ACDS Lecture Series

Lecture - 11

CSIR

Artificial Neural Network

G. N. Sastry and Team

ADVANCED COMPUTATION AND DATA SCIENCES (ACDS) DIVISION

CSIR-North East Institute of Science and Technology, Jorhat, Assam, India

Copyright ACDSD, CSIR-NEIST

11. Artificial Neural Network

11.1 Introduction
- 11.1.1 Motivation
- 11.1.2 BNN vs ANN
- 11.1.3 Definition of ANN
- 11.1.4 Applications

11.2 Overview – Architecture and Learning
- 11.2.1 A Brief History
- 11.2.2 McCulloch Pitts Model
- 11.2.3 Single Layer Perceptron
- 11.2.4 Multi Layer Perceptron
- 11.2.5 Feed Forward Neural Network
- 11.2.6 Feed Backward Neural Network
- 11.2.7 Kohonen Map Network
- 11.2.8 Hopfield Network
- 11.2.9 Radial Basis Function Network
- 11.2.10 Hybrid Neural Network
- 11.2.11 Towards Deep Learning
- 11.2.12 Learning Paradigm

11.3 Feed Forward Neural Network
- 11.3.1 Architecture
- 11.3.2 Components
- 11.3.3 Outcome Behavior
- 11.3.4 Structural Learning
- 11.3.5 Parameter Learning
- 11.3.6 Back Propagation
- 11.3.7 Gradient Descent
- 11.3.8 Examples
- 11.3.9 Generalization
- 11.3.10 Learning vs. Generalization
- 11.3.11 Overfitting
- 11.3.12 Estimation of Generalization Error
- 11.3.13 Regularization Methods
- 11.3.14 Batch Normalization
- 11.3.15 Black Box
- 11.3.16 Summary

Copyright ACDSD, CSIR-NEIST

11.4 Self Organizing Map
- 11.4.1 Introduction
- 11.4.2 Architecture
- 11.4.3 Components
- 11.4.4 Kohonen Map
- 11.4.5 Two Step Approach
- 11.4.6 Summary

11.5 Hopfield Network
- 11.5.1 Introduction
- 11.5.2 Architecture
- 11.5.3 Updating
- 11.5.4 Energy Function
- 11.5.5 Learning

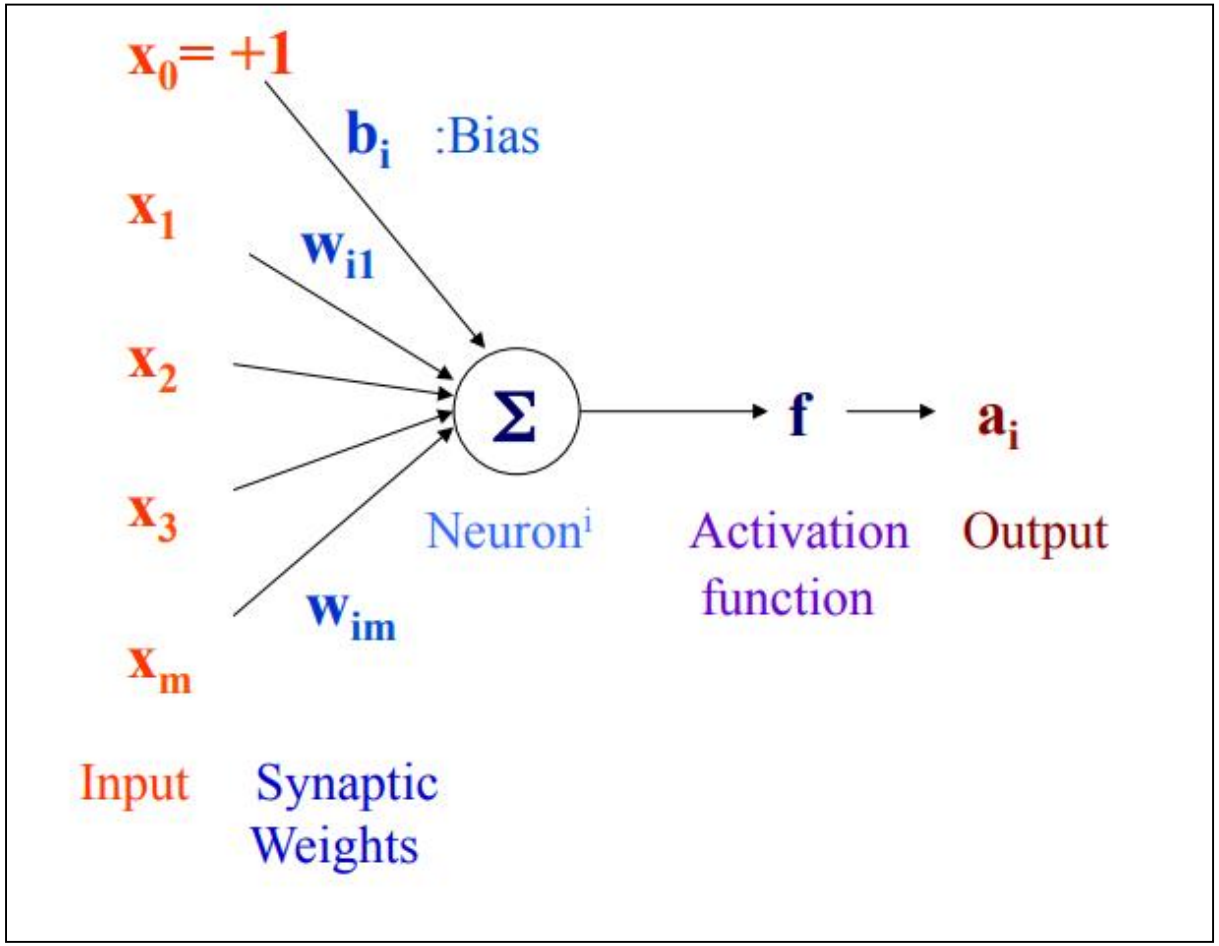
11. 6 Worked Examples

11.7 Exercise

11.6 References
- 11.6.1 Books
- 11.6.2 Video Lectures

Copyright ACDSD, CSIR-NEIST

Artificial Neuron



$$u = \sum_{j=1}^{m} x_j w_{ij}$$

$$a_i = f(u + b)$$

Output = sum (weight × input) + bias

Firing rules:

$$\phi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & 0 \leq v \leq \frac{1}{2} \\ 0 & v \leq 0 \end{cases}$$

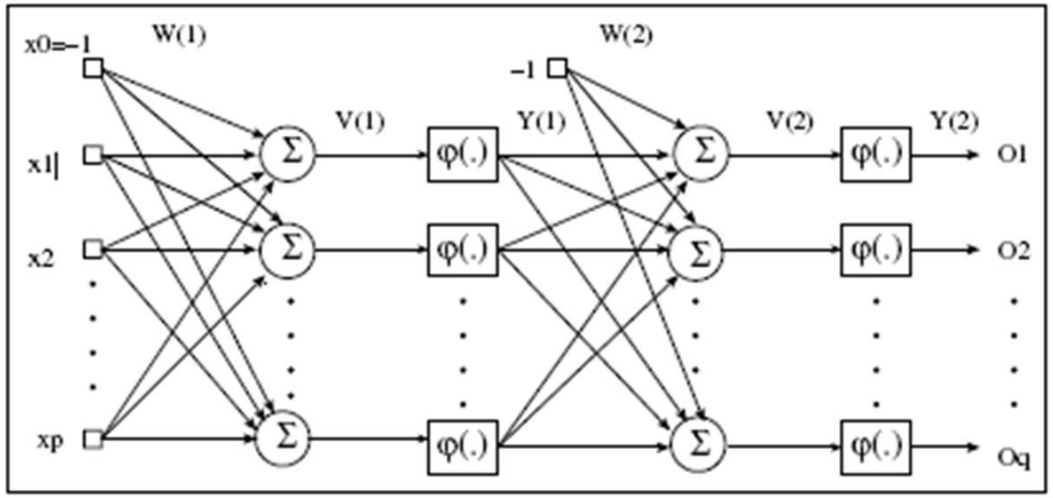
$$\varphi(v) = \tanh(v/2) = \frac{1-\exp(-v)}{1+\exp(-v)}$$

$$\varphi(v) = \frac{1}{1+\exp(-v)}$$

Copyright ACDSD, CSIR-NEIST

The Artificial Neural Network

- Computational models inspired by the human brain:
  - Massively parallel, distributed system, made up of simple processing units (neurons)
  - Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process
- Properties of Neural Network:
- Learning from examples
  - labeled or unlabeled
- Adaptivity
  - changing the connection strengths to learn things
- Non-linearity
  - the non-linear activation functions are essential

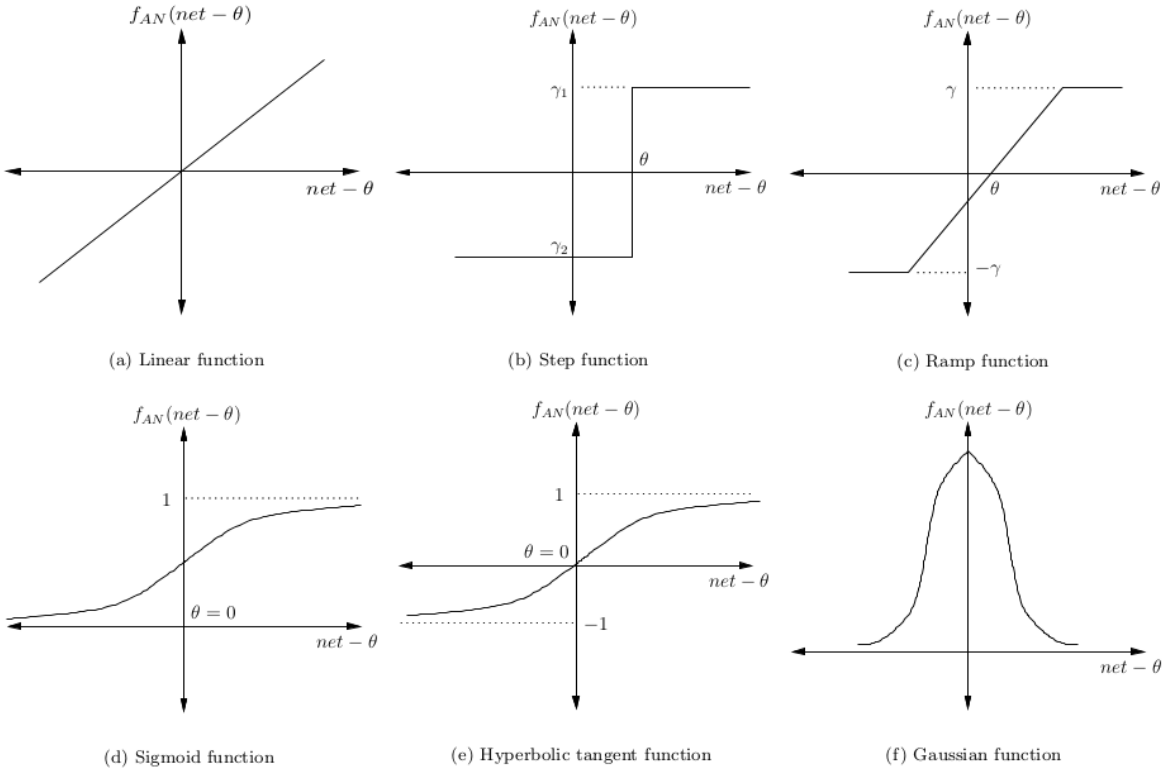


- Fault tolerance
  - if one of the neurons or connections is damaged, the whole network still works quite well
- Thus, they might be better alternatives than classical solutions for problems characterized by:
  - high dimensionality, noisy, imprecise or imperfect data; and
  - a lack of a clearly stated mathematical solution or algorithm

Copyright ACDSD, CSIR-NEIST

Activation Function

- Also known as the squashing function, it limits the amplitude of the output of the neuron
- There are many types of activation function:
  - Linear: $a = f(n) = n$
  - Threshold (hard limiting) : $a = \begin{cases} 1 \; if \; n \geq 0 \\ 0 \; if \; n < 0 \end{cases}$
  - Sigmoid: $a = 1/(1+e^{-n})$
  - Step function: $a = \begin{cases} a \; if \; v < c \\ b \; if \; v > c \end{cases}$
  - Ramp function: $a = \begin{cases} a \; if \; v < c \\ b \; if \; v > d \\ a + \left(\frac{(v-c)(b-a)}{d-c}\right) otherwise \end{cases}$



(a) Linear function (b) Step function (c) Ramp function

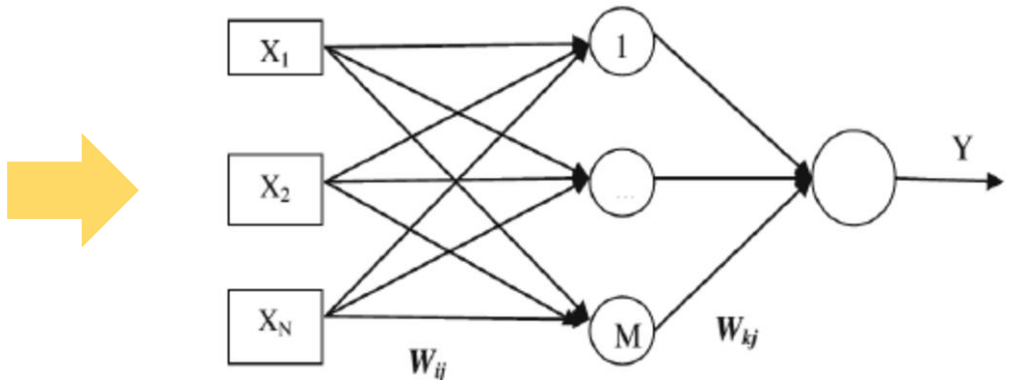
(d) Sigmoid function (e) Hyperbolic tangent function (f) Gaussian function

- The choice of activation function define the neural network

Copyright ACDSD, CSIR-NEIST

Feed Forward Neural Network (FFNN)

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix}$$



$$Y_{nj} = \sum_{i=1}^{n} x_i w_{ij}$$

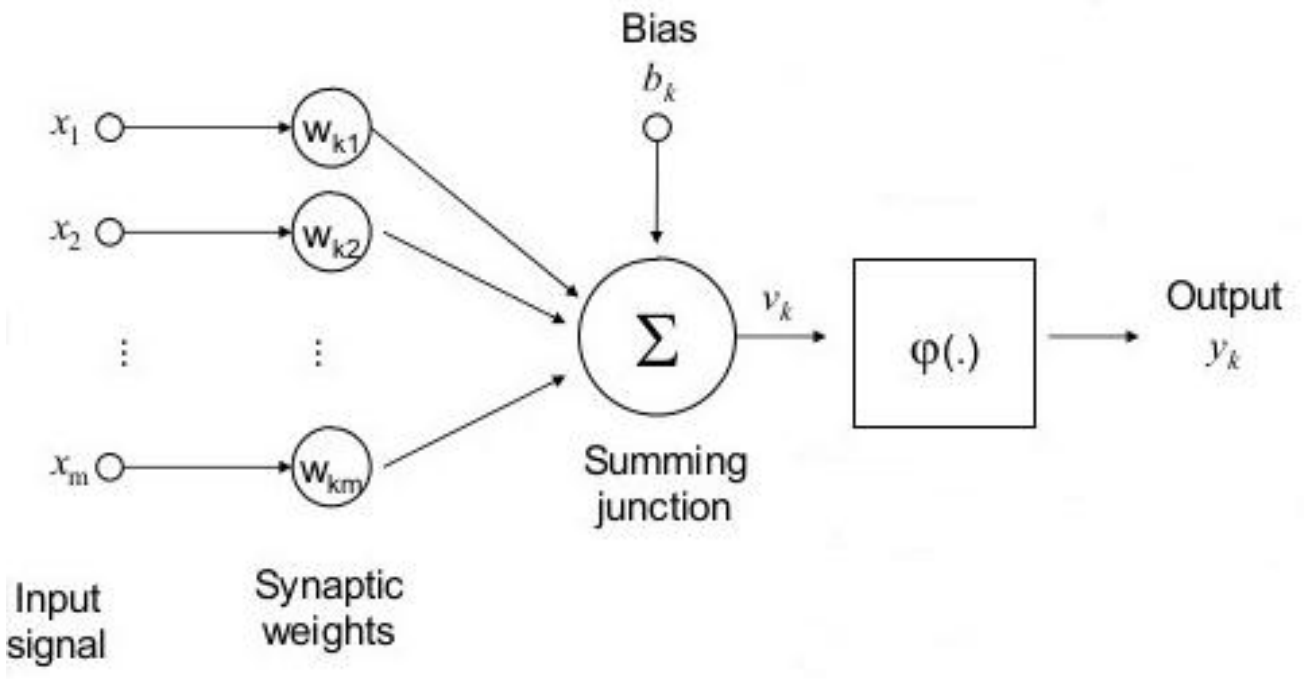
$$= x_0 w_{0j} + x_1 w_{1j} + \ldots + x_n w_{nj}$$

$$= x_0 w_{0j} + \sum_{i=1}^{j} x_i w_{ij}$$

$$Y_{nj} = b + \sum_{i=1}^{j} x_i w_{ij}$$

Copyright ACDSD, CSIR-NEIST

Single Layer FFNN (Perceptron)



