **Neuron in ANNs tend to have fewer connections than biological neurons.**
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).

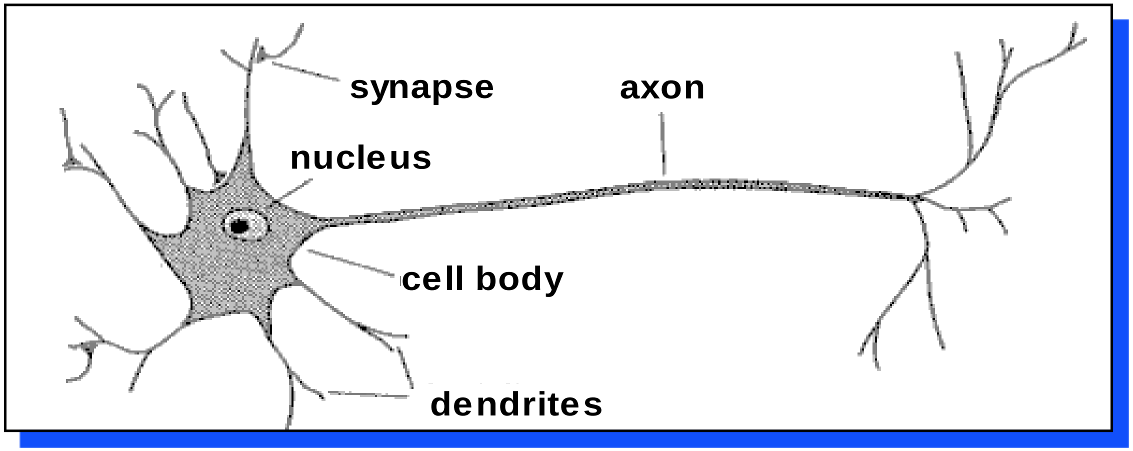
❑ **Before we discuss about ANN, we have to know some basic concept about biological neurons and their functions.**

Biological inspirations

- Some numbers…
  - ❖ The human brain contains about 10^{14} nerve cells (neurons)
  - ❖ Each neuron is connected to the others through 10^4 synapses

- Properties of the brain
  - ➢ It can learn, reorganize itself from experience
  - ➢ It adapts to the environment
  - ➢ It is robust and fault tolerant

Biological Neuron



- A neuron has
  - ❑ A branching input (dendrites which receive input signals.)
  - ❑ A branching output (the axon, which send output signals.)
- The information circulates from the dendrites to the axon via the cell body.
- Dendrites receive input from sensory organs such as the eyes, ears, noses, skin and from axons of other neurons.
- Axons send output to organs such as muscles and to dendrites of other neurons.

The McCulloch-Pitts model of the neuron

- An early attempt to form an abstract mathematical model of a neuron was by McCulloch and Pitts in 1943.
- Their model .....
  - ➢ receives a finite number of inputs x_1 , x_2, ... x_M
- ➢ Computes the weighted sum , $s = \sum_{i=1}^{M} \omega_i x_i$
- ➢ using the weights ω_1 , ω_2 , ... ω_M

The McCulloch-Pitts model of the neuron

- Thresholds ***s*** and outputs 0 or 1 depending on whether the weighted sum is less than or greater than a given threshold value T.
- Node inputs with positive weights are called excitatory.
- Node inputs with negative weights are called inhibitory.

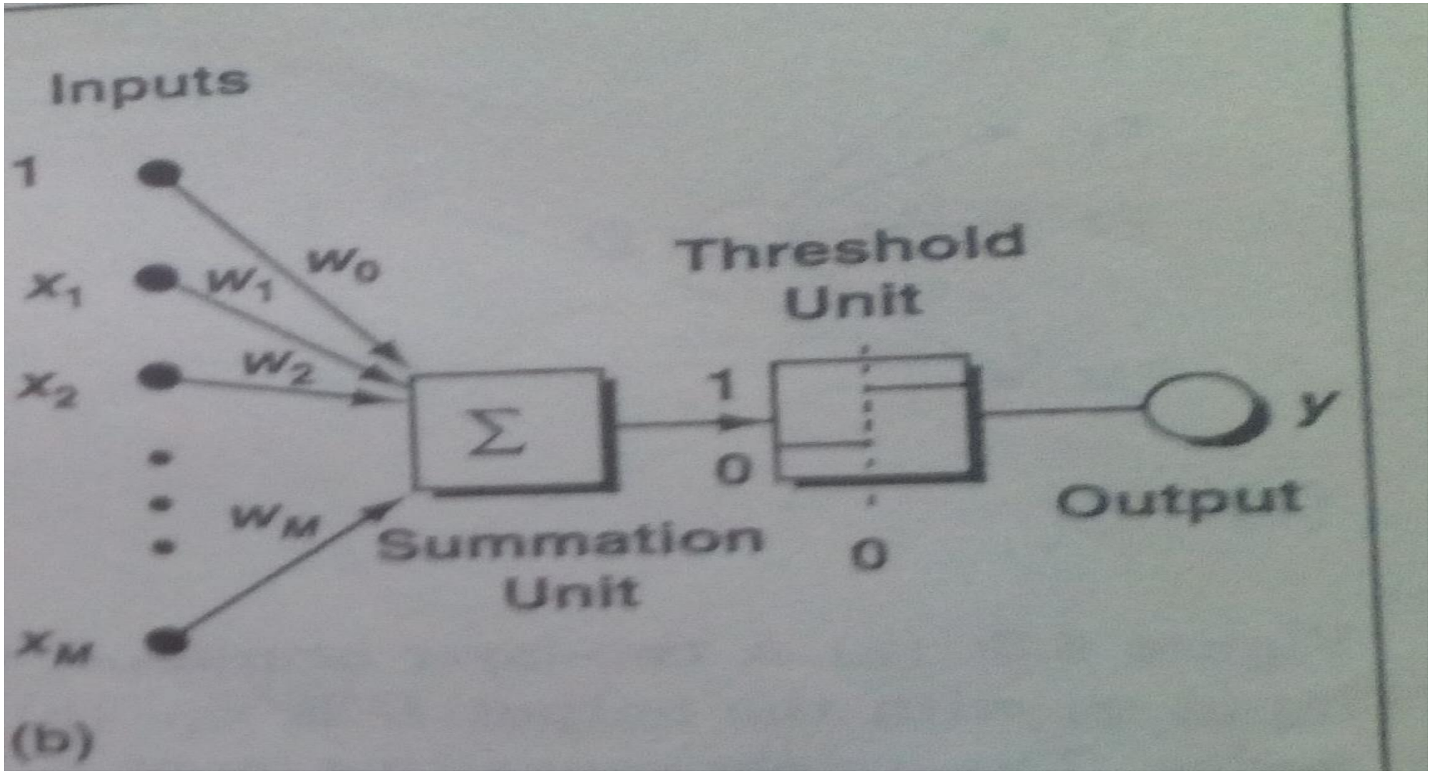
The McCulloch-Pitts model of the neuron

- The action of the model neuron is output a if
- $\omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_M x_M > T$ (1)
- and 0 otherwise.
- We rewrite Eq.(1) as the sum
- $D = \omega_0 x_0 + \omega_1 x_1 + \cdots + \omega_M x_M$ (2)
- Where, ω_0 = -T and x_0 =1.
- Output 1 if D>0 and Output 0 if D≤0.
- D = Decision Parameter.

The McCulloch-Pitts model of the neuron

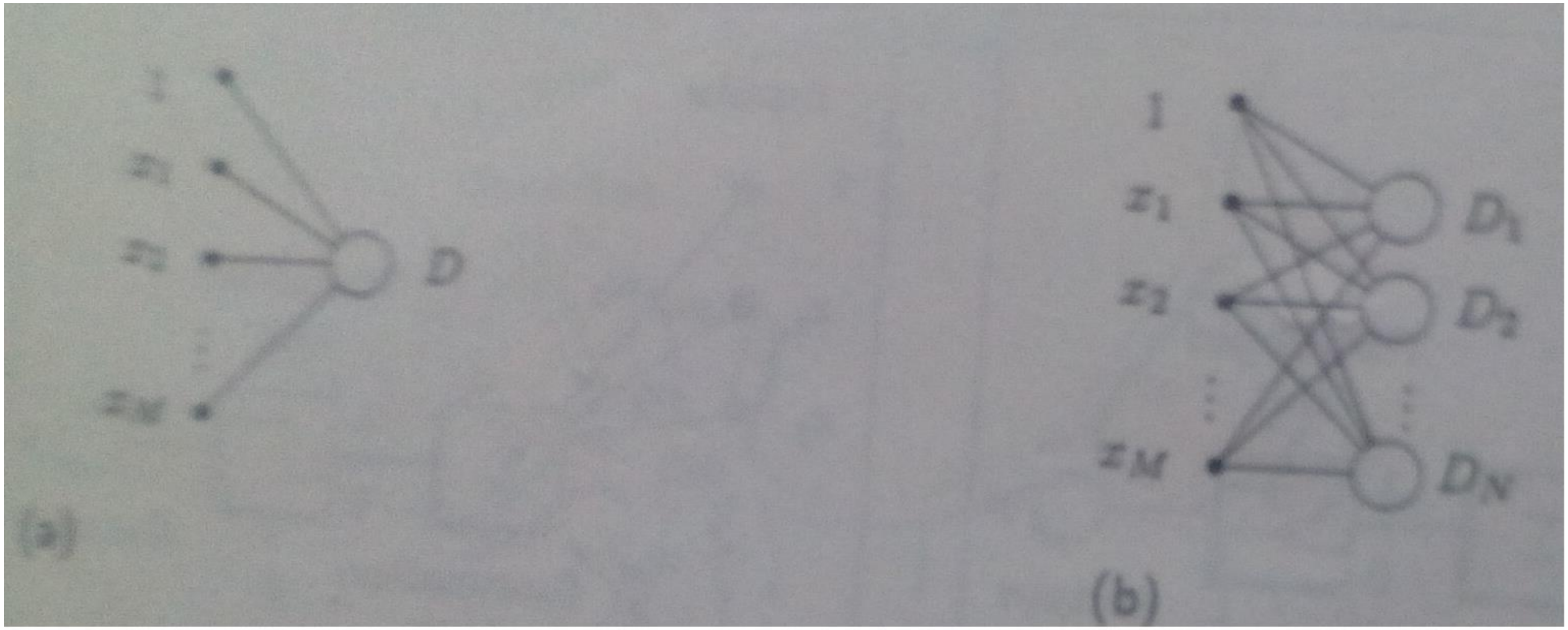
- The weight ω_0 in Eq.(1) is called the bias weight.
- The input x_0 in Eq.(1) is called the bias input.