$$v_k = \sum_{j=1}^{m} x_j w_{kj} + b_k$$

$$y_k = \varphi(v_k)$$

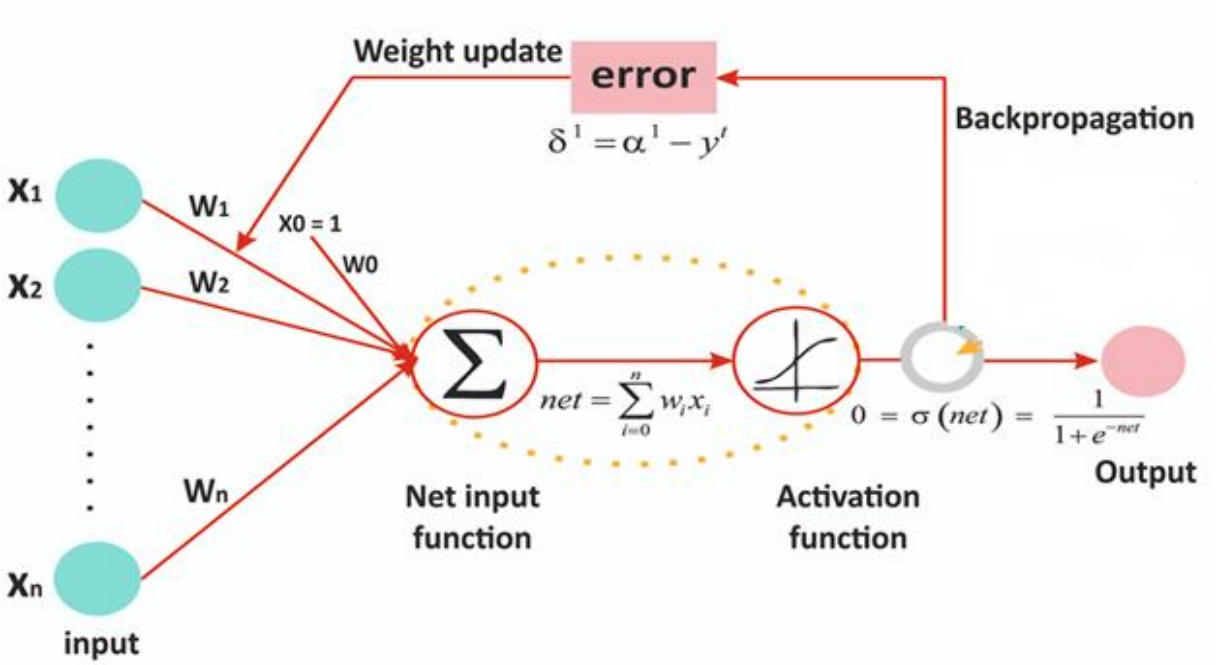
$$\varphi(v_k) = \begin{cases} +1 \; if \; v \geq 0 \\ -1 \; if \; v < 0 \end{cases}$$

$$w_i \leftarrow w_i + (a \times x_i \times e)$$

e = *error*
a = *learning rate*

Copyright ACDSD, CSIR-NEIST

Backpropagation Learning



$$E(\vec{x}) = \frac{1}{2}\sum_{k=1}^{K}(y_k(\vec{x}) - t_k(\vec{x}))^2$$

y – observed output

t – target output

$$w_{j,k} = w_{j,k} + (-\eta)\frac{\partial E(\vec{x})}{\partial w_{jk}}$$

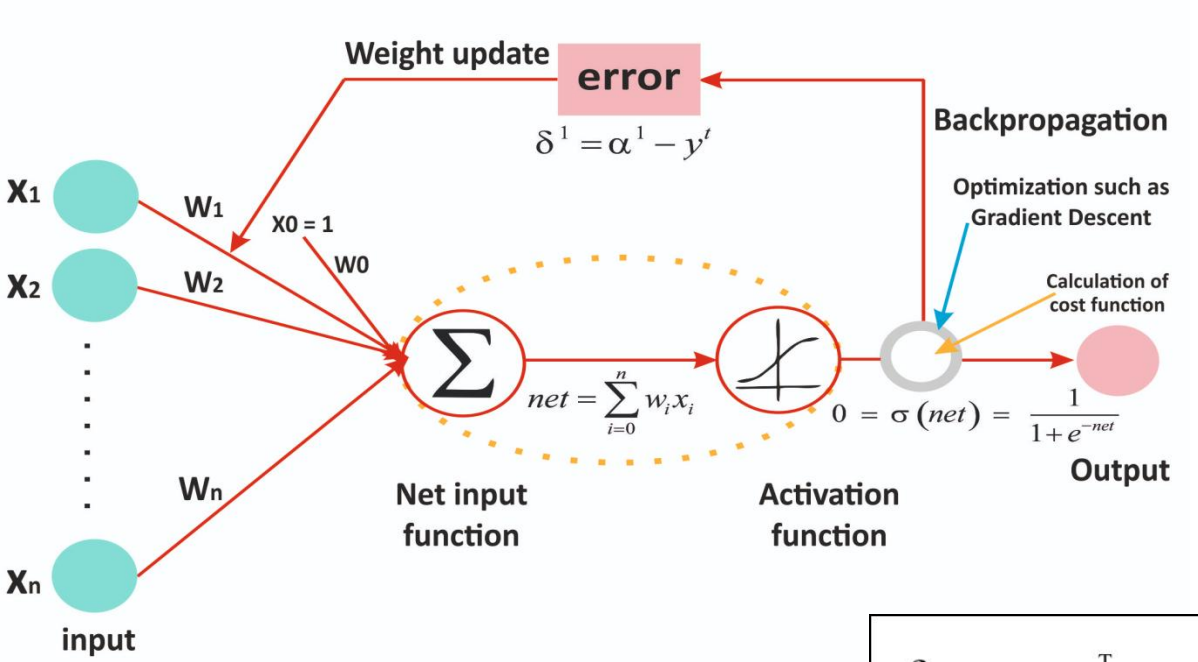
$$\theta_k = \theta_k + (-\eta)\frac{\partial E(\vec{x})}{\partial \theta_k}$$

$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

Copyright ACDSD, CSIR-NEIST

Gradient Descent



$$E = \sum_{k=1}^{R} (y_k - w^T x_K)^2$$

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

$$\frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

$$w_j \leftarrow w_j + 2\eta \sum_{k=1}^{R} \delta_k x_{kj}$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{R} \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$= \sum_{k=1}^{R} 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k$$

...where...

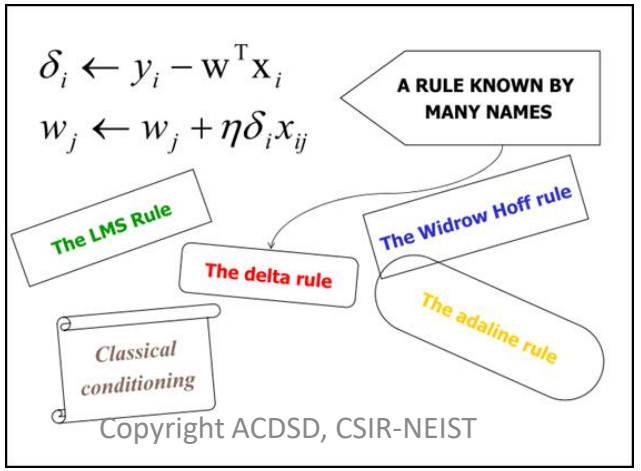
$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

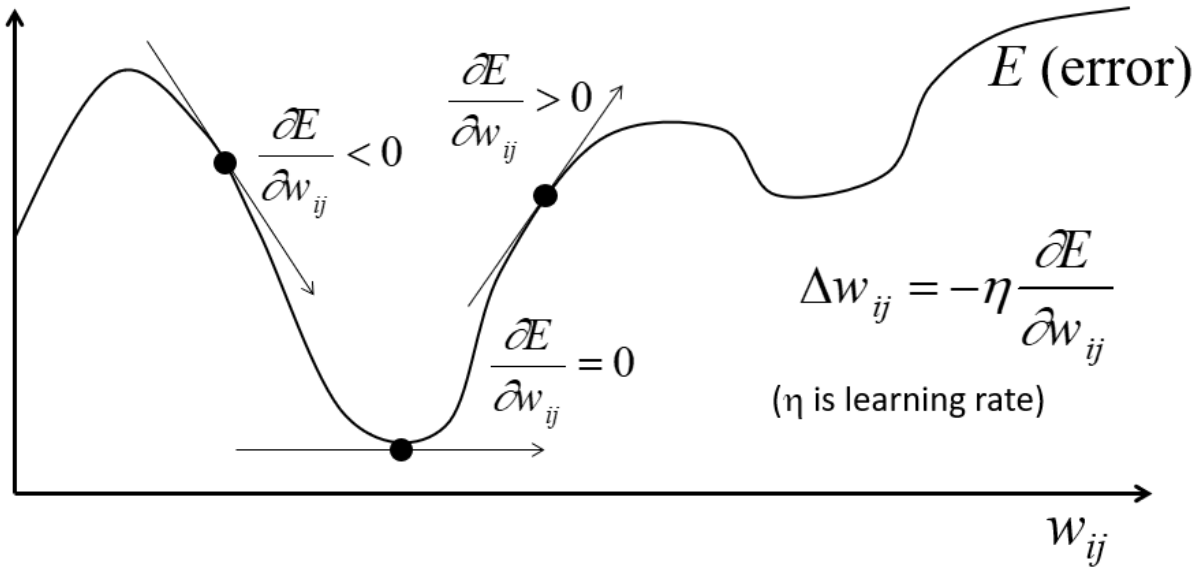
$$\delta_i \leftarrow y_i - \mathrm{w}^{\mathrm{T}} \mathrm{x}_i$$

$$w_j \leftarrow w_j + \eta \delta_i x_{ij}$$



Copyright ACDSD, CSIR-NEIST

Gradient Descent



$$\text{Gradient} \quad \nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

$$\text{Training rule: } \Delta w_i = -\eta \nabla E[\vec{w}] \qquad w_i \leftarrow w_i + \Delta w_i$$

$$\text{i.e.,} \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

Copyright ACDSD, CSIR-NEIST

Gradient Descent for Sigmoidal Unit

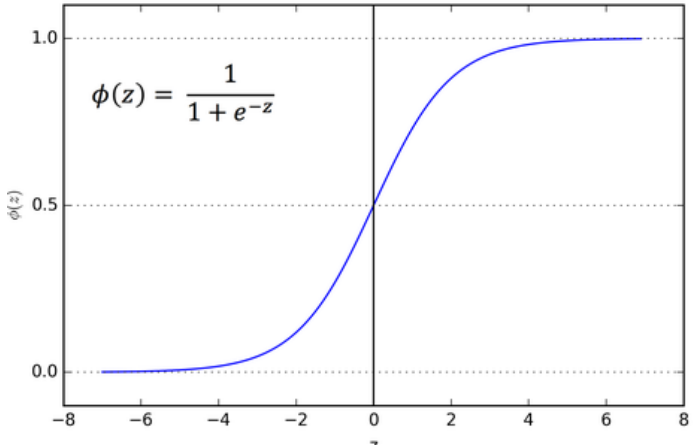
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right)$$

$$= -\sum_d (t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}$$



But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1-o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w}\cdot\vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{i,d}$$

Copyright ACDSD, CSIR-NEIST

Neural Nets for Unsupervised Learning: Self-Organizing Map

$$\Delta w_{kj}(n) = f(y_k(n), x_j(n)) \cong \Delta w_{kj}(n) = \eta y_k(n).x_j(n)$$ *(Hebian Learning for self amplification)*

$$x = [x_1, x_2, \dots, x_m]^T$$

$$w_j = [w_{j1}, w_{j2}, \dots, w_{jm}]^T, j \in [1, l]$$

$$i(x) = \arg\min_j \|x - w_j\|, j \in [1, l]$$

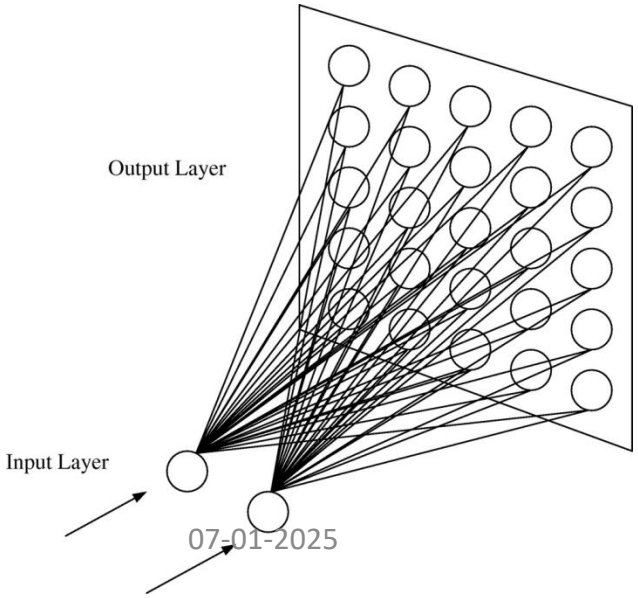
(Competitive Learning)

$$h_{j,i(x)} = \exp(-\frac{d_{j,i}}{2\sigma^2}), j \in [1, l]$$

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) n = 0, 1, 2, \dots$$

$$h_{j,i(x)}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), n = 0, 1, 2, \dots$$

(Cooperation Process)



$$\Delta w = \eta y_j x - g(y_j) w_j$$

$$Let\ g(y_i) = \eta y_j\ and\ y = h_{j,i(x)}, then$$

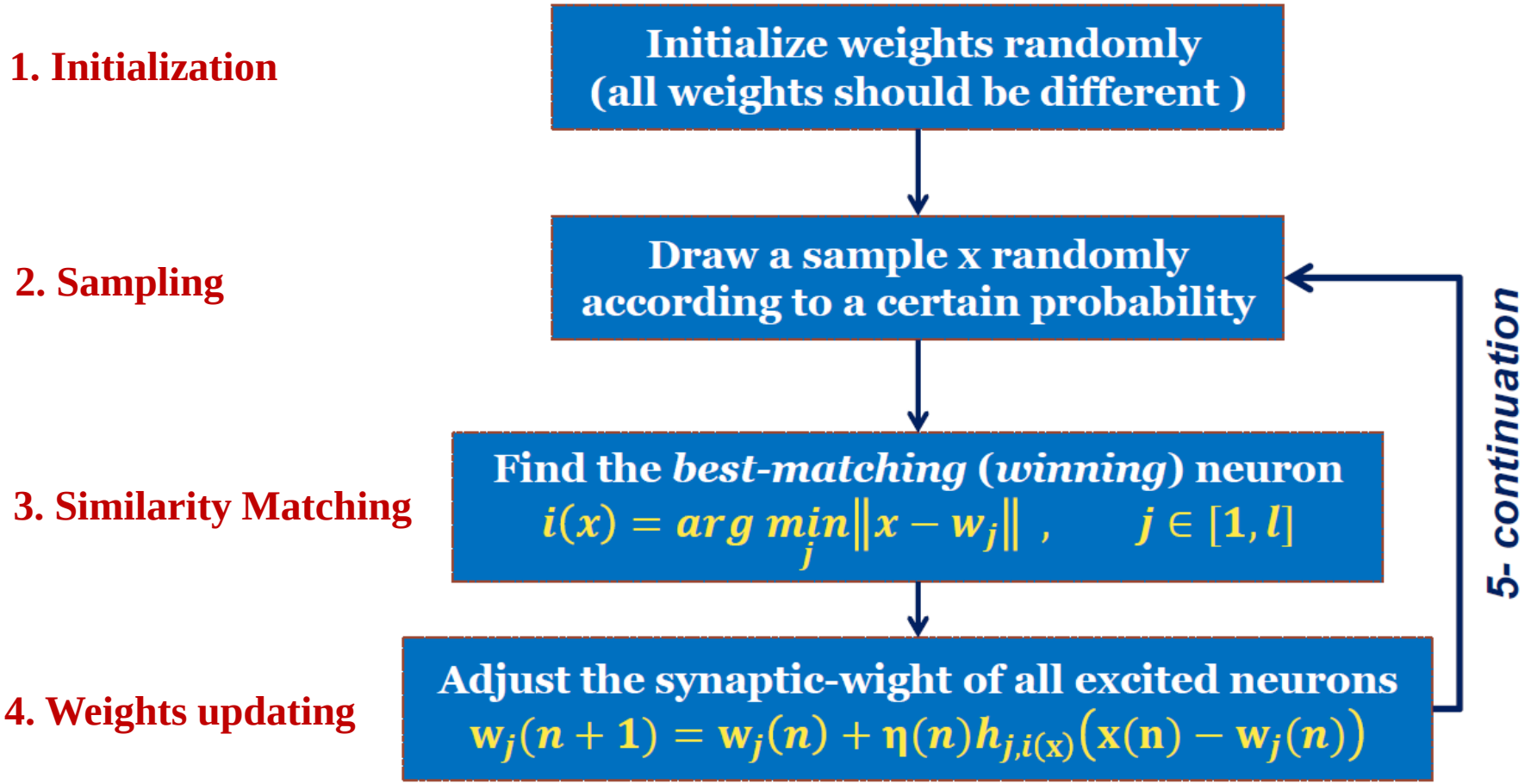
$$\Delta w = \eta h_{j,i(x)}(x - w_j)$$

$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(x(n) - w_j(n))$$

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right) n = 0, 1, 2, \dots$$

(Adaptive Process)

Self-Organizing Map Flowchart



Copyright ACDSD, CSIR-NEIST

Biological Inspiration

Humans perform complex tasks like vision, motor control, or language understanding very well

One way to build intelligent machines is to try to imitate the (organizational principles of) human brain.

Copyright ACDSD, CSIR-NEIST

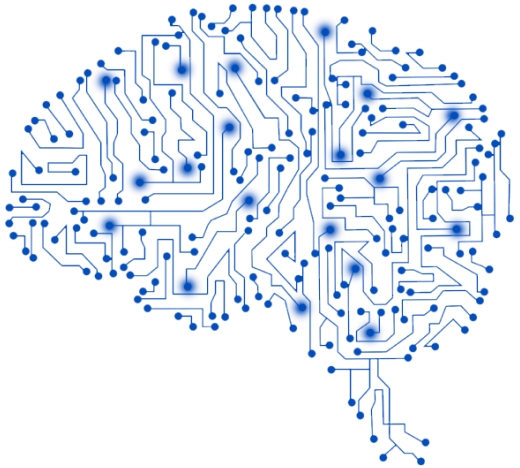
The Human Brain

- The human brain is a highly complex, non-linear, and parallel computer, composed of some 10^{11} neurons that are densely connected (~10^4 connection per neuron).

- A neuron is much slower (10^{-3}sec) compared to a silicon logic gate (10^{-9}sec), however the massive interconnection between neurons make up for the comparably slow rate
  - Complex perceptual decisions are arrived at quickly (within a few hundred milliseconds)

- 100-Steps rule: Since individual neurons operate in a few milliseconds, calculations do not involve more than about 100 serial steps and the information sent from one neuron to another is very small (a few bits)

- Plasticity: Some of the neural structure of the brain is present at birth, while other parts are developed through learning, especially in early stages of life, to adapt to the environment (new inputs).



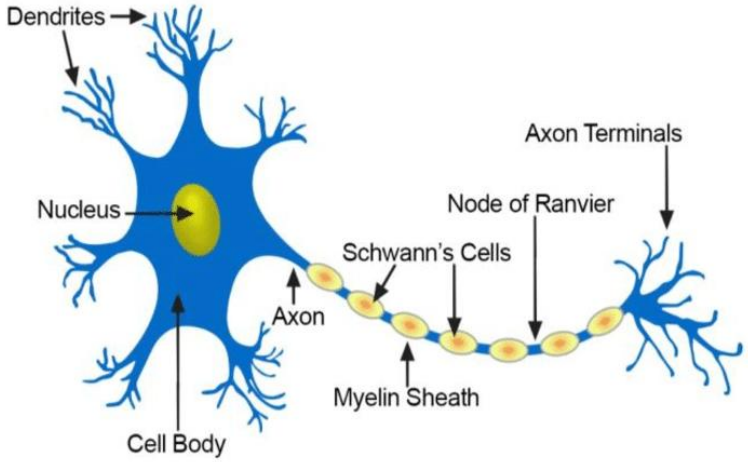
Copyright ACDSD, CSIR-NEIST

Biological Neuron

- A variety of different neurons exist (motor neuron, on-center off-surround visual cells…), with different branching structures.

- The connections of the network and the strengths of the individual synapses establish the function of the network.

– ***dendrites:*** nerve fibers carrying electrical signals to the cell
– ***cell body:*** computes a non-linear function of its inputs
– ***axon:*** single long fiber that carries the electrical signal from the cell body to other neurons
– ***synapse:*** the point of contact between the axon of one cell and the dendrite of another, regulating a chemical connection whose strength affects the input to the cell



Copyright ACDSD, CSIR-NEIST

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 modelled by adding an extra input

- ***v*** is called the induced field of the neuron $v = \sum_{j=0}^{m} w_j x_j$, where $w_0 = b$

- An artificial neuron:
  - computes the weighted sum of its input (called its net input)
  - adds its bias
  - passes this value through an activation function
- We say that the neuron "fires" (i.e. becomes active) if its output is above zero.

$$a_i = f(n_i) = f\left(\sum_{j=1}^{n} w_{ij} x_j + b_i\right)$$

- Bias can be incorporated as another weight clamped to a fixed input of +1.0
- This extra free variable (bias) makes the neuron more powerful.

Copyright ACDSD, CSIR-NEIST

Can we mimic artificially a biological brain?

Biological Neural Network	Artificial Neural Network

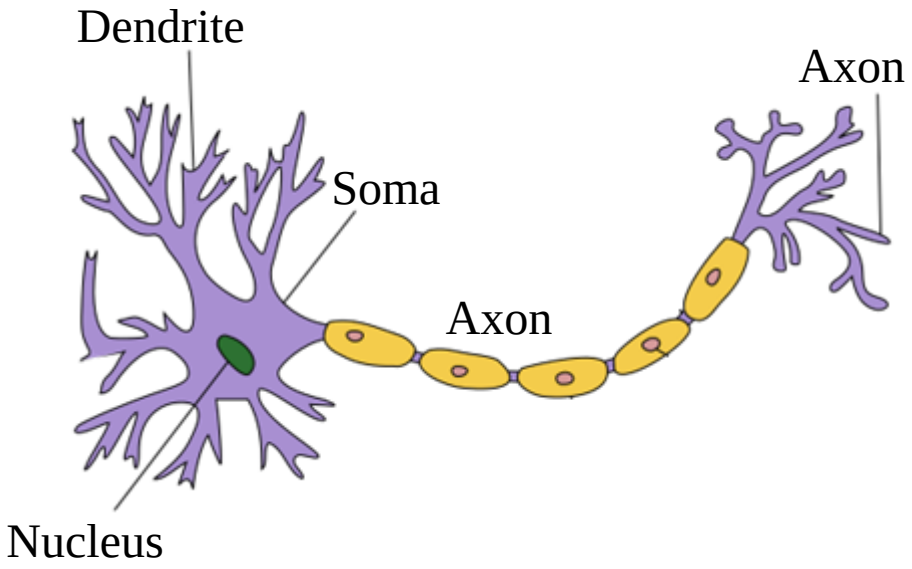
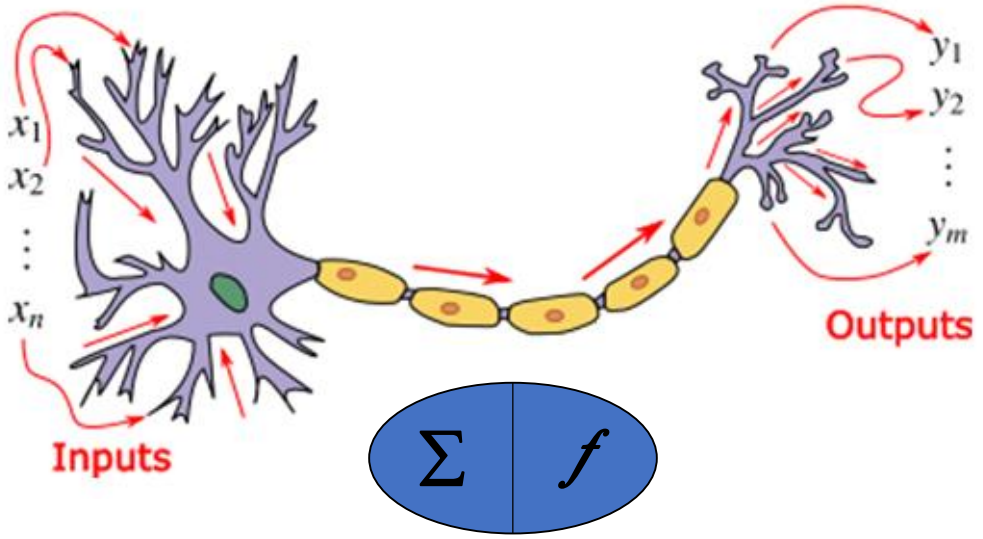

	

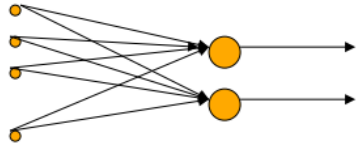
Image source: Wikipedia (https://en.wikipedia.org/wiki/Neuron)	Developed as generalizations of mathematical models of neural biology.

Copyright ACDSD, CSIR-NEIST

Different Network Topologies

Single layer feed-forward networks

– Input layer projecting into the output layer

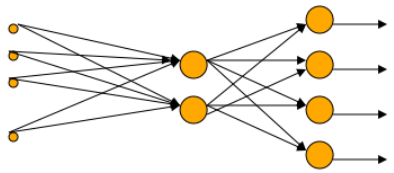


Input layer Output layer

Multi-layer feed-forward networks

– One or more hidden layers.

– Input projects only from previous layers onto a layer **typically, only from one layer to the next**

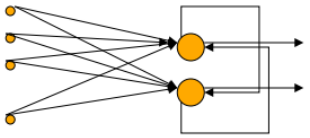


Input layer Hidden layer Output layer

2-layer or 1-hidden layer *fully connected* network

Recurrent networks

– A network with feedback, where some of its inputs are connected to some of its outputs (discrete time)



Input layer Output layer

Copyright ACDSD, CSIR-NEIST

Choosing the Network Topologies

- **Number of input nodes?**
  - Number of features
- **Number of output nodes?**
  - Suitable to encode the output representation
- **Which transfer function?**
  - Suitable to the problem
- **Number of hidden nodes?**
  - Not exactly known

Copyright ACDSD, CSIR-NEIST

The Topics Covered In This Section

11.1.1 Motivation
11.1.2 BNN vs ANN
11.1.3 Definition of ANN
11.1.4 Applications

Copyright ACDSD, CSIR-NEIST

11.1.1 Motivation

Can we mimic artificially a human biological brain?

Biological Neural Network	Artificial Neural Network

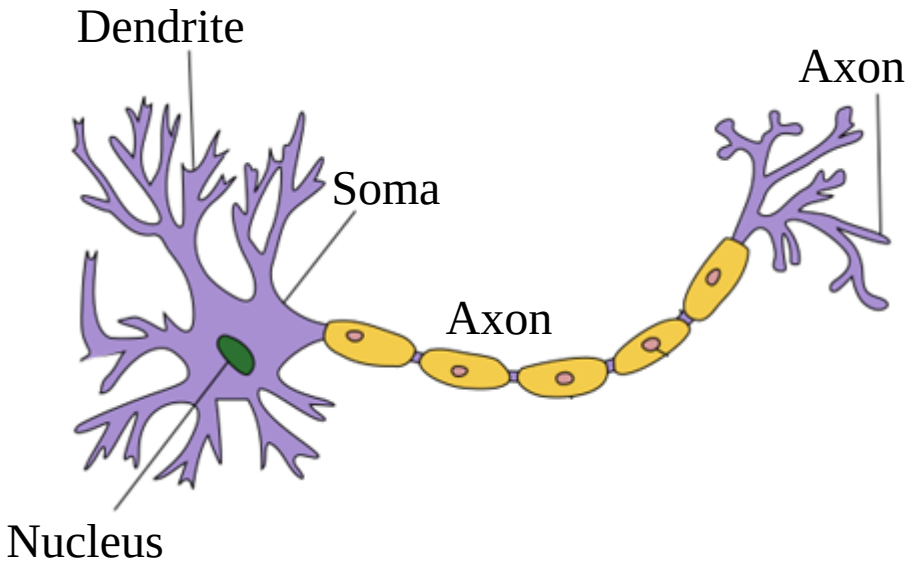
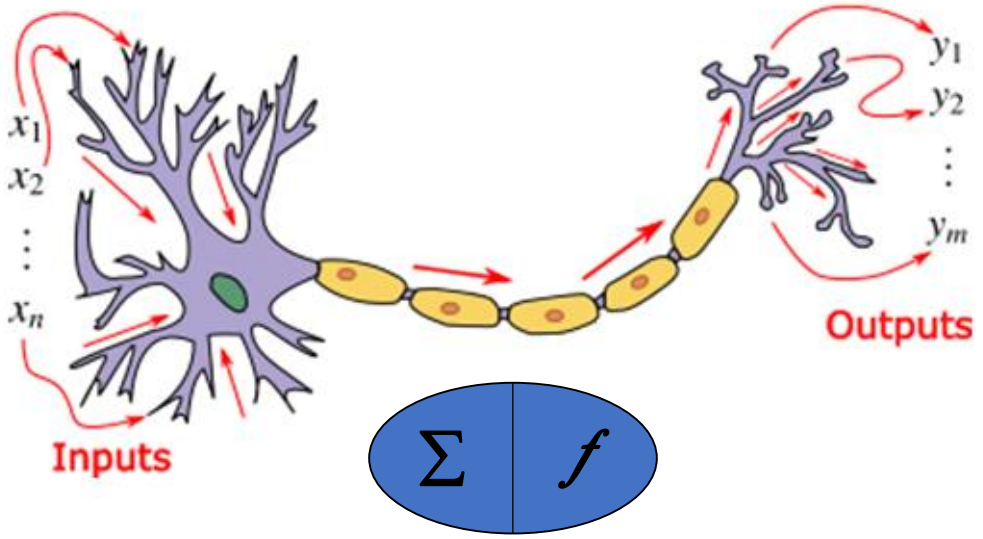

	

Image source: Wikipedia (https://en.wikipedia.org/wiki/Neuron)	Developed as generalizations of mathematical models of neural biology.

Copyright ACDSD, CSIR-NEIST

A biological neuron has three types of main components –

- **Dendrites:** Receives input signals (electrical) from other neurons
- **Soma (cell body):** Sums the incoming signals
- **Axon:** When sufficient input is received (greater than a threshold), the cell fires; that is it transmit a signal over its axon to other connected neuron cells.

Similarly compared to BNN, ANN has three types of main components –

- **Input:** Receives input signals from other neurons
- **Node (neuron):** Sums the incoming weighted inputs
- **Output:** When sufficient input is received (greater than a threshold), the neuron fires; that is it transmit a signal over its output to other connected neurons.

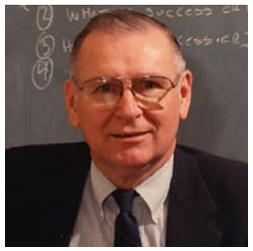
Copyright ACDSD, CSIR-NEIST

Biological	Artificial
Soma	Node
Dendrites	Input
Axon	Output
Synapse	Weight
Many Neurons	Few Neurons
Learning: Involves adjustments to the synaptic connections between neurons	Learning: Involves adjustments to the weights connections between neurons

Copyright ACDSD, CSIR-NEIST

11.1.3 Definition of ANN

LefTeri H. Tsoukalas & Robert E. Uhrig [1997]

Data processing system consisting of a large number of simple, highly interconnected processing elements (neurons) in an architecture inspired by the structure of the cerebral cortex of the brain.

Simon S. Haykin [1994]

A neural network is a massively parallel distributed processor that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

Copyright ACDSD, CSIR-NEIST

When To Use

- Problems where we can't formulate an algorithmic solution.
- Problems where we can get lots of examples of the behavior we require.
- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Noise in training examples.
- Problems where we need to pick out the structure from existing data.
- If long training time is acceptable.

Problems where applied

- Applied to following types of problems from real world –
  - Regression
  - Classification
  - Clustering
  - Association and Pattern Recognition

Copyright ACDSD, CSIR-NEIST

11.1.4 Applications

- Regression:
  - Forecasting (prediction on base of past history)
  - Forecasting e.g., predicting behavior of stock market
- Classification:
  - Image recognition
  - Speech recognition
  - Diagnostic
  - Fraud detection
  - Face recognition
- Clustering:
  - Clients profiles
  - Disease subtypes
- Associations and Pattern recognition
  - Network attacks
  - Breast cancer
  - Handwriting recognition
  - Retrieve an image from corrupted one

Copyright ACDSD, CSIR-NEIST

11.2 Overview – Architecture and Learning

The Topics Covered In This Section

11.2.1 A Brief History
11.2.2 McCulloch Pitts Model
11.2.3 Single Layer Perceptron
11.2.4 Multi Layer Perceptron
11.2.5 Feed Forward Neural Network
11.2.6 Feed Backward Neural Network
11.2.7 Radial Basis Function Network
11.2.8 Kohonen Map Network
11.2.9 Hopfield Network
11.2.10 Hybrid Neural Network
11.2.11 Towards Deep Learning
11.2.12 Learning Paradigm

Copyright ACDSD, CSIR-NEIST

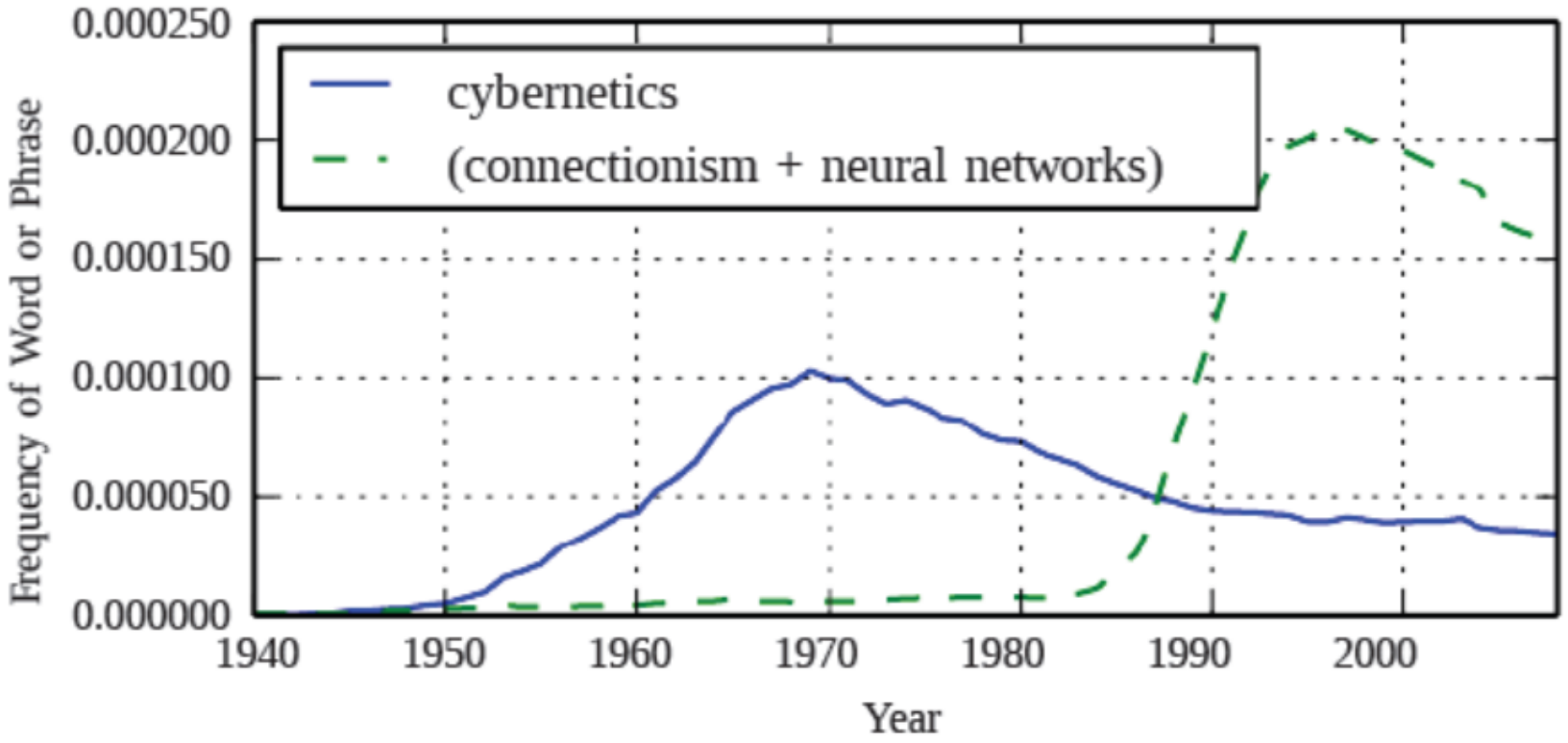
11.2.1 A Brief History

Year	Achievement
1943	McCulloch-Pitts neurons (McCulloch and Pitts) – Mankind first NN
1958	Single Layer Perceptron (Rosenblatt) – Learned its own weight values; convergence proof
1965	Multi Layer Perceptron (Alexey Grigorevich Ivakhnenko) – Introduced multiple hidden layers
1969	Limitations of perceptron (Minsky, Papert) – Proved limitations of single-layer perceptron networks
1972	Kohonen network (Kohonen)
1982	Hopfield network (Hopfield) – Introduced energy-function concept
1986	Backpropagation re-discovered (Rumelhart, Hinton, McClelland) – Method for training multilayer nets
1990	Radial Basis Function Network
2000 -	Breakthrough in Deep learning