The McCulloch-Pitts model of the neuron



Model of a neuron with a bias weight.

A two layer neural net



- Figure(a). A two-layer neural net with one output. Fig.(b). A two-layer neural net with multiple output.

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

Contd..

- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. **Each link has a weight**,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

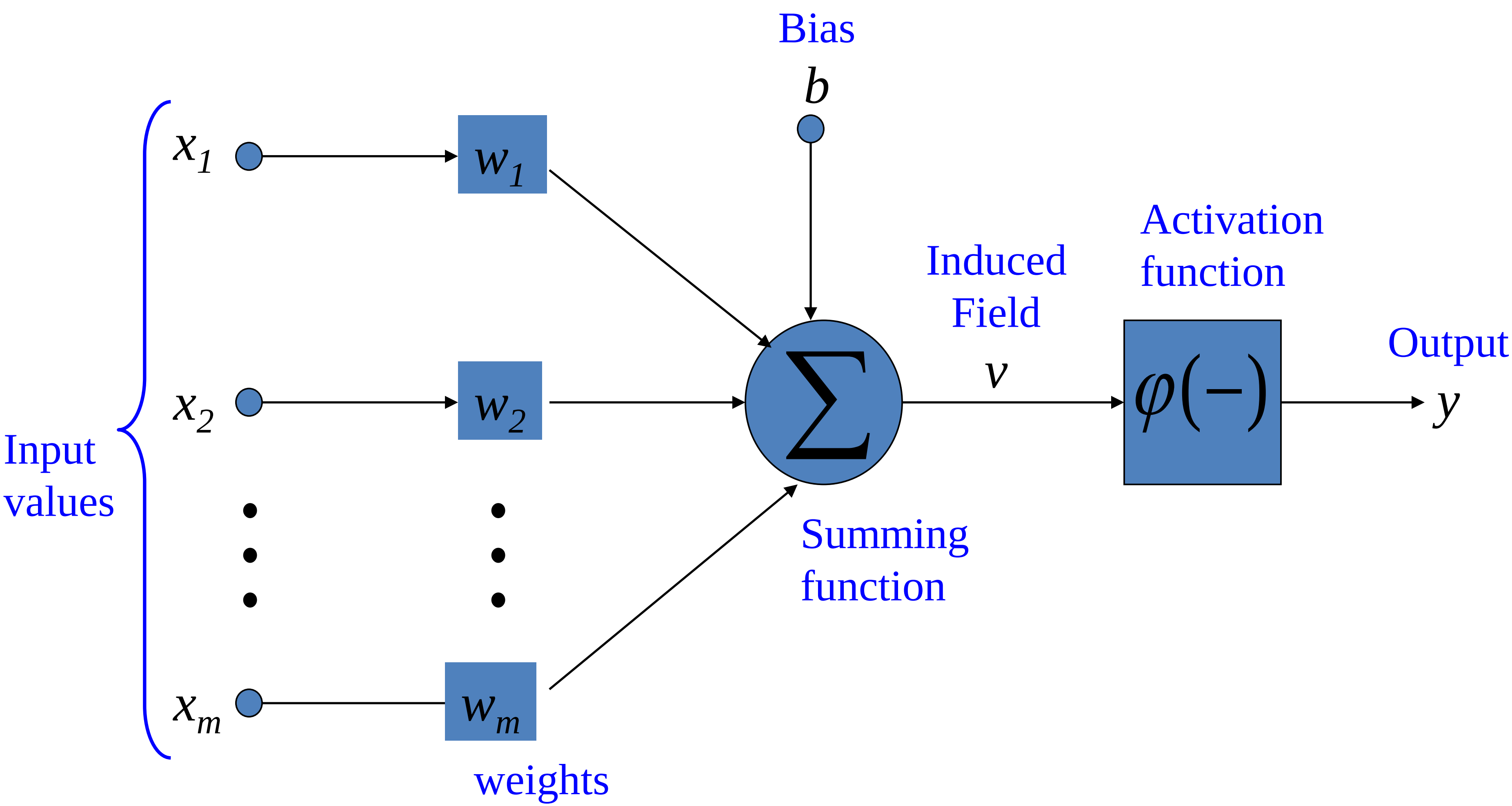
Applications off NNs

- **classification**
  - in marketing: consumer spending pattern classification
  - In defence: **radar and sonar image classification**
  - In agriculture & fishing: fruit and catch grading
  - In medicine: ultrasound and electrocardiogram image classification, EEGs, medical diagnosis
- **recognition and identification**
  - In general computing and telecommunications: speech, vision and handwriting recognition
  - In finance: signature verification and **bank note verification**, voice recognition, Person Identification
- **assessment**
  - In engineering: product inspection monitoring and control
  - In defence: target tracking
  - In security: motion detection, surveillance image analysis and fingerprint matching
- **forecasting and prediction**
  - In finance: foreign exchange rate and stock market forecasting
  - In agriculture: crop yield forecasting , **GIS** (Geographical Information Systems)
  - In marketing: sales forecasting
  - In meteorology: weather prediction

Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of links, describing the neuron inputs, with weights W_1, W_2, …, W_m
  2. An adder function (linear combiner) for computing the weighted sum of the inputs:
     (real numbers)
     $$\mathbf{u} = \sum_{j=1}^{m} \mathbf{w_j x_j}$$
  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias and the 'u' is the weighted sum.
     $$\mathrm{y} = \varphi(\mathrm{u} + b)$$

The Neuron Diagram



Bias of a Neuron

- The bias ***b*** has the effect of applying a transformation to the weighted sum ***u***

  v = u + b

- The **bias is an external parameter** of the neuron. It can be modeled by adding an extra input.
- ***v*** is called **induced field** of the neuron

$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$

Neuron Models

- The choice of activation function φ determines the neuron model.

Examples:

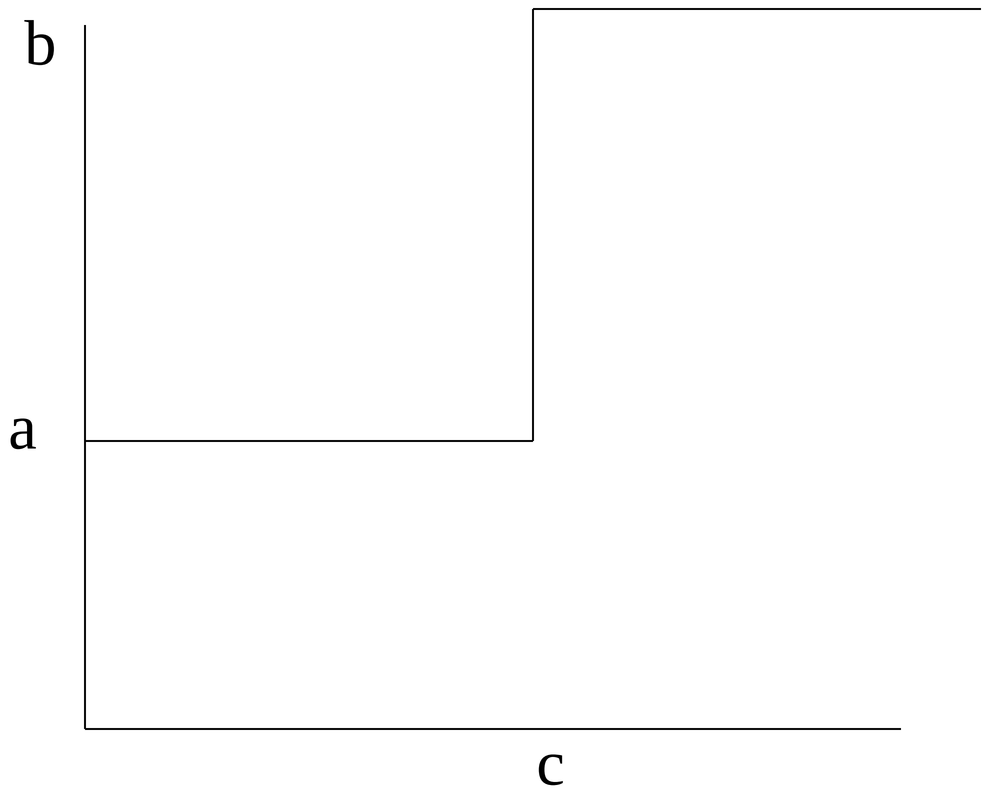
- step function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

- ramp function: $$\mathbf{f(x)} = \begin{cases} \mathbf{1} & x \geq 0 \\ x & 0 \leq x \leq 1 \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

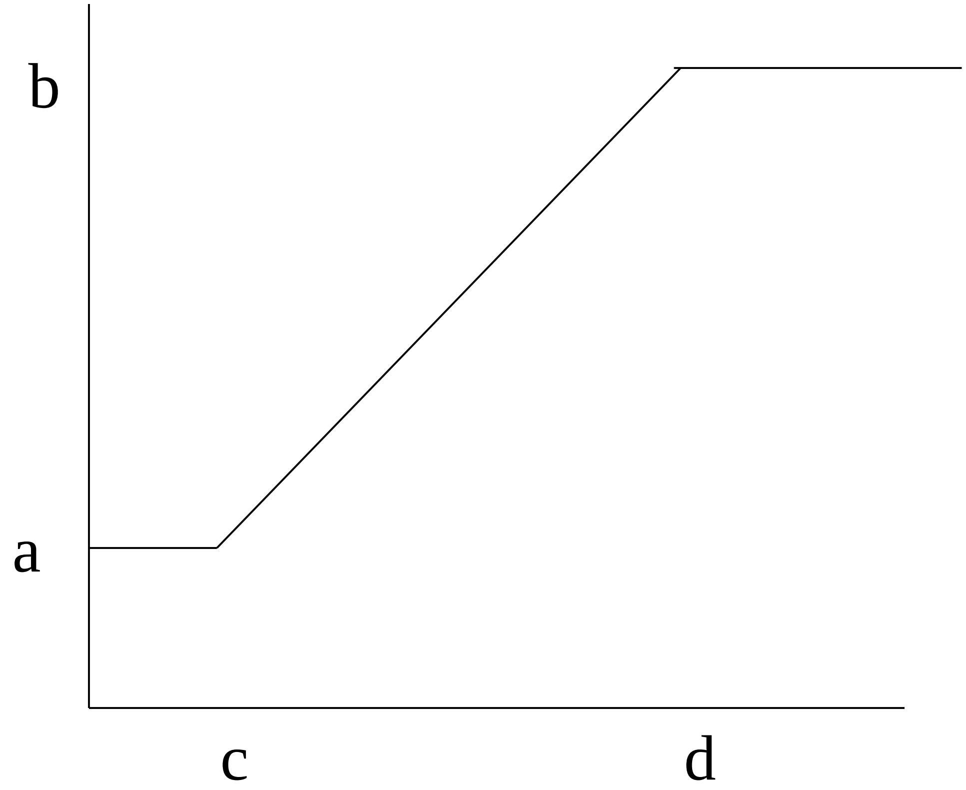
- sigmoid function with z,x,y parameters $$\varphi(v) = z + \frac{1}{1 + \exp(-xv + y)}$$

- Gaussian function: $$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2 \right)$$

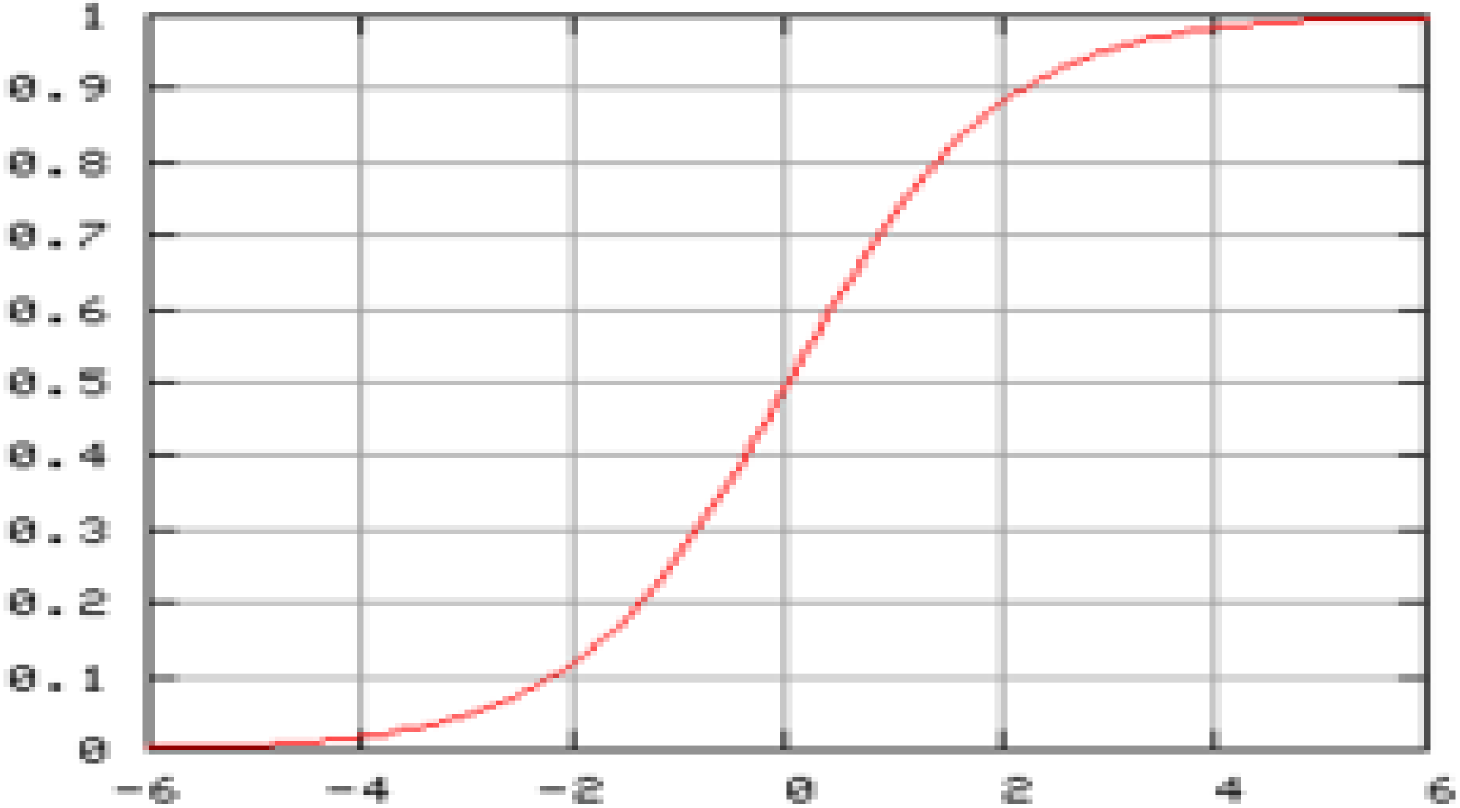
Step Function



Ramp Function

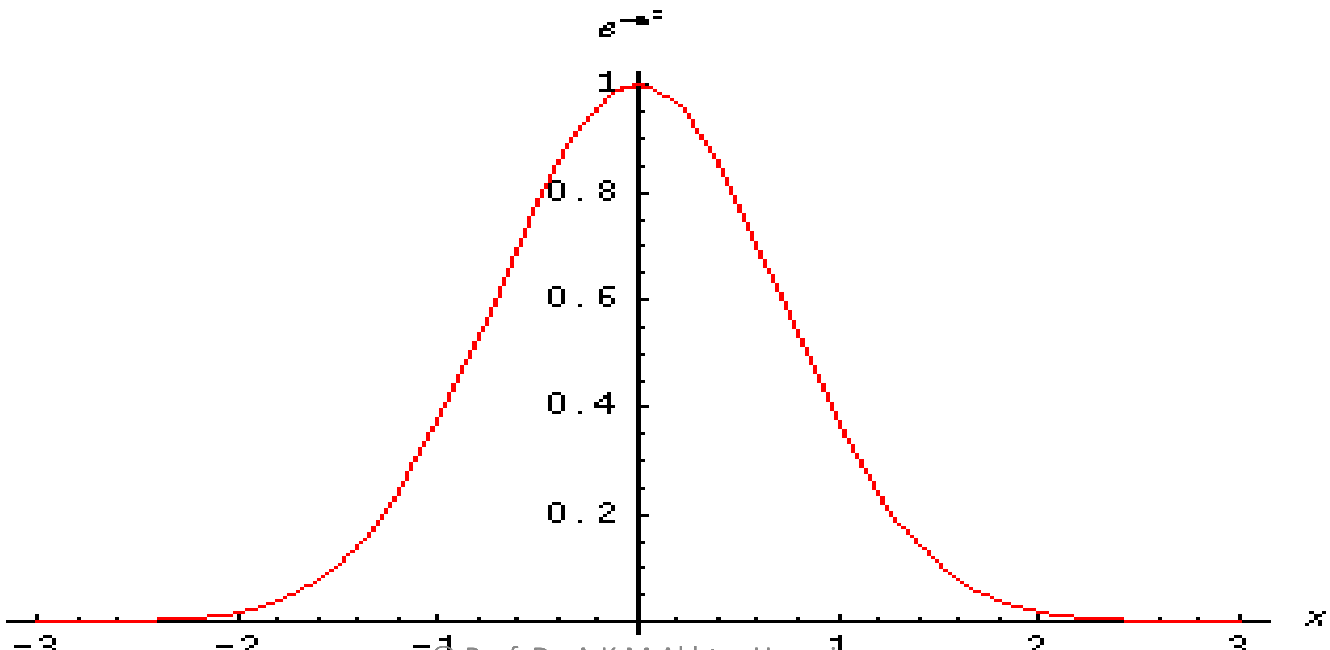


Sigmoid function



• The **Gaussian function** is the probability function of the normal distribution. Sometimes also called the frequency curve.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2},$$