Copyright ACDSD, CSIR-NEIST

Three Waves of ANN throughout time:

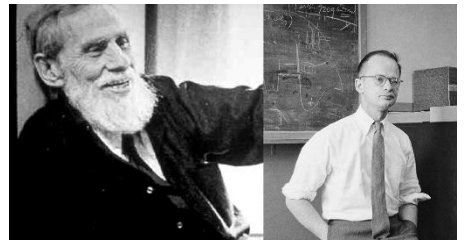
1) Cybernetics – MCP, Hebbian Learning, Single Layer perceptron.
2) Connectionism or Neural Network – Perceptron or FFNN, Backpropagation, SoM etc.
3) Deep Learning – CNN, RNN, GAN etc.



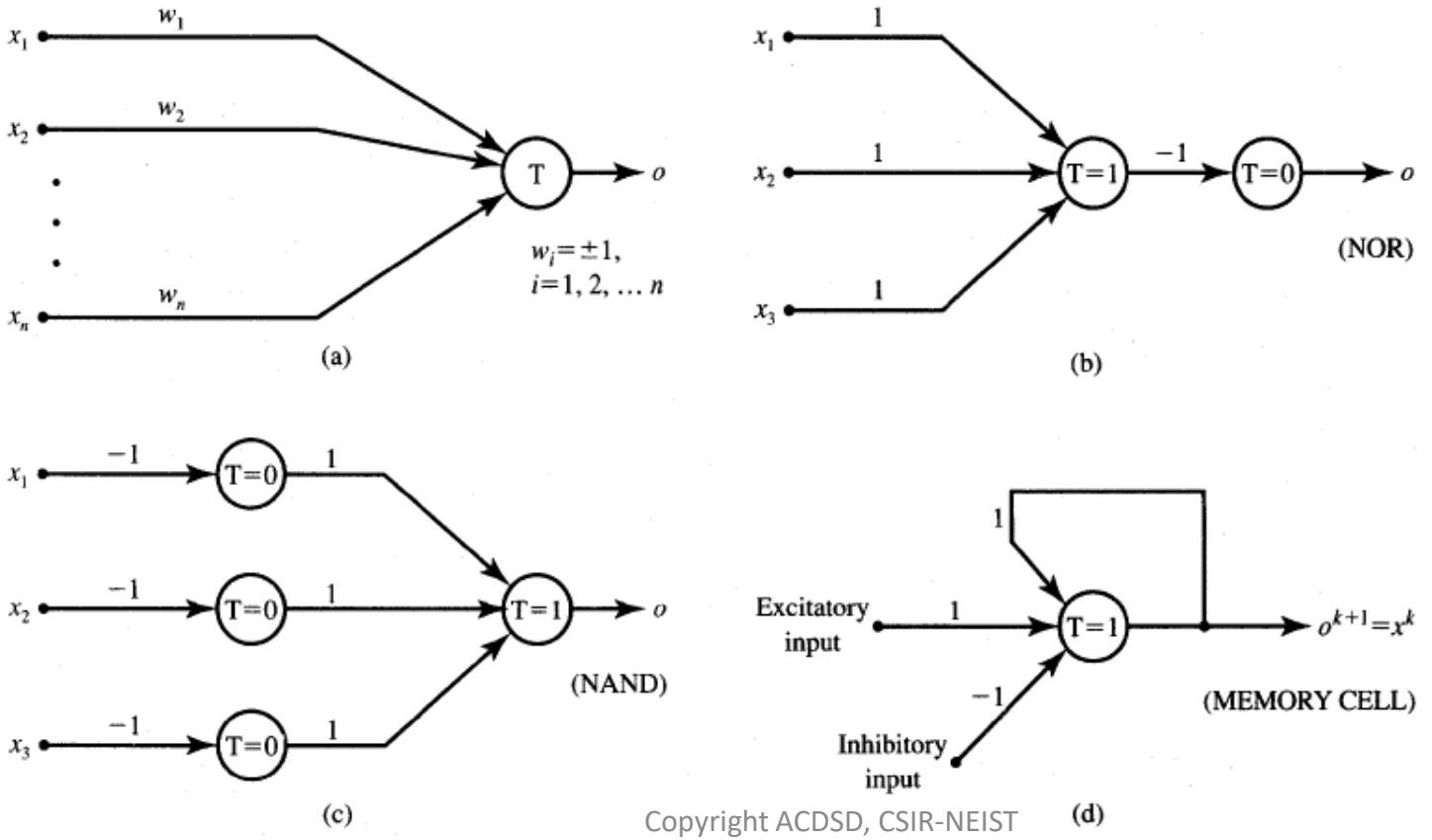
Copyright ACDSD, CSIR-NEIST

11.2.2 McCulloch Pitts (MCP) Model

- First introduced by Warren McCulloch and Walter Pitts in 1943.



$$o^{k+1} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$



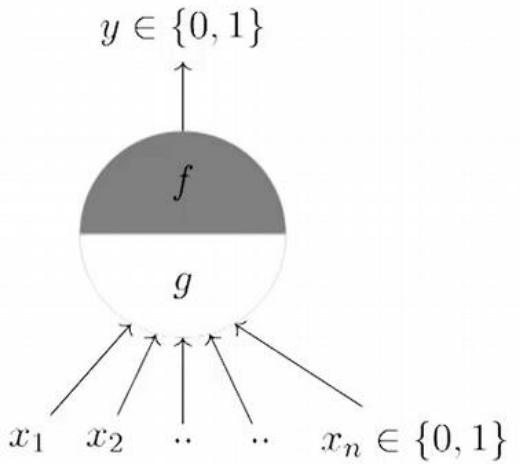
Copyright ACDSD, CSIR-NEIST

11.2.2 McCulloch Pitts (MCP) Model

Working:

- g aggregates the inputs.
- Function f takes a decision based on the aggregation.
- Θ is thresolding parameter.

Note: Inputs can be either excitatory or inhibitory.



$$g(x_1, x_2, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

Advantages:

- Simple model

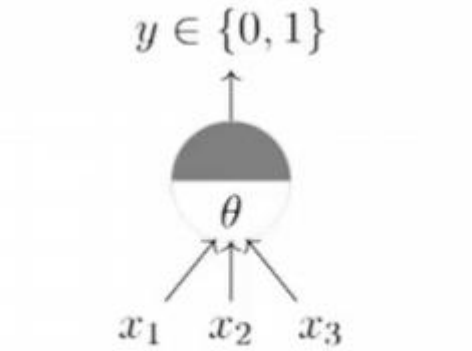
Disadvantages:

- Just a primitive model.
- All input nodes are limited to Boolean values.
- Need to hand code the threshold.
- A single MP model can represent linearly separable Boolean functions.

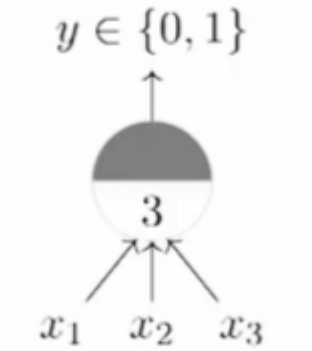
Copyright ACDSD, CSIR-NEIST

11.2.2 McCulloch Pitts (MCP) Model

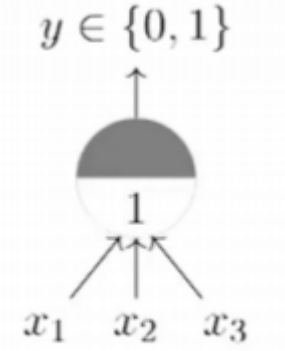
Capabilities:



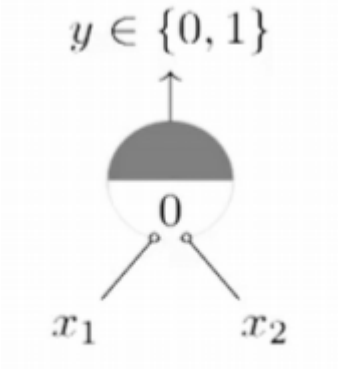
A McCulloch Pitts unit



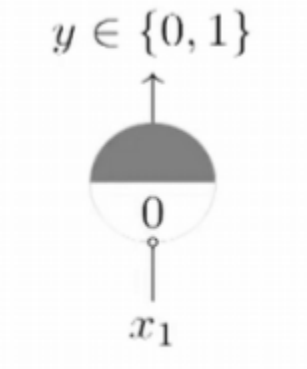
AND function



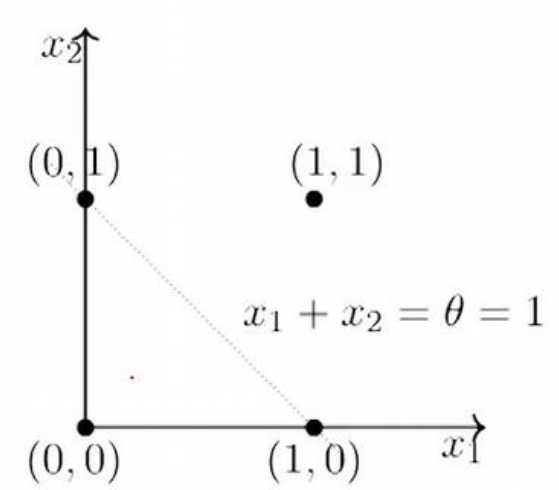
OR function



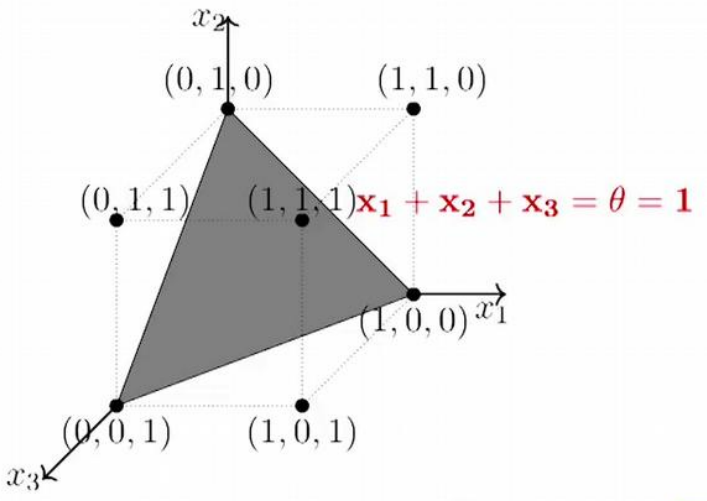
NOR function



NOT function



OR function

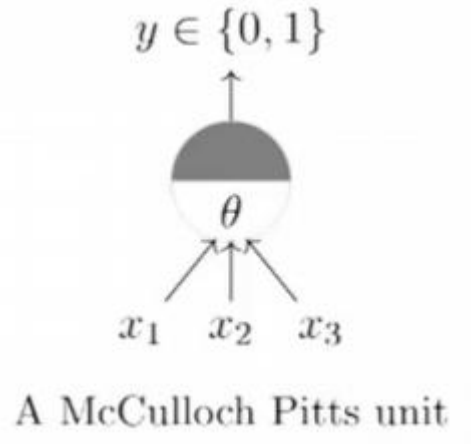


OR function

Copyright ADSD, CSIR-NEIST

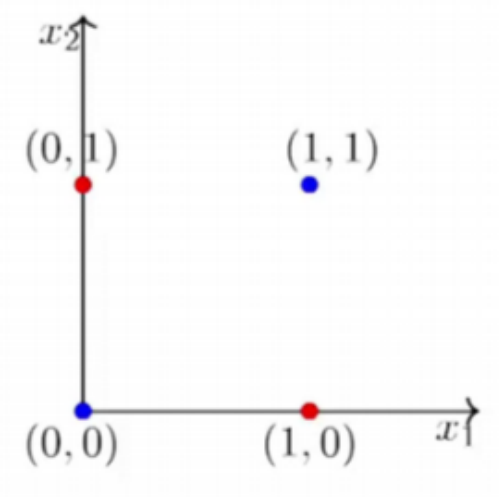
11.2.2 McCulloch Pitts (MCP) Model

Limitations:



A McCulloch Pitts unit

XOR		
X1	**X2**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

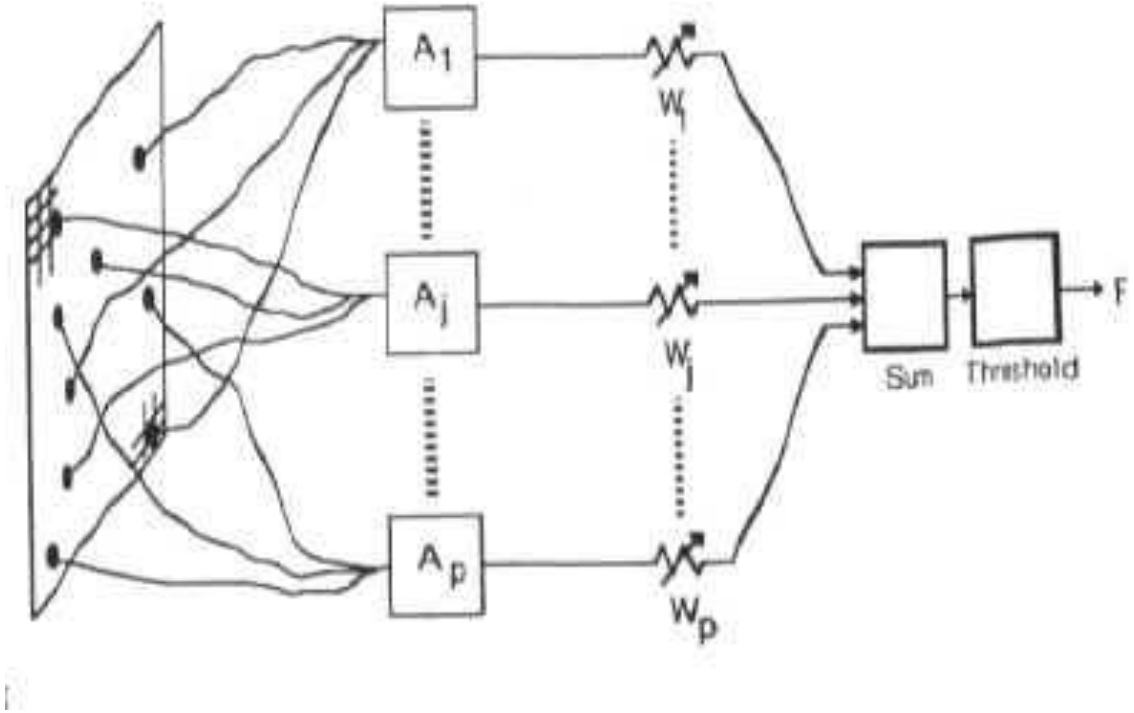


Θ = ?

Copyright ACDSD, CSIR-NEIST

11.2.3 Single Layer Perceptron

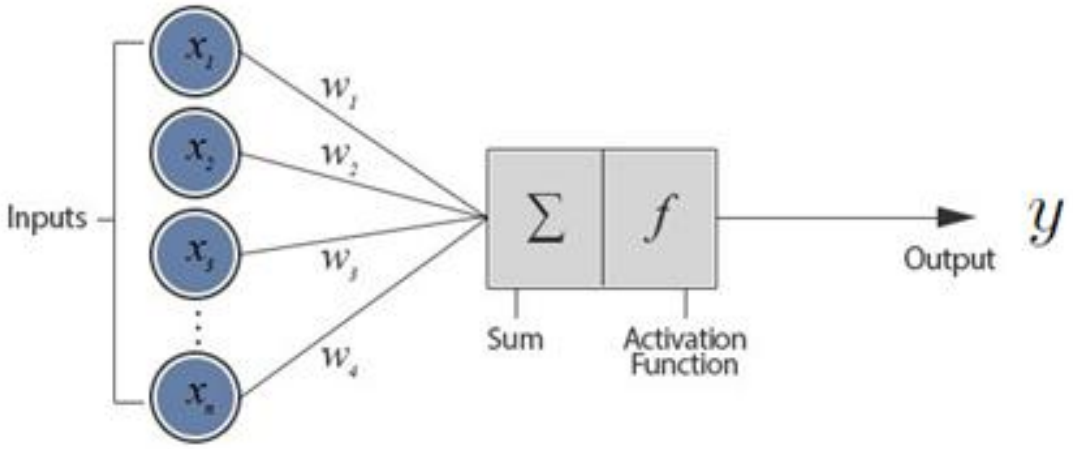
- First Introduced by Frank Rosenblatt (psychologist) in 1957.