Network Architectures

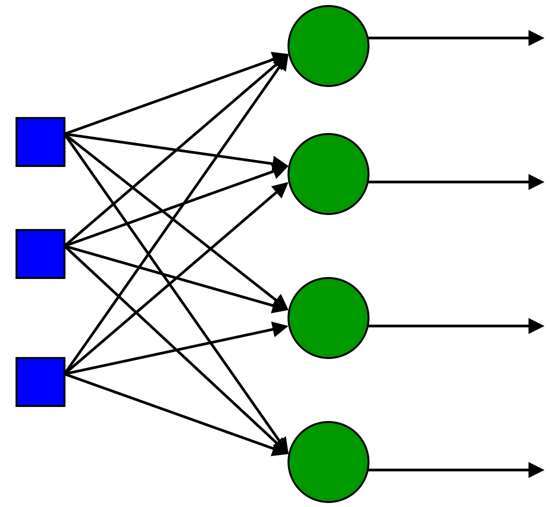
- Three different classes of network architectures

  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent

- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer of source nodes

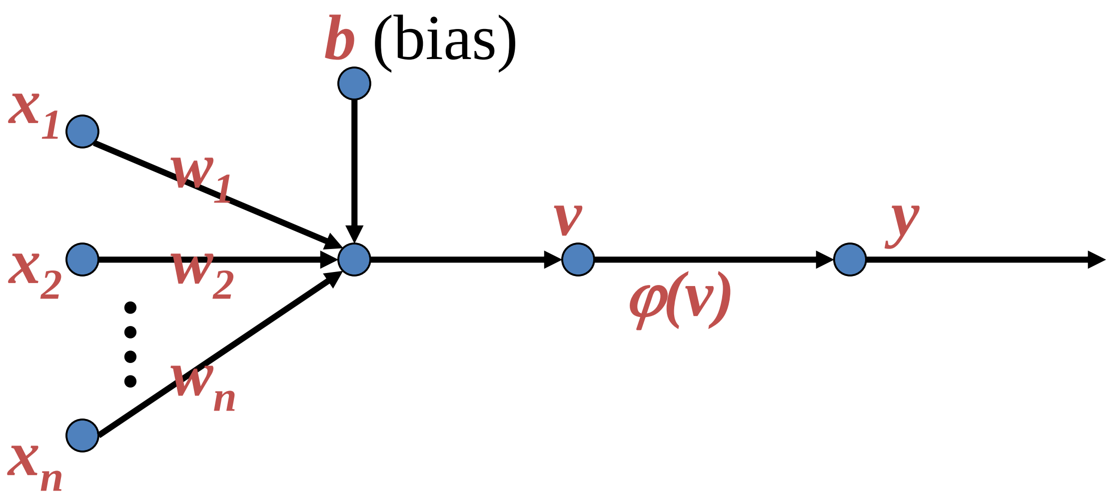


Output layer of neurons

Perceptron: Neuron Model
(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Supervised learning

- The desired response of the neural network in function of particular inputs is well known.
- A “Professor” may provide examples and teach the neural network how to fulfill a certain task.
- In Supervised learning, you train the machine using data which is well **"labeled**." It means some data is already tagged with the correct answer. It can be compared to learning which takes place in the presence of a supervisor or a teacher.

Supervised learning

- 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 propagated back through the system, causing the system to adjust the weights which control the network.
- Example for Supervised Learning Methods:
  - ❑ **Artificial Neural Network**
  - ❑ **Linear regression for regression problems.**
  - ❑ **Random forest for classification and regression problems.**
  - ❑ **Support vector machines for classification problems.**

Unsupervised learning

- Unsupervised learning is a machine learning technique, where you do not need to supervise the model. Instead, you need to allow the model to work on its own to discover information. It mainly deals with the unlabeled data.
- No need of a professor
    - ❑ The network finds itself the correlations between the data.

Unsupervised learning

- 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.

Example for Unsupervised Learning Methods:

- K-means **clustering**.
- KNN (k-nearest neighbors)
- Hierarchal **clustering**.
- Principle Component Analysis.
- Independent Component Analysis.
- Apriori **algorithm**.

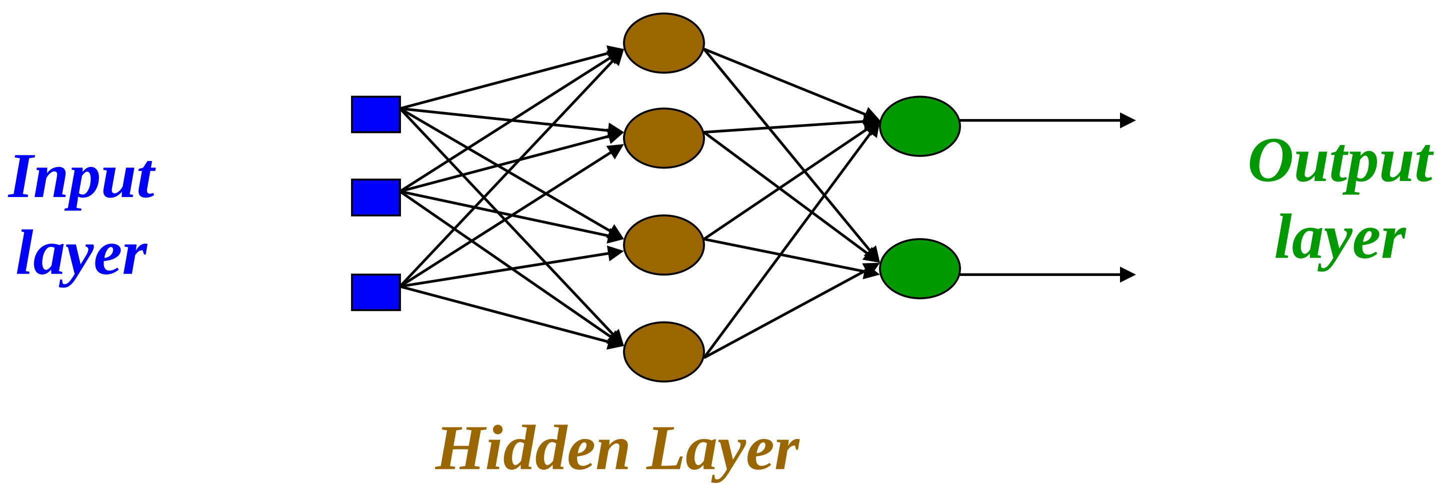
Network Architectures

- Different classes of network architectures:
  - ❖**Multi-Layer Feed-Forward**
  - ❖**Backpropagation Neural Network**

- The architecture of a neural network is linked with the learning algorithm used to train.

Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



3-4-2 Network

Multi layer feed-forward NN (FFNN)

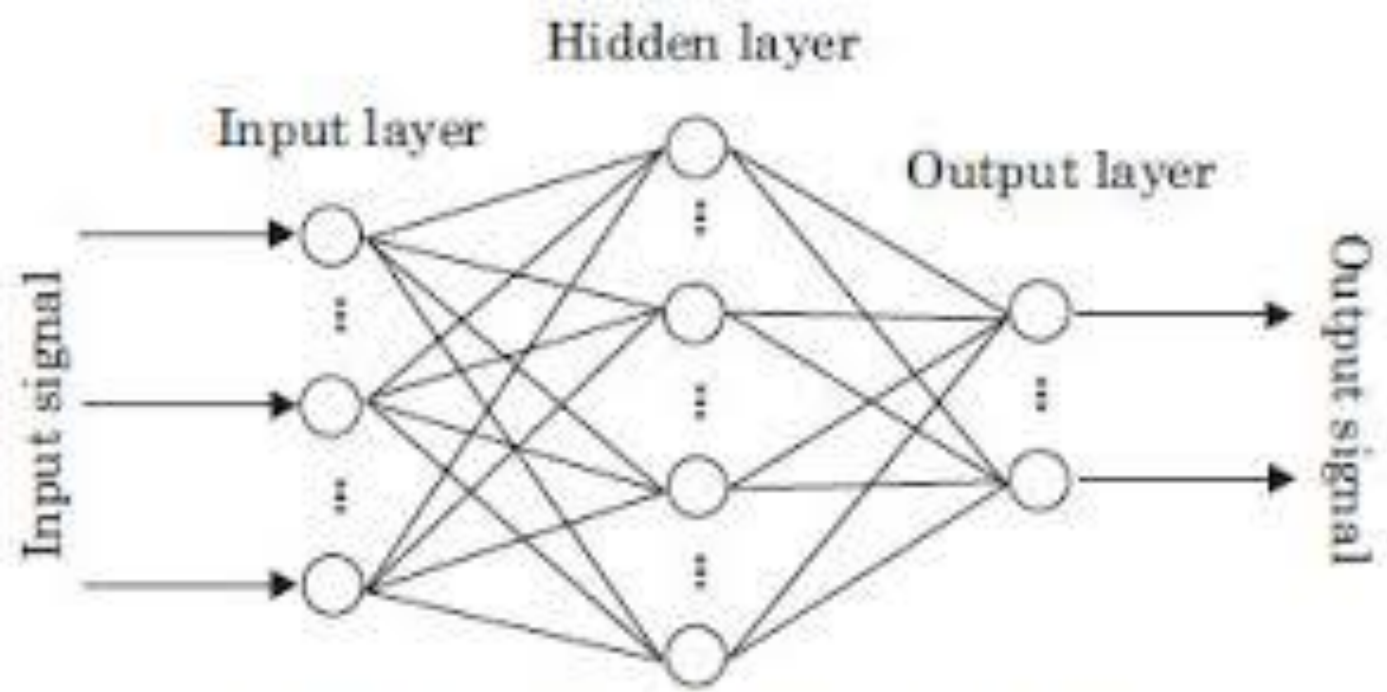


Figure 1. Structured chart of neural network.