Copyright ACDSD, CSIR-NEIST

11.2.3 Single Layer Perceptron

Working:



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Advantages:

- Introduction to numerical weights for inputs (Can give more importance to some node).
- Mechanism for learning the parameter (No need to hand code).
- Inputs are not limited to Boolean values (Real valued inputs and binary outputs).

Disadvantages:

- A single MP model can represent linearly separable Boolean functions *i.e.* classes which can be divided by a line or hyper-plane.

Copyright ACDSD, CSIR-NEIST

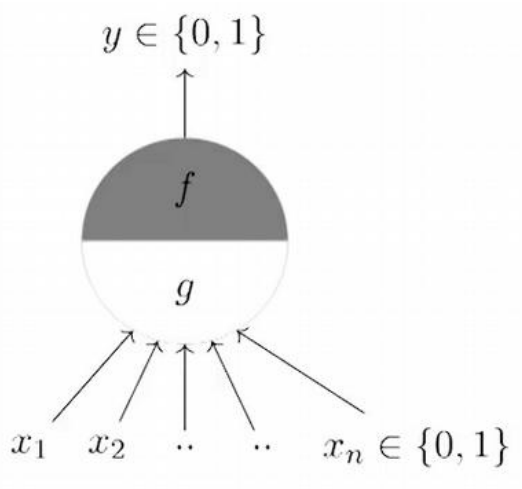
11.2.3 Single Layer Perceptron

MCP vs Perceptron:

McCulloch Pitts Neuron

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq \theta$$

$$= 0 \quad if \sum_{i=0}^{n} x_i < \theta$$



Perceptron

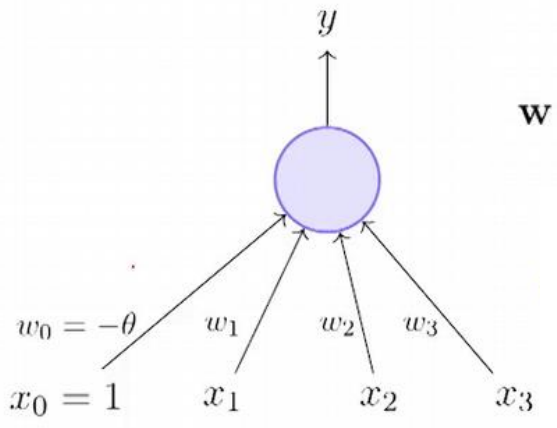
$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < \theta$$

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$

$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$

$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathbf{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$



Copyright ACDSD, CSIR-NEIST

11.2.3 Single Layer Perceptron

Capabilities:

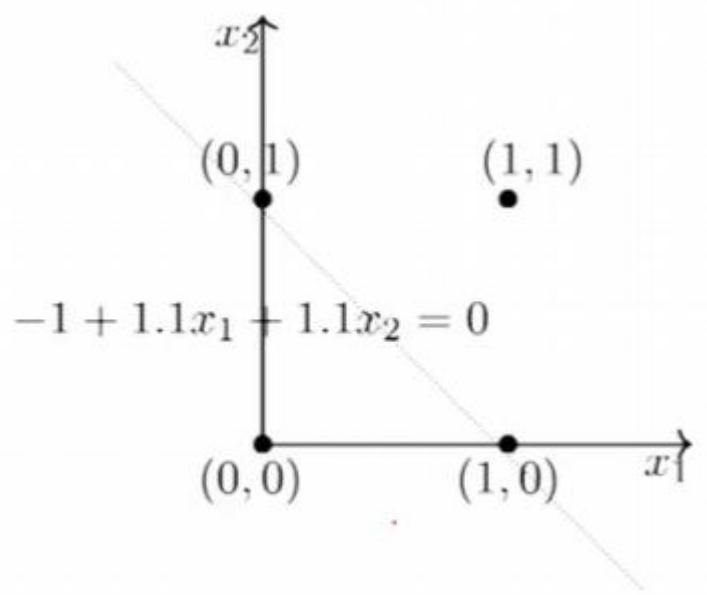
x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$W_1*0 + W_2*0 + W_0 < 0 \quad \Rightarrow W_0 < 0$

$W_1*0 + W_2*1 + W_0 \geq 0 \quad \Rightarrow W_2 \geq - W_0$

$W_1*1 + W_2*0 + W_0 \geq 0 \quad \Rightarrow W_1 \geq - W_0$

$W_1*1 + W_2*1 + W_0 \geq 0 \quad \Rightarrow W_1 + W_2 \geq - W_0$

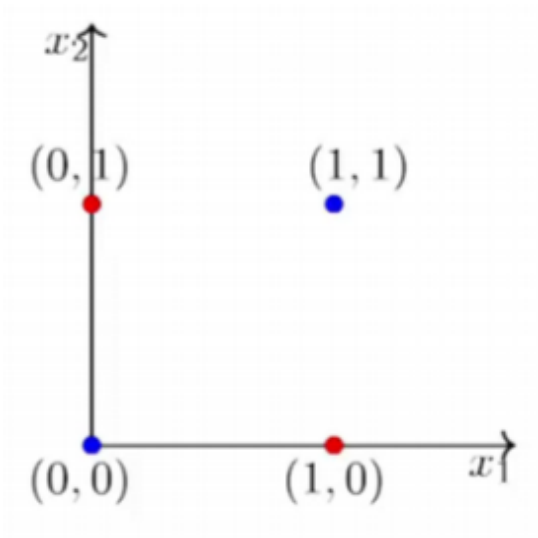


Copyright ACDSD, CSIR-NEIST

11.2.3 Single Layer Perceptron

Limitations:

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$



$W_1*0 + W_2*0 + W_0 < 0 \quad \Rightarrow W_0 < 0$

$W_1*0 + W_2*1 + W_0 \geq 0 \quad \Rightarrow W_2 \geq - W_0$

$W_1*1 + W_2*0 + W_0 \geq 0 \quad \Rightarrow W_1 \geq - W_0$

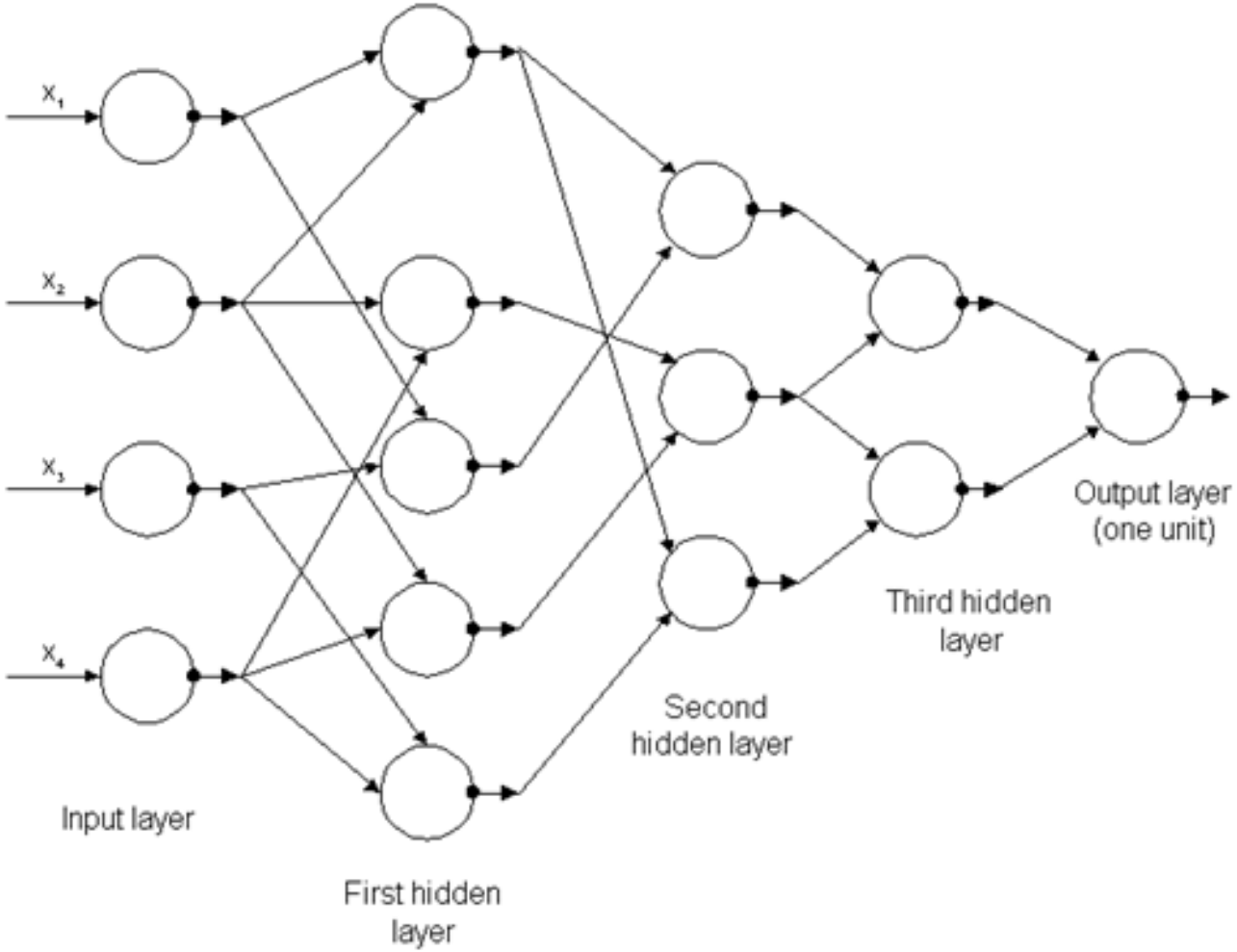
$W_1*1 + W_2*1 + W_0 < 0 \quad \Rightarrow W_1 + W_2 < - W_0$

- No solution (Because 4th condition contradicts condition 2nd and 3rd)
- XOR requires two separation hyperplanes!
- Can use other tricks (e.g. complicated threshold functions) but complexity blows up
- Thus a more complex network needed that combine simple perceptrons to address non linear classification tasks.

Copyright ACDSD, CSIR-NEIST

11.2.4 Multi Layer Perceptron

- First Generation MLP introduced by Alexey Grigorevich Ivakhnenko in 1965.



Copyright ACDSD, CSIR-NEIST

11.2.4 Multi Layer Perceptron

Working:

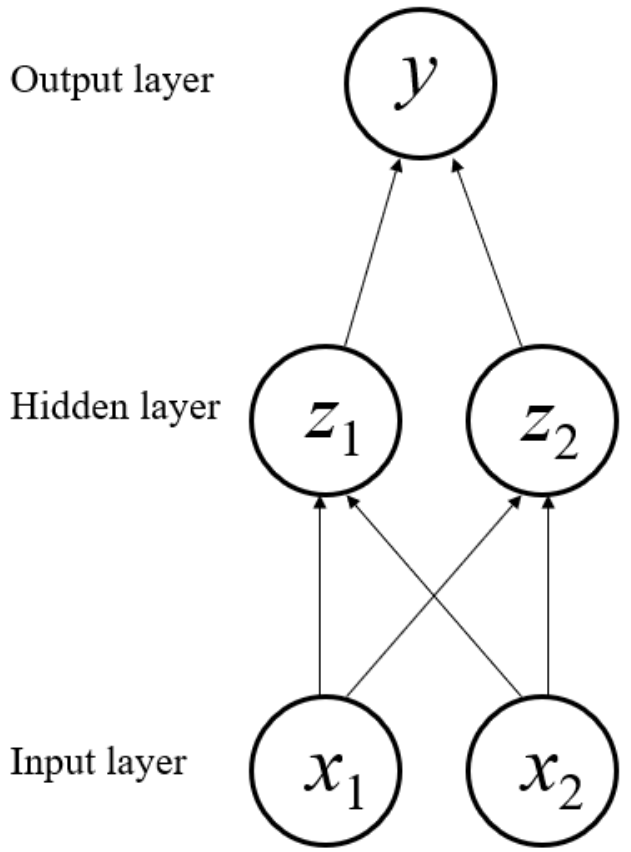
- Same as perceptron but with multiple layers.

Advantages:

- MLP can represent non-linear Boolean functions.

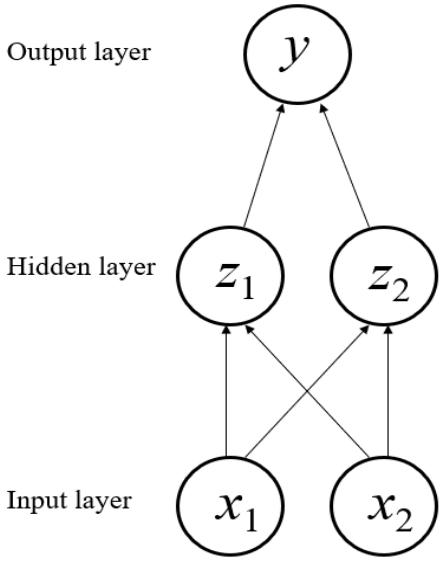
Disadvantages:

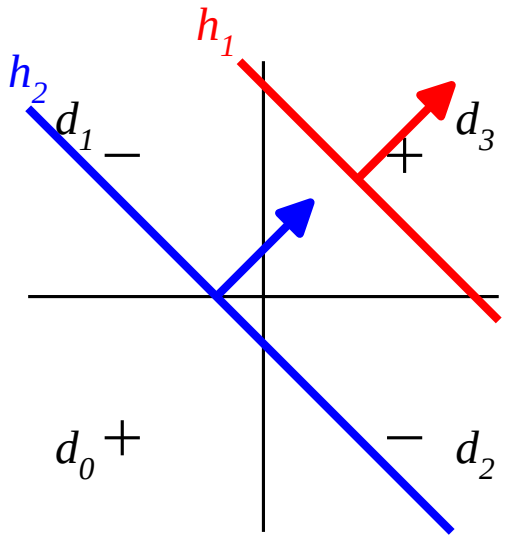
- Can not realize any continuous function.

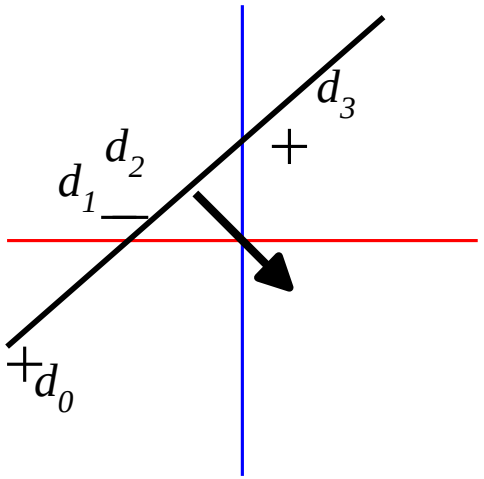


Copyright ACDSD, CSIR-NEIST

Capabilities:



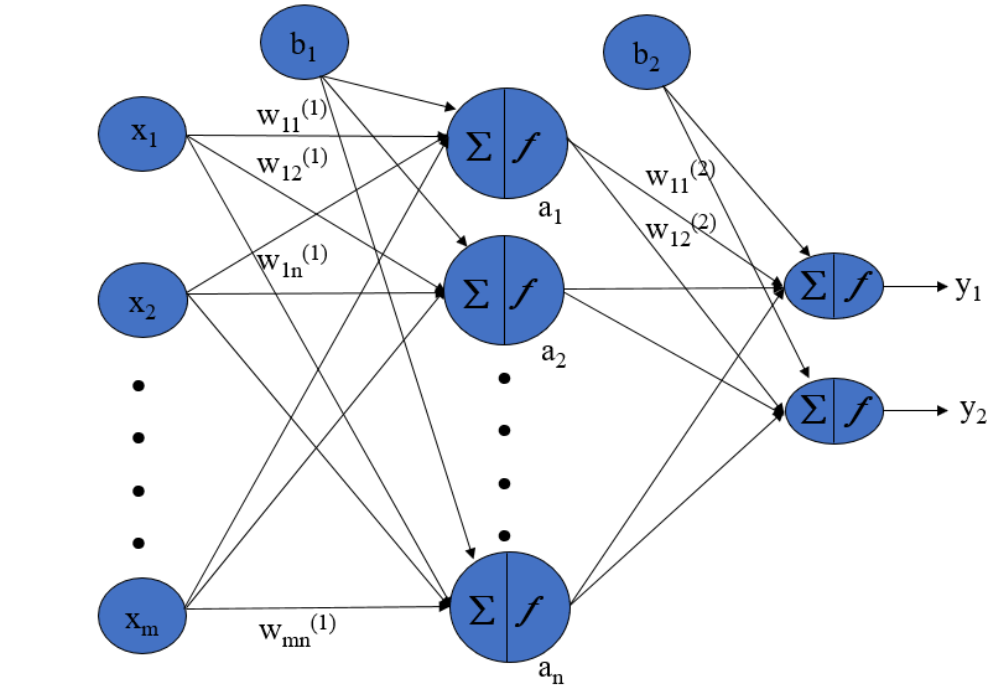


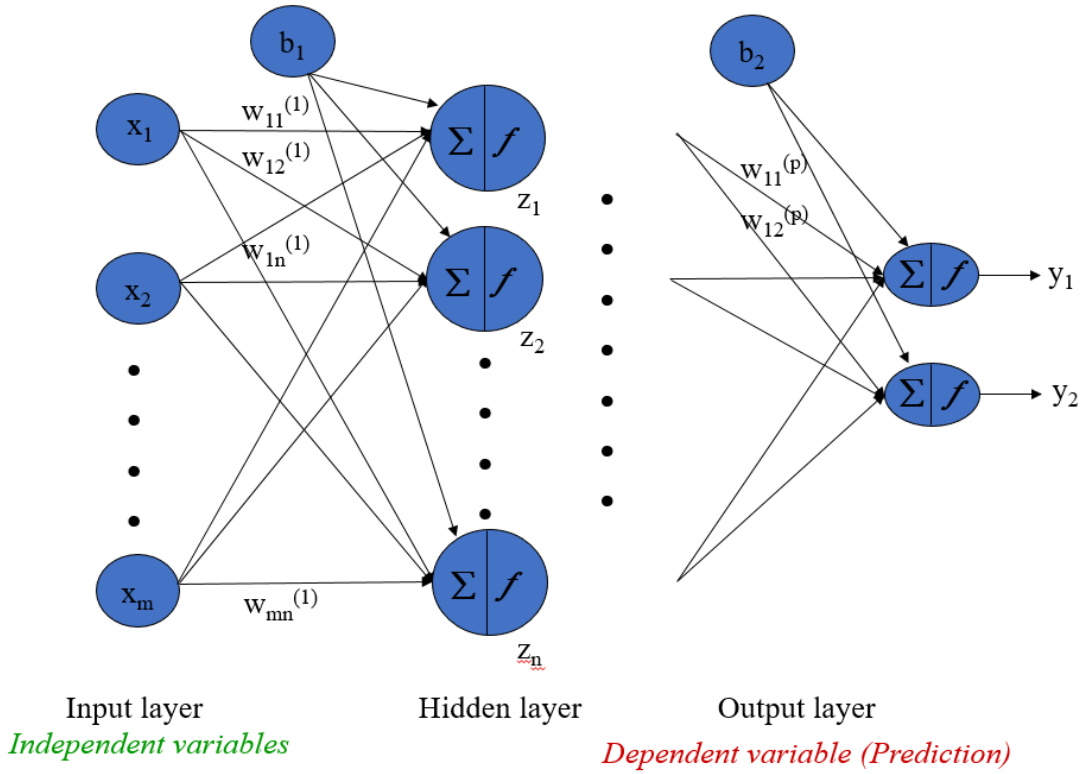


Single layer	Multi layer
Can not represent non linear function	Can represent any linear and non linear functions
Insufficiently expressive; they are non-linear i.e. can not represent non-linear and continuous functions	Sufficiently expressive; can represent any Boolean and continuous functions using appropriate activation functions
Can not deal with generalized data sets	Can be trained to deal with generalized data sets, i.e. via error back-propagation

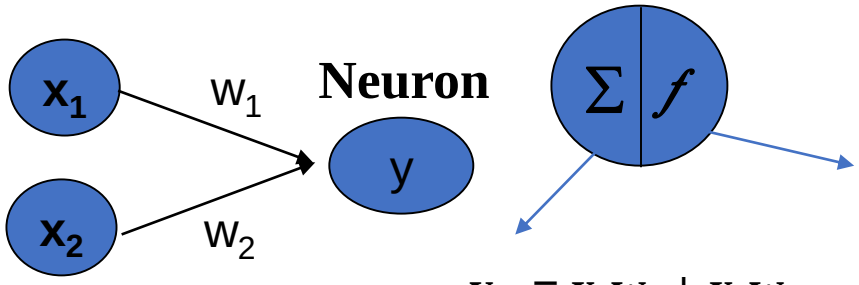
Copyright ACDSD, CSIR-NEIST

11.2.5 Feed Forward Neural Network





Deep Neural Network → ANN with more than 3 hidden layer



$y_{in} = x_1w_1 + x_2w_2$

Activation Function:

if $y_{in} >= \theta$ $\quad f(y_{in}) = 1$

else $\quad f(y_{in}) = 0$

Copyright ACDSD, CSIR-NEIST

11.2.5 Feed Forward Neural Network

Pros

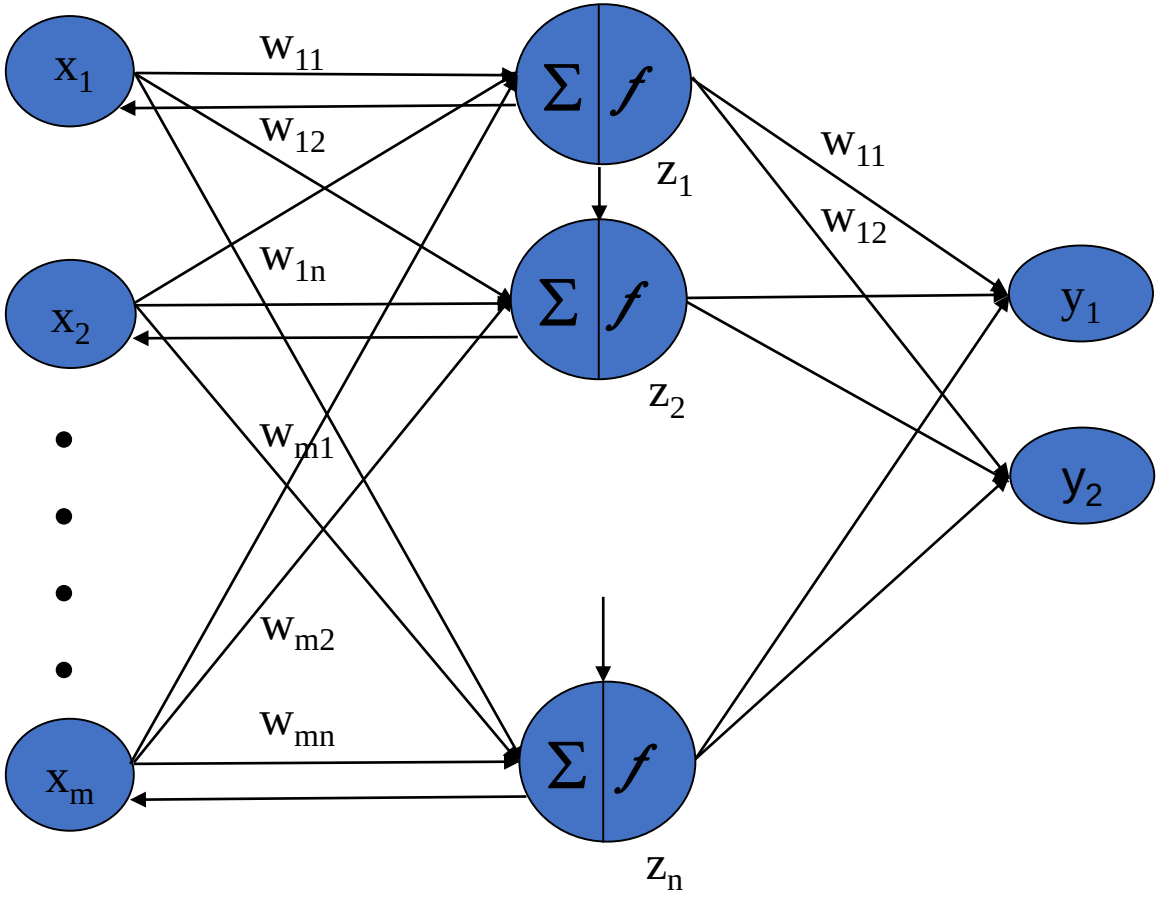
- They are extremely powerful computational devices (Turing equivalent, universal computers).
- Able to deal with high dimensional nonlinear data and high tolerance to noisy data.
- Can handle variety of problem types (Handles both numerical and categorical variables).
- ANN can learn and generalize from training data – no need for enormous feats of programming.
- Lack of cycles - computation proceeds uniformly from input units to output units.

Cons

- They are deemed to be black-box solutions, lacking expandability and Poor interpretability.
- Hard to find optimal values of parameters, determined empirically which effect output.
- Training may take a long time for large datasets; which may require case sampling.
- May converge to a local, not global, minimum of error.
- Overfitting due to repeated training to reduce error.
- Being able to learn anything can make it harder to learn specific things - “bias-variance tradeoff”.

Copyright ACDSD, CSIR-NEIST

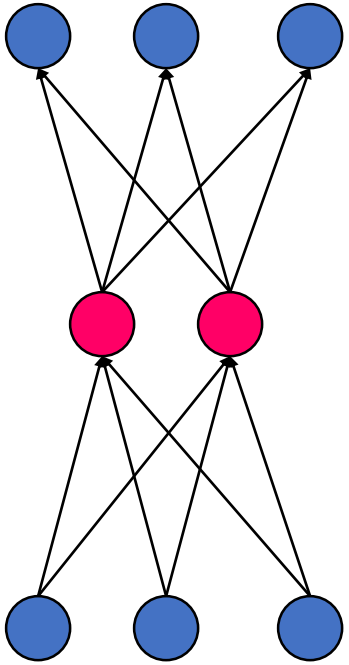
11.2.6 Feed Backward Neural Network

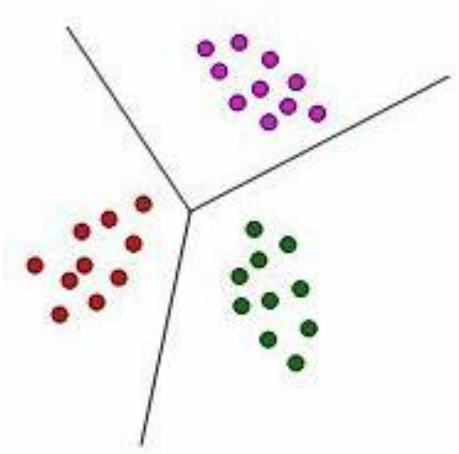


Copyright ACDSD, CSIR-NEIST

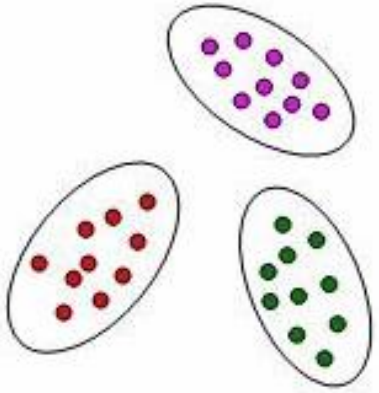
11.2.7 Radial Basis Function Network

- Typically used for function approximation, pattern classification etc.
- Two layer feed-forward structure with each hidden unit implementing radial activated function
- Training involves updating centres of network for hidden neuron and output layer weights.





MLP

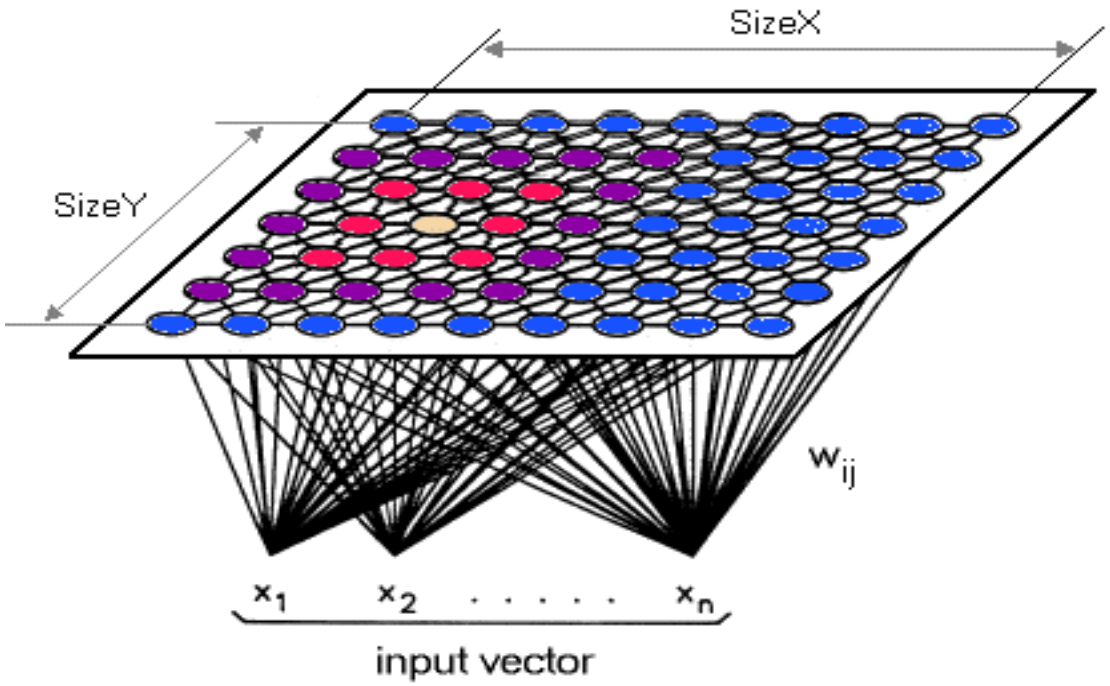


RBF

Copyright ACDSD, CSIR-NEIST

11.2.8 Kohonen Map Network

- First introduced by the Finnish Professor Teuvo Kohonen.
- Applies to clustering type problems.



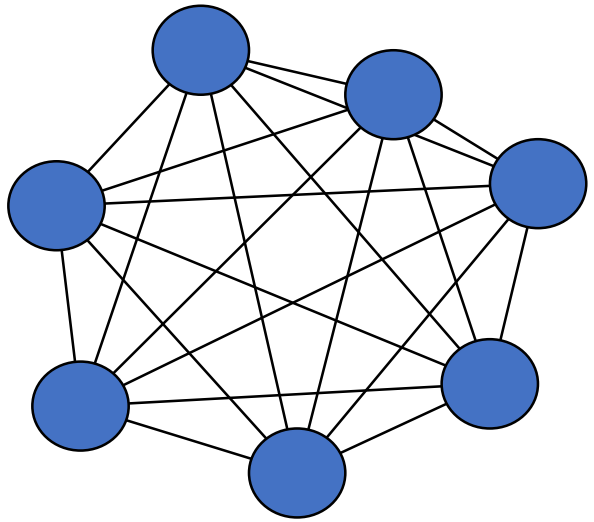
- In unsupervised learning there is no feedback.
- Network must discover patterns, regularities, features for the input data over the output.
- While doing so the network might change in parameters - this process is called self-organizing.

Copyright ACDSD, CSIR-NEIST

11.2.9 Hopfield Network

- First introduced by John Hopfield
- Highly interconnected neurons
- Applies to solving complex computational problems (e.g., optimization, association)

- McCulloch-Pitts neurons with outputs -1 or 1, and threshold Θ
- Symmetric weights $(w_{ij} = w_{ji})$ and $w_{ii} = 0$



Completely Connected

Copyright ACDSD, CSIR-NEIST

11.2.10 Hybrid Neural Network

Advantages: Improved accuracy and Enhanced flexibility

Using SOM and MLP

- SOM employing unsupervised learning used for clustering
- MLP employing Back Propagation Algorithm used for classification
- Output from SOM is given as input to MLP

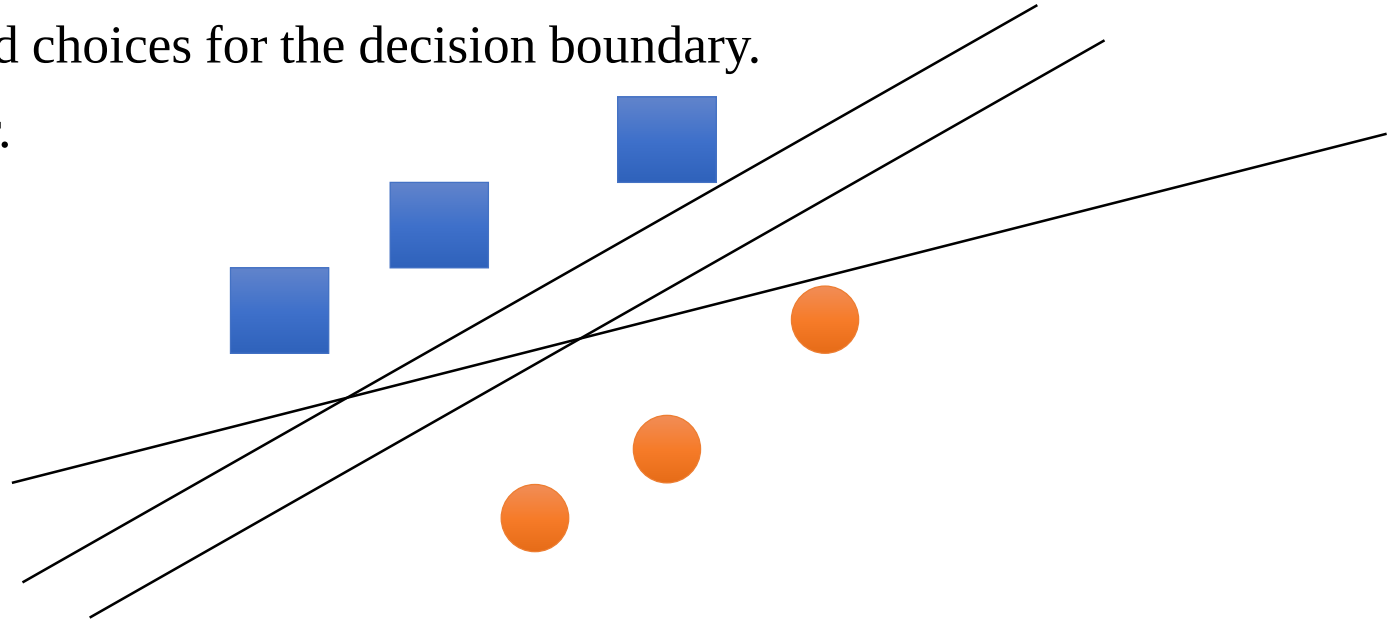
Using SOM and RBF

- SOM employing unsupervised learning used for clustering
- RBF for classification
- Output from SOM is given as input to RBF network

Copyright ACDSD, CSIR-NEIST

ANN with SVM

- ANN can lead to many equally valid choices for the decision boundary.
- SVM helps to best margin classifier.