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- The activation function for node i may be defined as a simple form of the **sigmoid function** in the following manner:

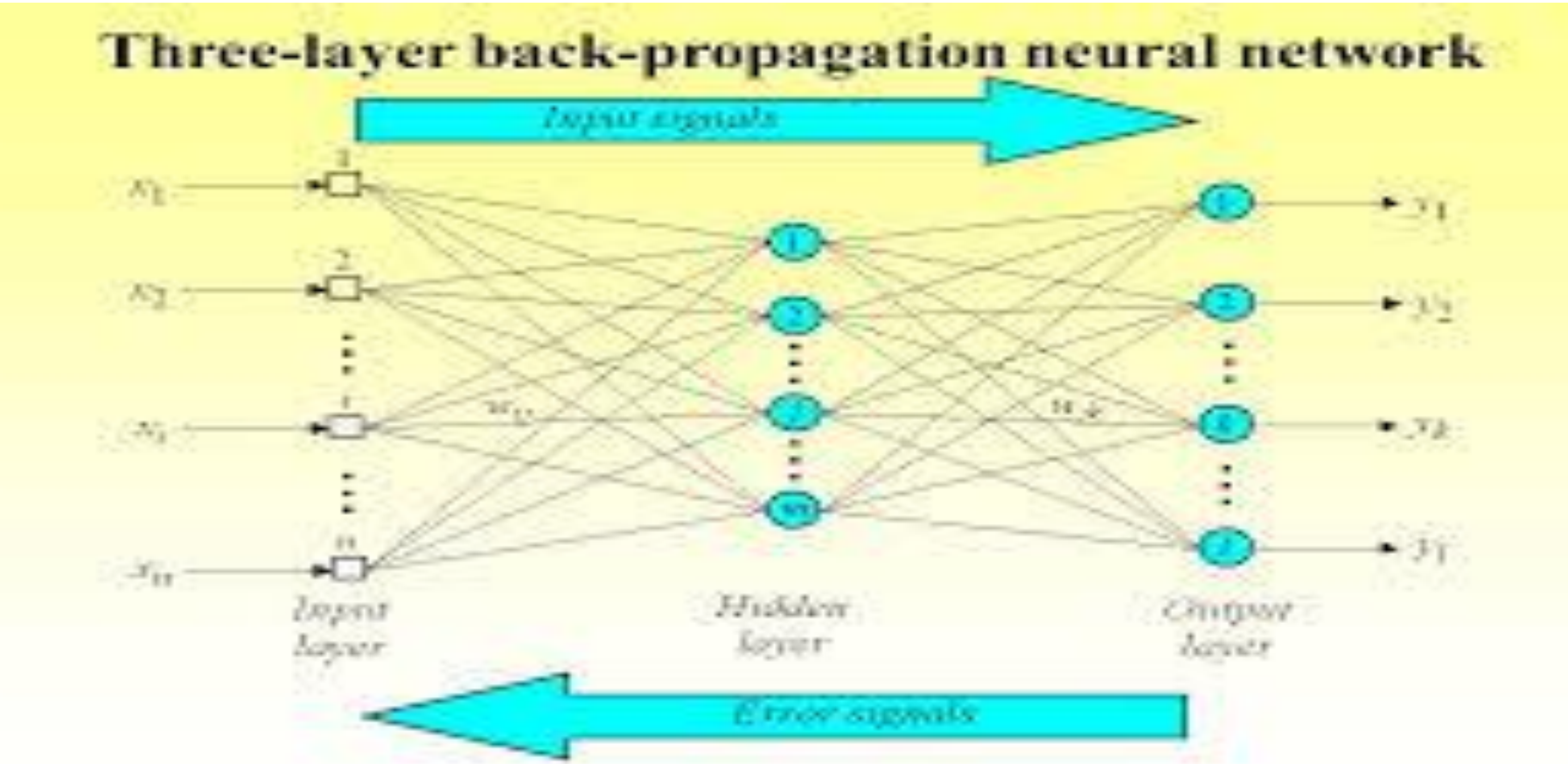
$$\varphi(Vi) = \frac{1}{1+e^{(-A*Vi)}}$$

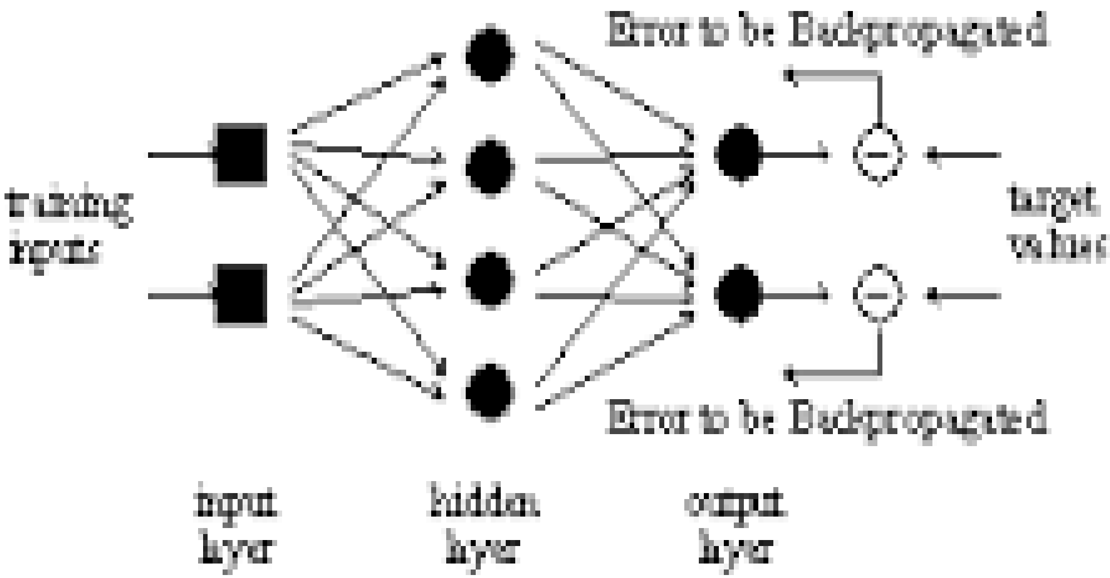
where $A > 0$, $V_i = \sum W_{ij} * Y_j$, such that W_{ij} is a weight of the link from node i to node j and Y_j is the output of node j.

Backpropagation Neural Network : Training Algorithm:

- The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks.
- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

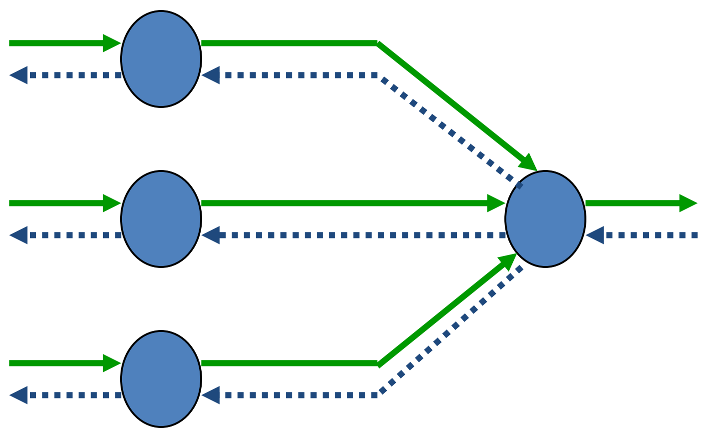
Backpropagation Neural Network :




Error to be Backpropagated
training inputs
target values
Error to be Backpropagated
input layer
hidden layer
output layer

Backpropagation

- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

Contd..

- Consider a network of three layers.
- Let us use i to represent nodes in input layer, j to represent nodes in hidden layer and k represent nodes in output layer.
- w_{ij} refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value Yj of node j

$$\mathbf{Yj} = \frac{1}{1+e^{-X_j}}$$

where, $X_j = \sum x_i \cdot w_{ij} - \theta_j$, $1 \leq i \leq n$; n is the number of inputs to node j, and θ_j is threshold for node j

Total Mean Squared Error

- The error of output neuron *k* after the activation of the network on the *n-th* training example (x(n), d(n)) is:

$$e_k(n) = d_k(n) - y_k(n)$$

- The network error is the sum of the squared errors of the output neurons:

$$\mathbf{E(n)} = \sum \mathbf{e_k^2(n)}$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

Weight Update Rule

- The Back propagation weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Back propagation algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta \mathrm{w}_{\mathrm{ij}} = -\eta \frac{\partial E}{\partial \mathrm{w}_{\mathrm{ij}}}$$

Back propagation learning algorithm (incremental-mode)

n=1;

initialize **weights** randomly;

while (stopping criterion not satisfied or n <max_iterations)

for each example (**x**,*d*)

- run the network with input x and compute the output y

- update the weights in backward order starting from those of the output layer:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

with Δw_{ji} computed using the (generalized) Delta rule

end-for

n = n+1;

end-while;

Stopping criterions

- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not used for updating the weights.

Backpropagation Neural Network :

- This Algorithm is similar to BackPropagation algorithm [60, 62], but the major change is to calculate error rate. Here the *Euclidean distance* is used to calculate the error rate, which is the difference between the present output and the target output. The proposed algorithm is as follows:
- The learning of *MDER-BP* [53] Algorithm has been accomplished by error Back Propagation Neural Network (**Fig.8.7**). The weight vectors and and are the weighted values between layers and *,* and respectively.