Bayesian Neural Network

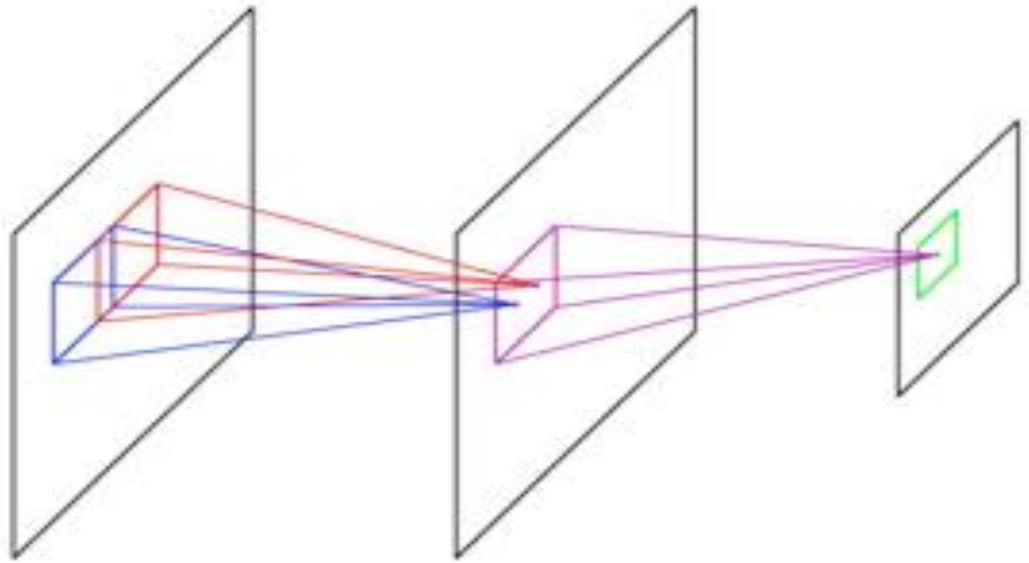
- Bayesian Logistic Regression by inserting a prior on the weights
  - Equivalent to L2 Regularization
- Error Backprop then becomes Maximum A Posteriori (MAP) rather than Maximum Likelihood (ML) training

$$R(\theta) = \frac{1}{N}\sum_{n=0}^{N} L(y_n - f(x_n)) + \lambda||\theta||^2$$

Copyright ACDSD, CSIR-NEIST

Convolutional Neural Network

- The network is not fully connected.
- Different nodes are responsible for different regions of the image.
- This allows for robustness to transformations.

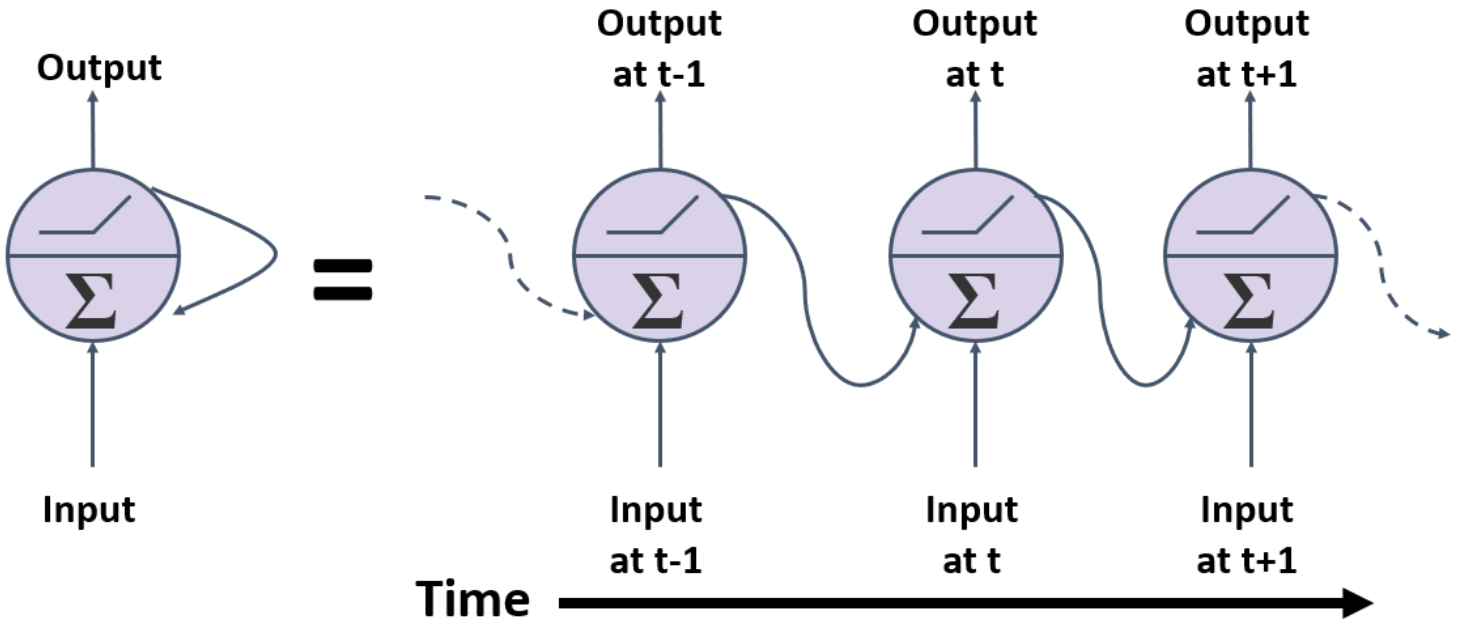


Input Image Convolutional Layer Subsampling Layer

Copyright ACDSD, CSIR-NEIST

Recurrent Neural Network

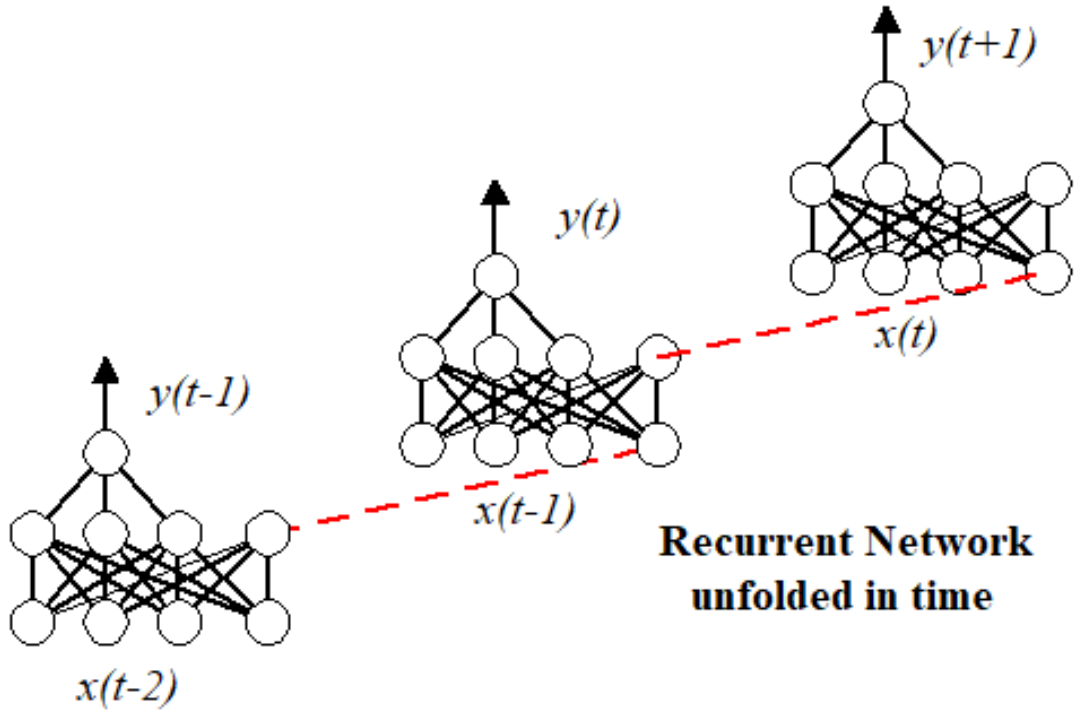
- Output or hidden layer information is stored in a context or memory layer.
- Recurrent networks have at least one feedback connection:
  - They have directed cycles with delays: they have internal states (like flip flops).
  - Creates an internal state of the network which allows it to model short-time memory
  - Response to an input depends on initial state which may depend on previous inputs.



Copyright ACDSD, CSIR-NEIST

RNN Through Time

- Employ feedback (positive, negative, or both)
- Can learn temporal patterns (time series)



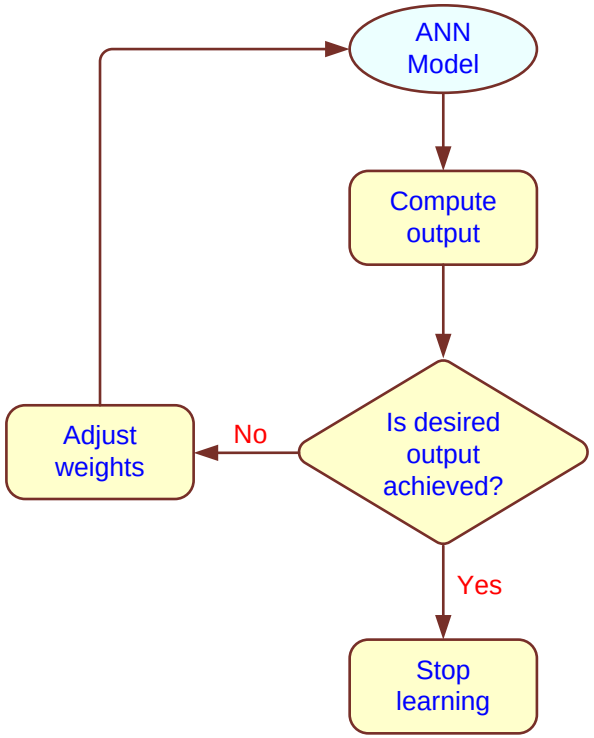
- Examples
  - ***Hopfield network***
  - ***Boltzmann machine*** (Hopfield-like net with input & output units)

- Recurrent backpropagation networks: for small sequences, unfold network in time dimension and use backpropagation learning

Copyright ACDSD, CSIR-NEIST

Supervised Learning

- Compute temporary outputs.
- Compare outputs with desired targets.
- Adjust the weights based on learning algorithm and repeat the process.



Unsupervised Learning

- No feedback to say how output differs from desired output (no error signal) or even whether output was right or wrong.
- Network must discover patterns in the input data by itself.
- NN modifies the weights so that the most similar input vectors are assigned to same output unit.

Copyright ACDSD, CSIR-NEIST

11.2.12 Learning Paradigm

	Supervised	Unsupervised
Data	Labeled examples (input, desired output)	Unlabeled examples (different realizations of the input)
Tasks	Classification Regression	Clustering Association
NN models	Feed-forward NN Radial Basis Function NN ***Deep Learning:*** ***CNN, RNN***	Kohenen Self-organizing maps Hopfield Networks ***Deep Learning:*** ***Boltzman Machine, Autoencoder***
Learning	Delta rule(Backpropagation) – FFNN Radial Basis – RBFNN	Competitive Learning – SOM Hebbian Learning – Hopfield Network

Note: Capability of ANN largely depends on the learning algorithm and network architecture used.

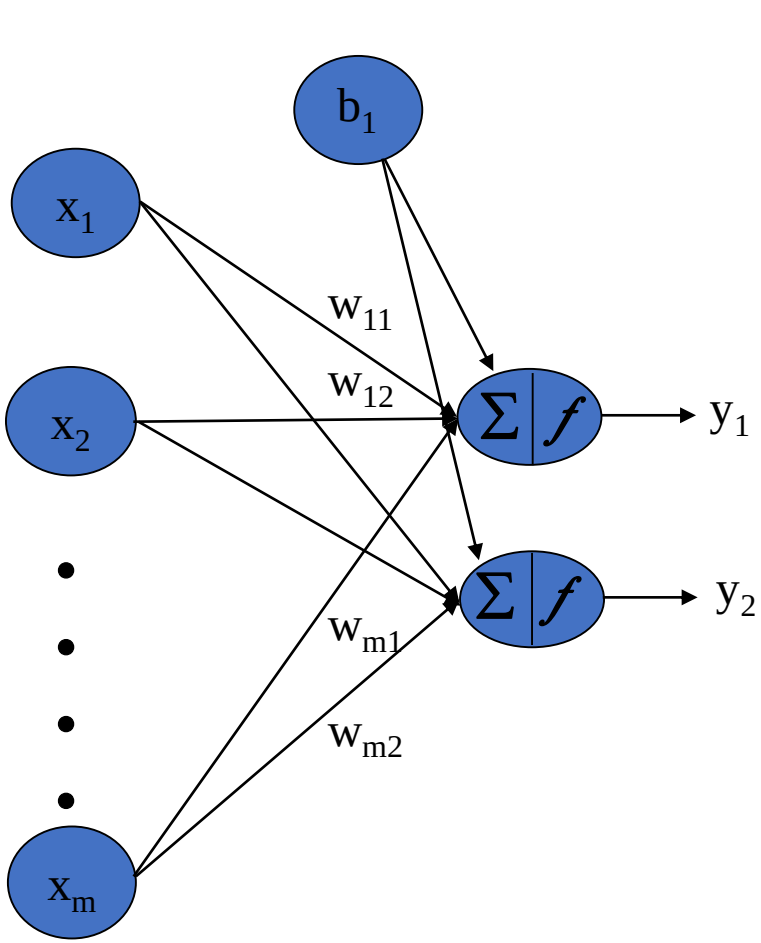
Copyright ACDSD, CSIR-NEIST

The Topics Covered In This Section

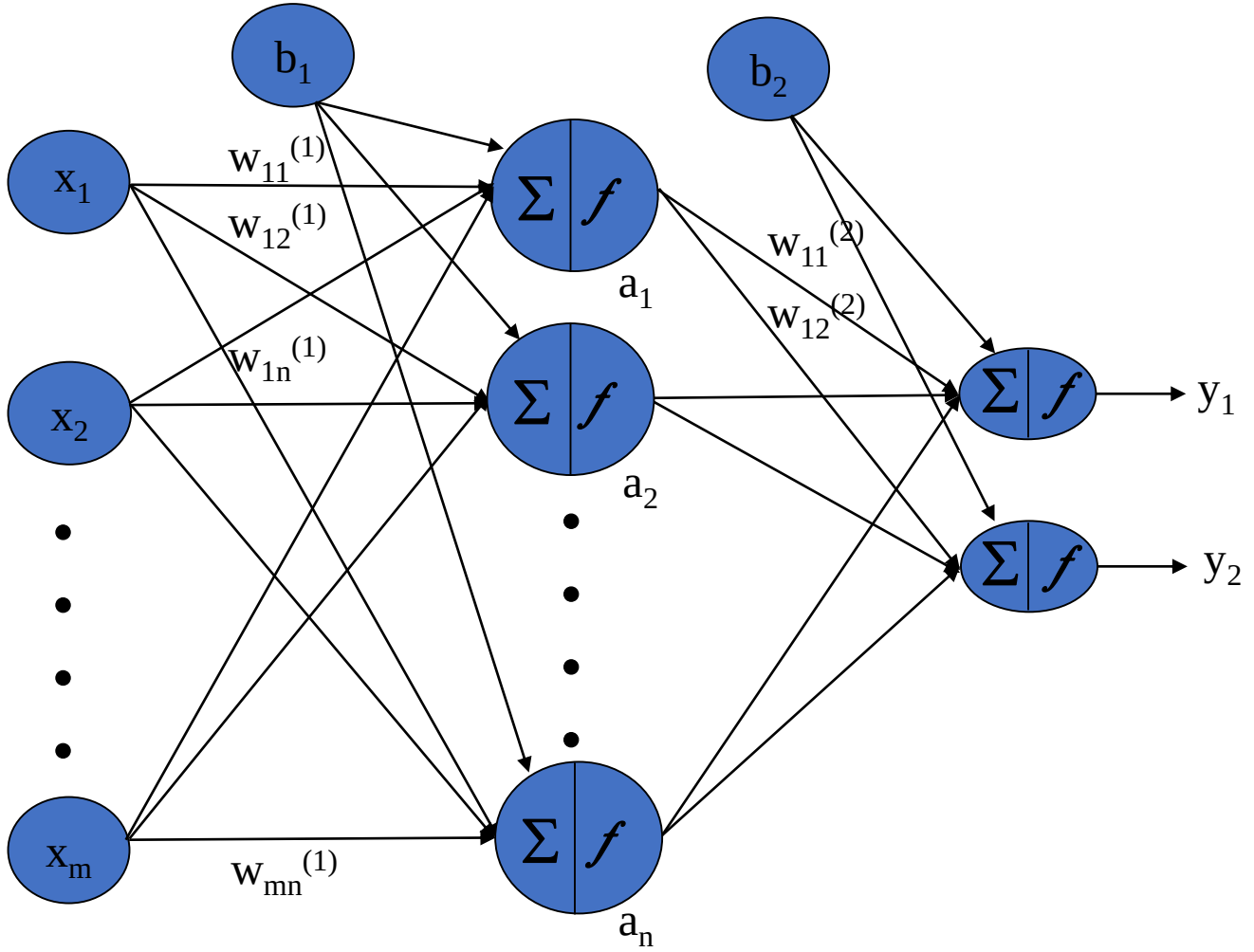
11.3.1 Architecture
11.3.2 Components
11.3.3 Outcome Behavior
11.3.4 Structural Learning
11.3.5 Parameter Learning
11.3.6 Back Propagation
11.3.7 Gradient Descent
11.3.8 Examples
11.3.9 Generalization
11.3.10 Learning vs. Generalization
11.3.11 Overfitting
11.3.12 Estimation of Generalization Error
11.3.13 Regularization Methods
11.3.14 Batch Normalization
11.3.15 Black Box
11.3.16 Summary