Backpropagation ….. Conti…

- In the *hidden layer*, each PE computed the weighted sum according to the equation, which is given by
- $$net_{aj} = \sum W_{ij} O_{ai} \qquad \textbf{(1)}$$
- Where O_{ai} is the input of unit i for pattern number a. The threshold, uh_j of each PE is then added to its weighted sum to obtain the activation $active_j$ of that PE i,e,
- $$active_j = net_{aj} + uh_j \qquad (2)$$
- Where uh_j is the hidden *threshold weight* for *j*th PEs.

Backpropagation ..... Conti…

- This activation determines whether the output of the respective PE is either 1 or 0 (fires or not) by using a sigmoid function,
- $$O_{aj} = \frac{1}{1 + e^{-k_1 * active_j}} \qquad (3)$$
-
- Where k_1 is called the *spread factors*, these O_{aj} are then serve as the input to the output computation. Signal O_{aj} are then fan out to the output layer according to the relation,

Backpropagation ….. Conti…

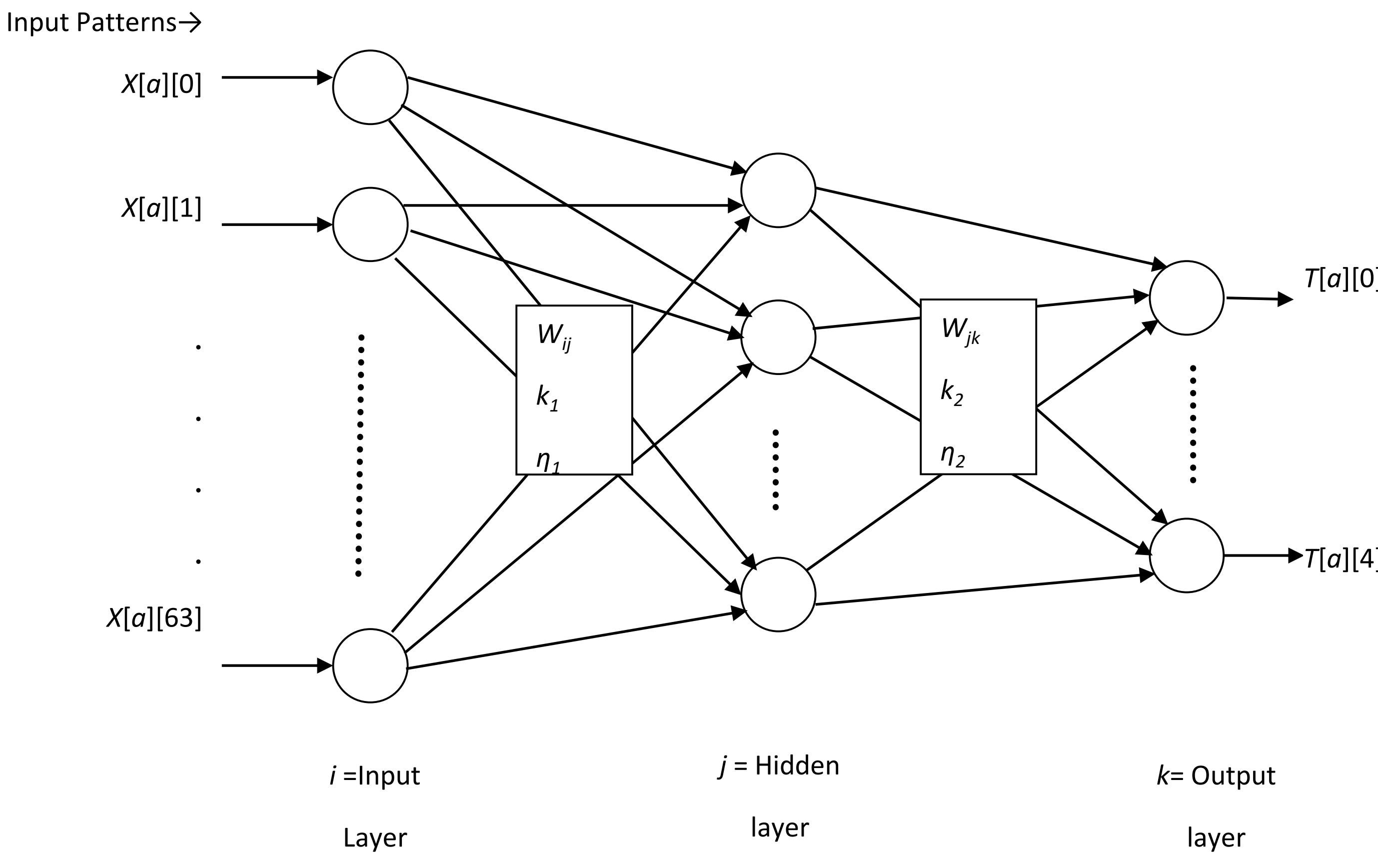


Fig.8.7 MDER- BackPropagation Neural Network

Backpropagation ….. Conti…

- $$net_{ak} = \sum W_{jk} O_{aj} \qquad (4)$$
- and the output threshold weight uo_k for k-th output PEs is added to it to find out the activation $active_{(k)}$
- $$active_k = net_{ak} + uo_k \qquad (5)$$
- The actual output O_{ak} is computed using the same sigmoid function,

Backpropagation ….. Conti…

- $$O_{ak} = \frac{1}{1 + e^{-k_2 * active_k}} \qquad (6)$$
- Here another *spread factor* k_2 has been employed for the output units.
- In the *second stage*, after completing the *feed-forward propagation,* an error is computed by comparing the output O_{ak} with the respective target t_{ak} ,i.e
- $$\delta_{ak} = \sqrt{\sum_{k=0}^{k-1} (t_{ak} - O_{ak})^2} \qquad (7)$$

Backpropagation ….. Conti…

- This error is then used to adjust the weight vector using the equation,
- $$\Delta W_{jk} = \eta_2 k_2 \delta_{ak} O_{aj} O_{ak} (1 - O_{ak}) \qquad (8)$$
- Where,

$$\int^{/} (activeo_k) = k_2 O_{ak} (1 - O_{ak})$$

the derivation of sigmoid function and is the *learning factor* of the network.

Backpropagation ….. Conti…

- The weight vector W_{jk} is then adjusted to $W_{jk} + \Delta W_{jk}$. For the threshold weight of the output PE, similar equation is applied,

$$\Delta uo_k = \eta_2 k_2 \delta_{ak} O_{ak} (1 - O_{ak}) \qquad (9)$$

and the *new threshold weight* equaled

$uo_k + \Delta uo_k$

- In the *next step*, this error and the adjusted weight vector W_{jk} are feedback to the hidden layer to adjust the weight vector W_{ij} and threshold weight *uh*$_{j.}$ In this layer change in weight vector W_{ij} is computed by using equation,
- $\Delta W_{ij} = \eta_1 k_1 O_{ai} O_{aj} (1 - O_{ak}) \sum \delta_{ak} W_{jk}$ (10)
- Where, $\int^{/} (activeh_j) = k_1 O_{ai} (1 - O_{aj})$ and η_1 is the *learning factor* of the network.

- The weight vector W_{ij} is then adjusted to

  $W_{ij} + \Delta W_{ij}$.

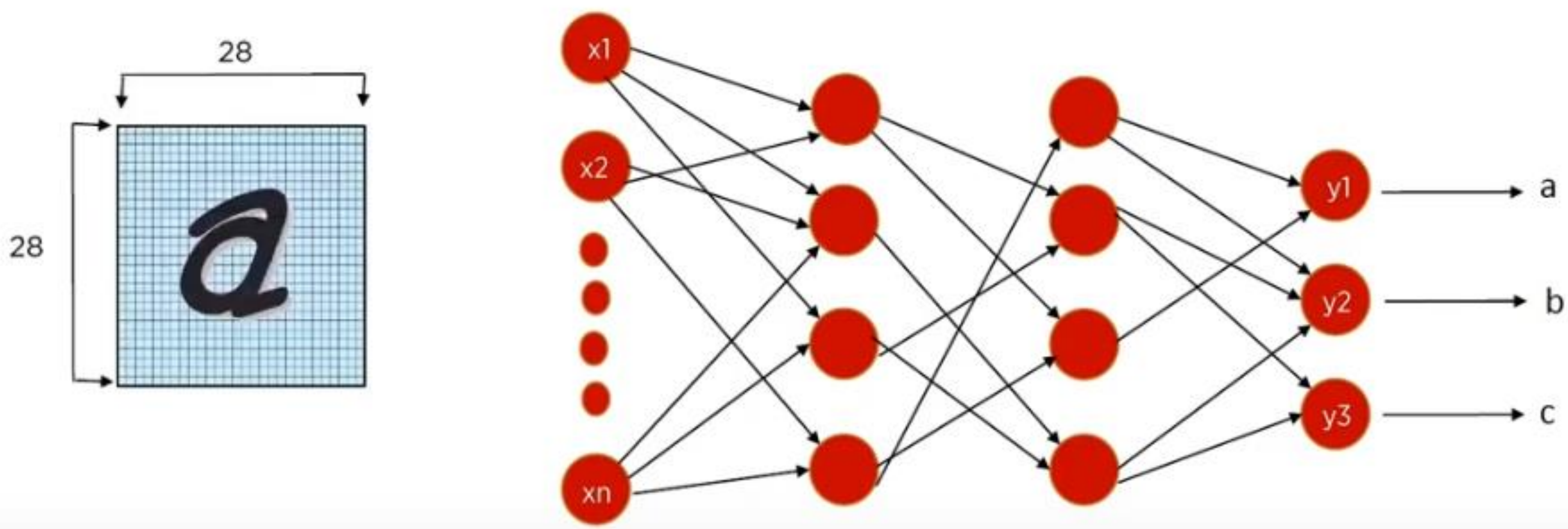
- For the threshold weights of the hidden PEs, similar equation is applied
- $$\Delta uh_j = \eta_1 k_1 (1 - O_{aj}) \sum \delta_{ak} W_{jk} \qquad (11)$$
- and *new threshold weights* are calculated

$$uh_j + \Delta uh_j$$

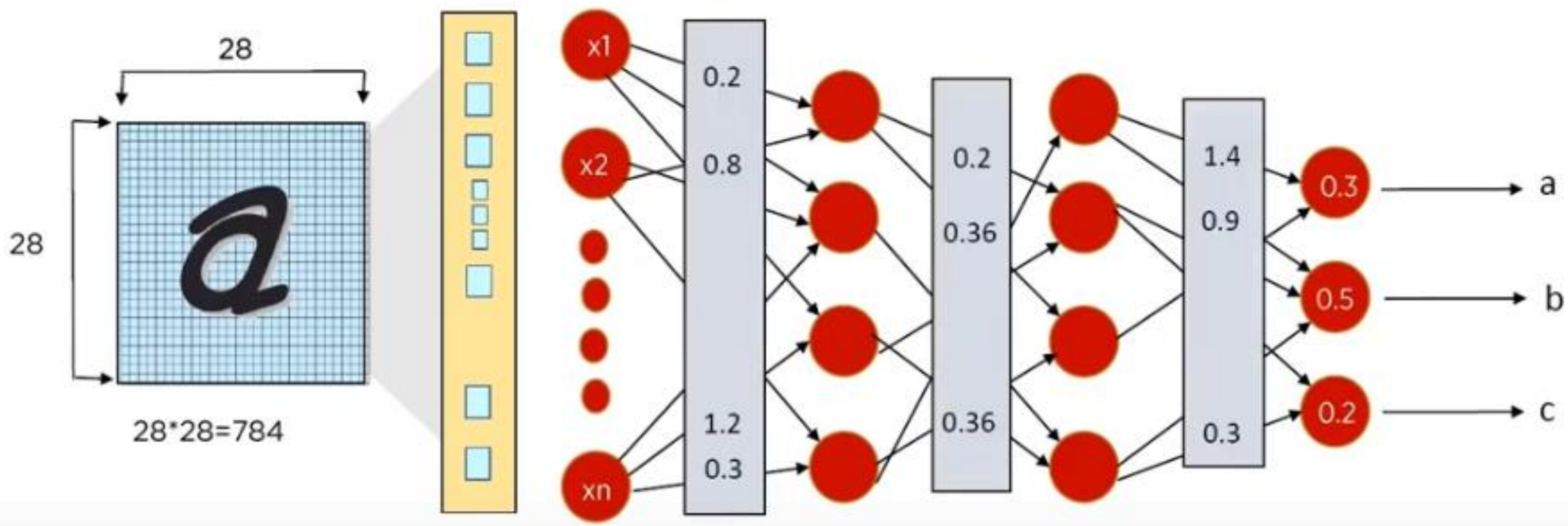
- When the sum-squared error is minimum then we stop the recursion.
- Finally, the weighted and threshold values are stored to test the program.

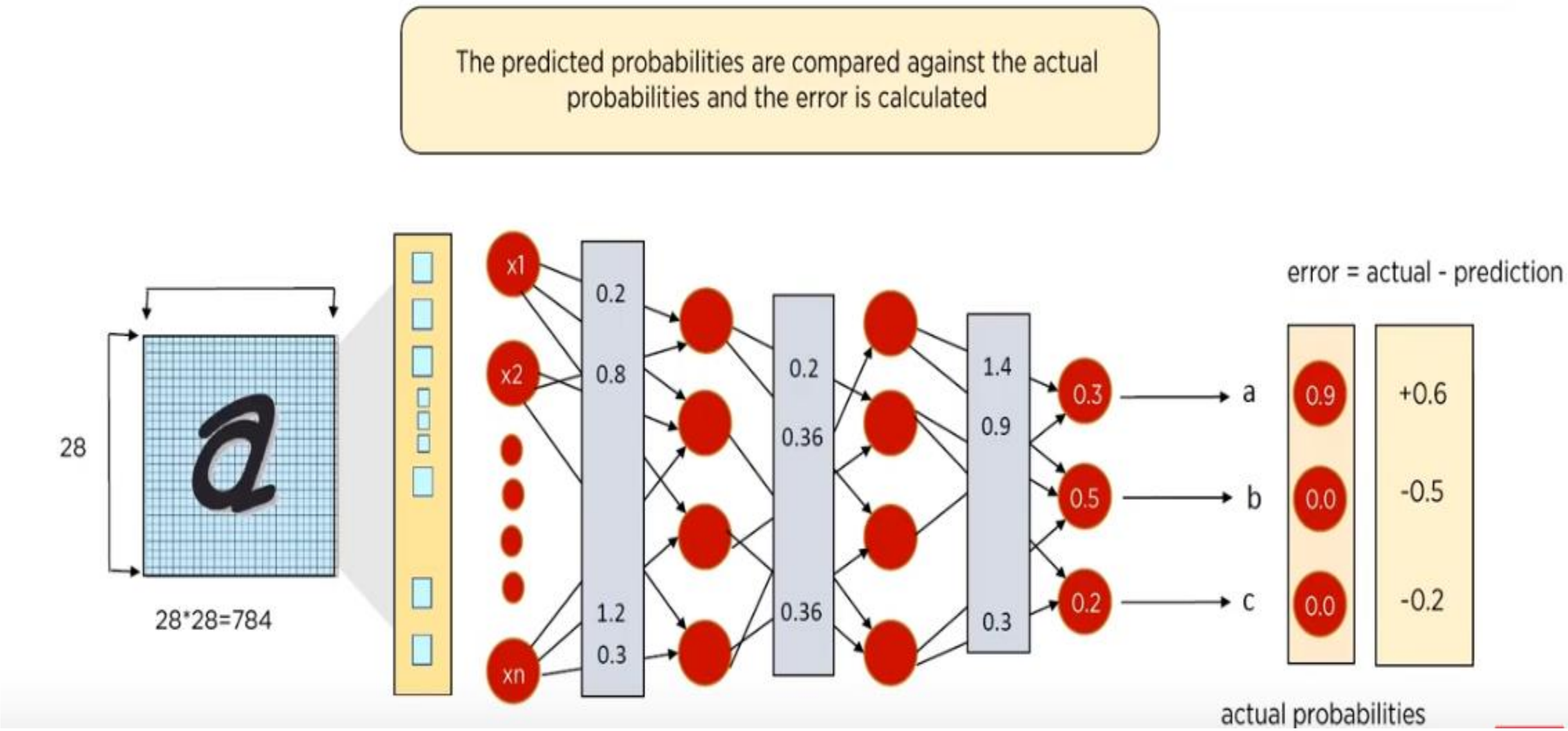
Example for Training

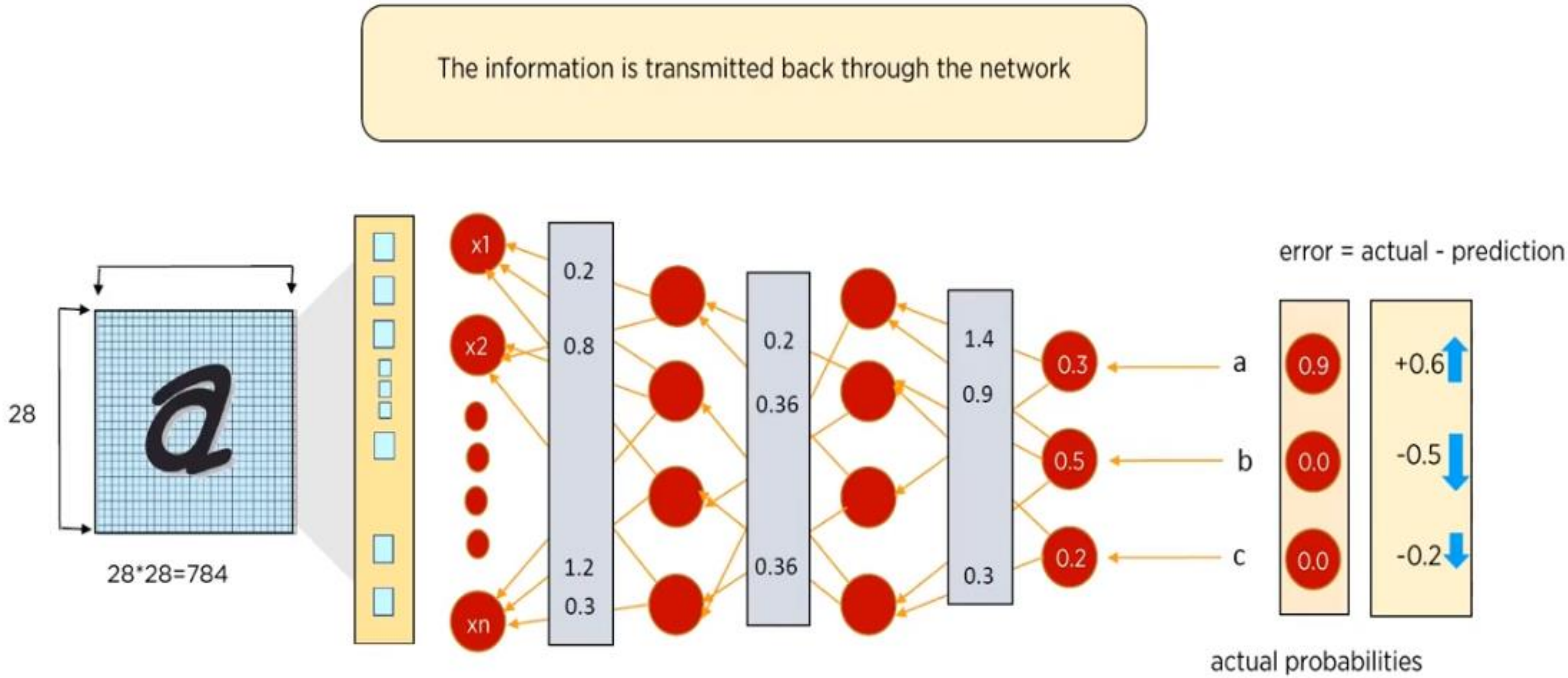
The handwritten alphabets are present as images of 28*28 pixels



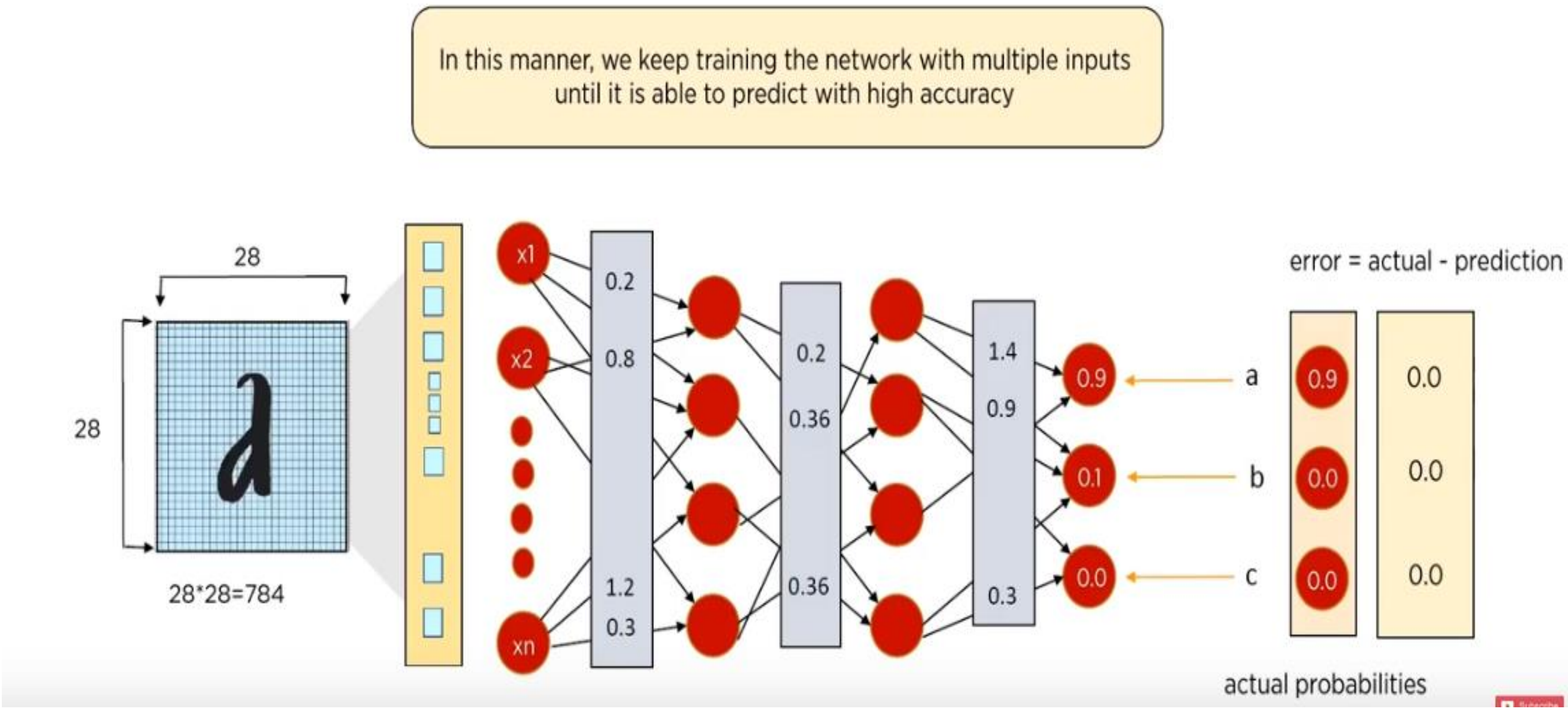
The initial prediction is made using the random weights assigned to each channel



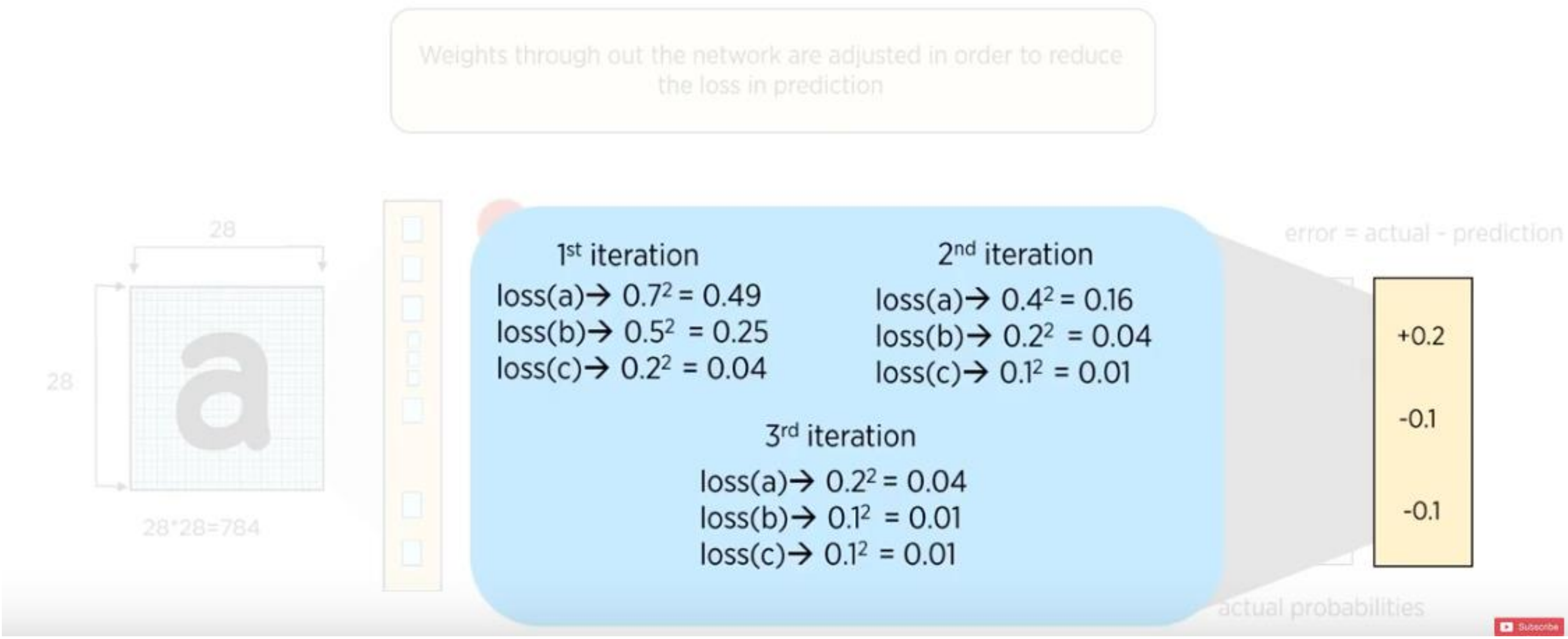

The predicted probabilities are compared against the actual probabilities and the error is calculated
28
28*28=784
x1
x2
xn
0.2
0.8
1.2
0.3
0.2
0.36
0.36
1.4
0.9
0.3
0.3
0.5
0.2
a
b
c
error = actual - prediction
0.9
0.0
0.0
+0.6
-0.5
-0.2
actual probabilities



The information is transmitted back through the network
28
28*28=784
x1
x2
xn
0.2
0.8
1.2
0.3
0.2
0.36
0.36
1.4
0.9
0.3
0.3
0.5
0.2
error = actual - prediction
a
b
c
0.9
0.0
0.0
+0.6
-0.5
-0.2
actual probabilities



In this manner, we keep training the network with multiple inputs until it is able to predict with high accuracy
28
28
28*28=784
x1
x2
xn
0.2
0.8
1.2
0.3
0.2
0.36
0.36
1.4
0.9
0.3
0.9
0.1
0.0
a
b
c
error = actual - prediction
0.9
0.0
0.0
0.0
0.0
0.0
actual probabilities



Weights through out the network are adjusted in order to reduce the loss in prediction
28
28
28*28=784
1st iteration
loss(a)→ $0.7^2 = 0.49$
loss(b)→ $0.5^2 = 0.25$
loss(c)→ $0.2^2 = 0.04$
2nd iteration
loss(a)→ $0.4^2 = 0.16$
loss(b)→ $0.2^2 = 0.04$
loss(c)→ $0.1^2 = 0.01$
3rd iteration
loss(a)→ $0.2^2 = 0.04$
loss(b)→ $0.1^2 = 0.01$
loss(c)→ $0.1^2 = 0.01$
error = actual - prediction
+0.2
-0.1
-0.1
actual probabilities

- **THE END**