Copyright ACDSD, CSIR-NEIST

Feed Forward ANN



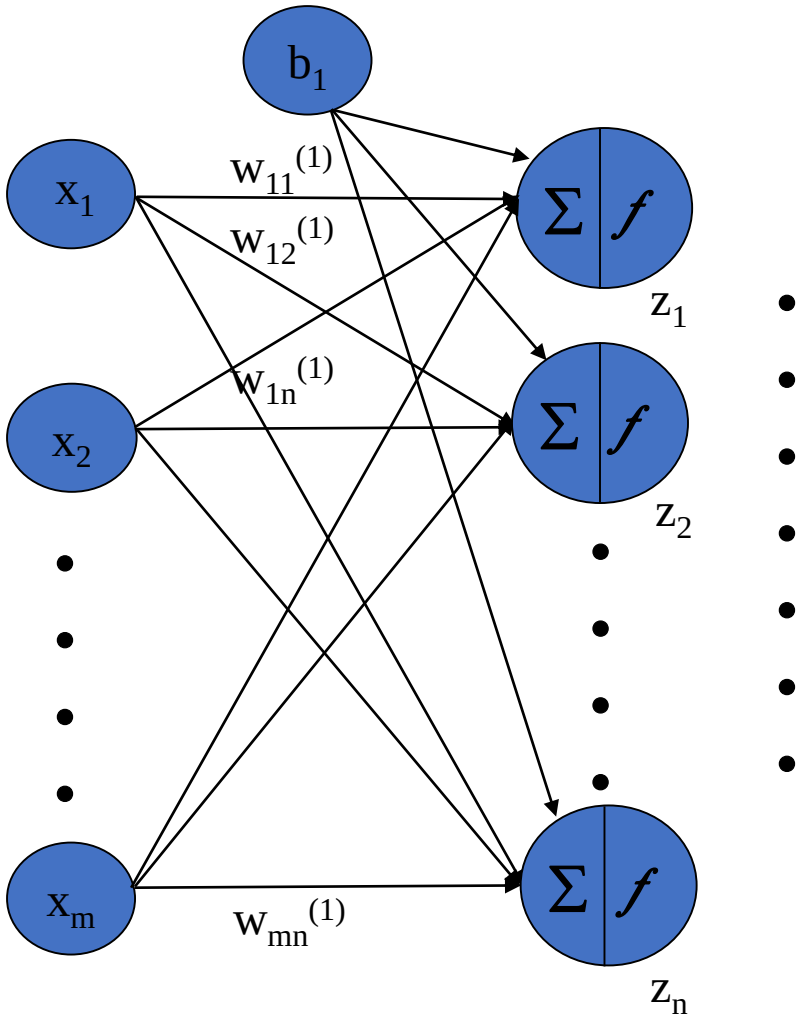
Input layer Output layer

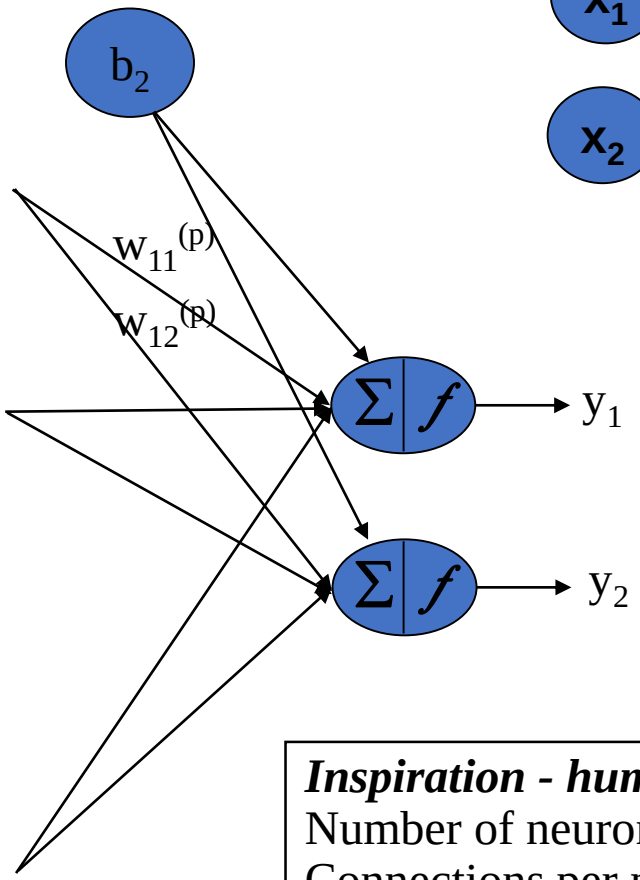


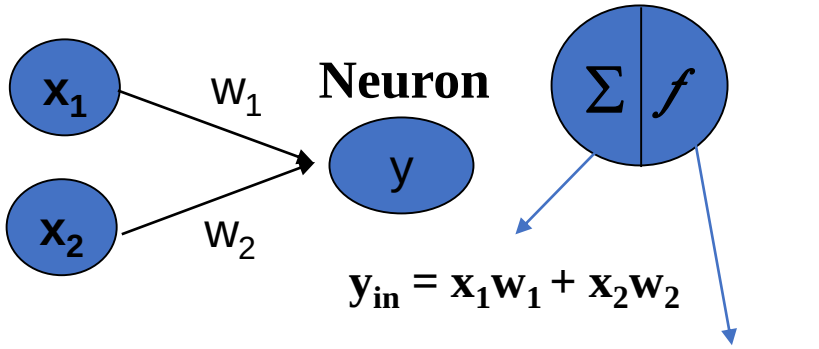
Input layer Hidden layer Output layer

Independent variables *Dependent variable (Prediction)*

Feed Forward ANN with Multiple Layers







Activation Function:

if $y_{in} >= \theta$ $f(y_{in}) = 1$

else $f(y_{in}) = 0$

Inspiration - humans

Number of neurons ~10^{10}

Connections per neuron ~10^{4-5}

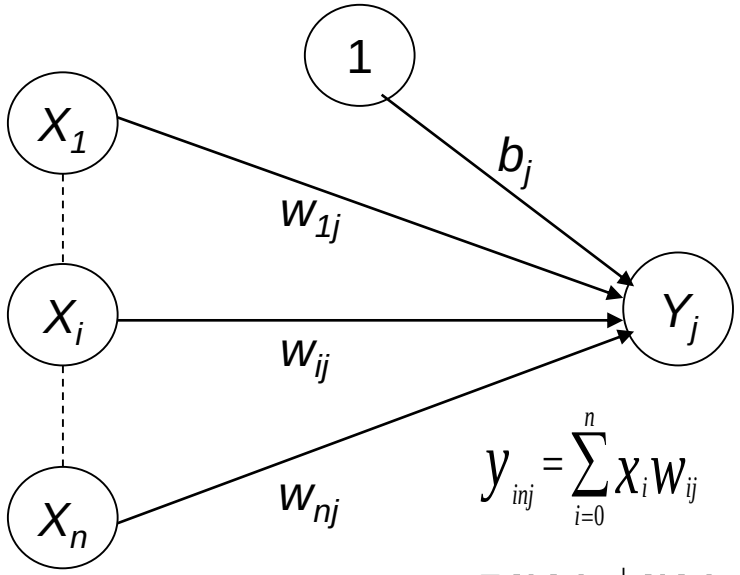
Input layer Hidden layer Output layer

Copyright ACDSD, CSIR-NEIST

Weights

- Each neuron connected to every other neuron by means of directed links associated with weights.
- Weights contain information about the input signal and is represented as a matrix.

Weight matrix W = $\begin{bmatrix} w_{11} w_{12} w_{13} \cdots w_{1m} \\ w_{21} w_{22} w_{23} \cdots w_{2m} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ w_{n1} w_{n2} w_{n3} \cdots w_{nm} \end{bmatrix}$



$$y_{inj} = \sum_{i=0}^{n} x_i w_{ij}$$

$$= x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + \ldots + x_n w_{nj}$$

$$= w_{0j} + \sum_{i=1}^{n} x_i w_{ij}$$

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

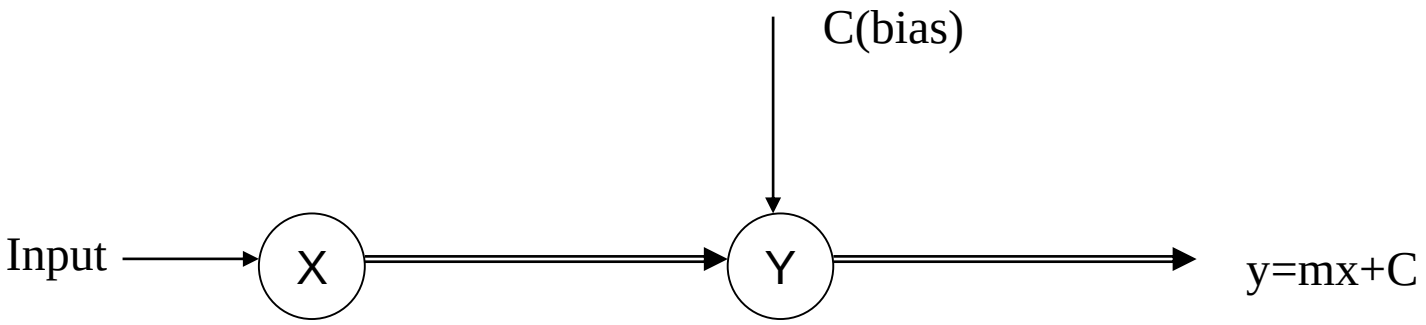
Network Weight Initialization

- Random initial values +/- some range.
- Randomly initialize weights w(i) generally as -0.5<=i<=0.5

Copyright ACDSD, CSIR-NEIST

Bias

- Bias is like another weight (just like an intercept added in a linear equation).
- Bias is of two types: Positive bias - increase the net input and Negative bias - decrease the net input
- Its included by adding a component $x_0=1$ to the input vector X. $X=(1,X_1,X_2...X_i,...X_n)$



Why Bias is required?

- Used to adjust the output along with the weighted sum of the inputs to the neuron.
- Allows to shift the activation function to either right or left.
- Adding a bias term solves the output result 0 problem when original inputs are zero.

Copyright ACDSD, CSIR-NEIST

Activation functions

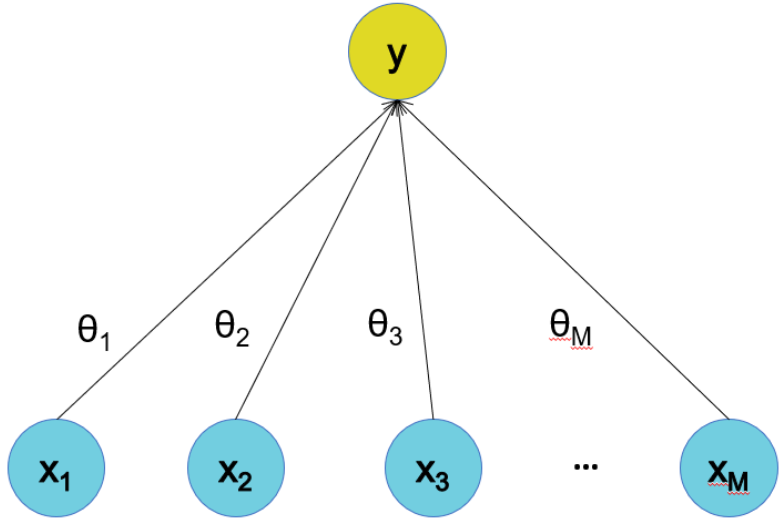
- Sum of the weighted input signal is applied with an activation to calculate the output of a neuron.
- It provides the non linearity to the summation output.
- A NN without AF is simply a linear regression model i.e. output signal is a simple **linear function**.
- So, would not be able to learn and model other complicated, high-dimensional, non-linear kinds of data such as images, videos, audio, speech etc.

Linear Regression $y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$

where $\sigma(a) = a$

Logistic Regression $y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



- Another important feature of a AF is that it should be differentiable to perform learning.

Copyright ACDSD, CSIR-NEIST

Layers and Types

- The input layer
  - Introduces input values the world outside into the network.
  - No activation function or other processing.
- The hidden layer(s)
  - Activation function or other processing.
  - Two hidden layers are sufficient to solve any problem.
- The output layer
  - Functionally just like the hidden layers.
  - Outputs are passed on to the world outside the neural network.

#of input nodes: Depends upon the #of features.

#of input nodes: Depends upon the no of classes.

- Single output for Regression
- Multiple Outputs used for N-way classification)

Copyright ACDSD, CSIR-NEIST

11.3.3 Output Behavior

Behavior of an ANN to any particular input depends upon

- Structure of the network (architecture) and structure of each node (activation function)
- Weights and bias parameters on each of the connections

.... these must be learned !

These leads to two kinds of learning

- Structure Learning – Change in network structure and hyperparameters
- Parameter learning – Train the network to learn the parameters

Copyright ACDSD, CSIR-NEIST

Determining Optimal Network Structure

a) How many hidden layers?

b) How many hidden units per layer?

c) How should the units be connected? (e.g., Fully, Partial, using domain knowledge)

Getting Optimal Solution

- Finding a good network structure is example of a search problems.
- Some approaches to search for a solution for this problem include ***Genetic algorithms.***
- But using GAs can be very cpu-intensive.

Copyright ACDSD, CSIR-NEIST

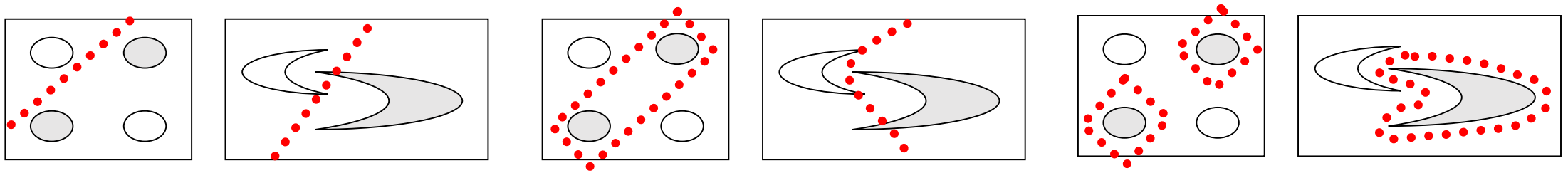
11.3.4 Structural Learning

Number of Hidden Layers

- If the function to learn is linear no need of hidden layer.
- If the function to learn is non-linear there is need of hidden layer.
- Apparently, three layers is almost always good enough.
- Also fewer layers are faster in execution and training

Significance of hidden layers

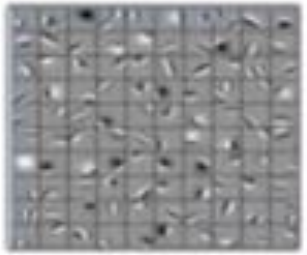
- 0 hidden layers: linear classifier - Hyperplanes
- 1 hidden layer: Boundary of convex region (open or closed)
- 2 hidden layers: Combinations of convex regions (arbitrary boundaries)

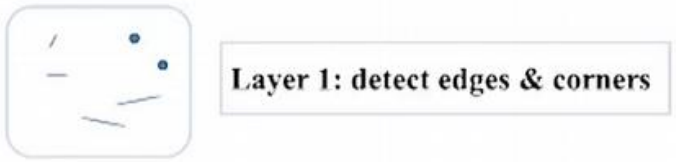


Copyright ACDSD, CSIR-NEIST

11.3.4 Structural Learning

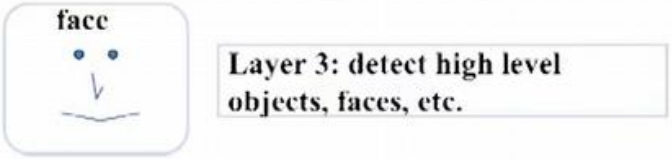
- **What are the hidden layers doing?**
  - **Feature Extraction**





Detect edges and corners

Group all together and come up with some meaningful feature groups



Collects all this and leads to higher level objects

Copyright ACDSD, CSIR-NEIST

11.3.4 Structural Learning

Number of Hidden Units

Too Low

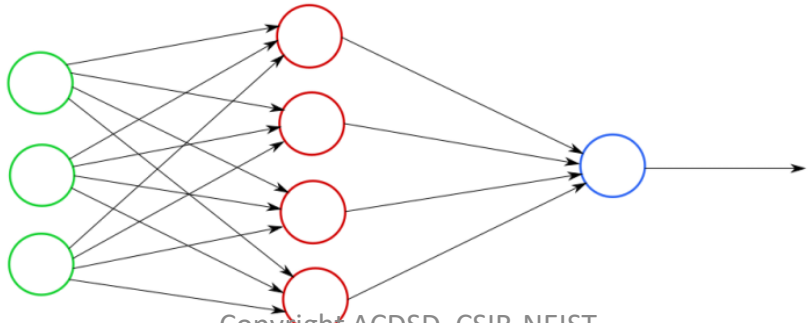
- The network may not be able to represent more complicated patterns.

Too High

- Given too many hidden units, a neural net will simply memorize the input patterns.
- Because of overtraining, its best to have less #of hidden layers and then increase the numbers.

Universal Approximation Theorem

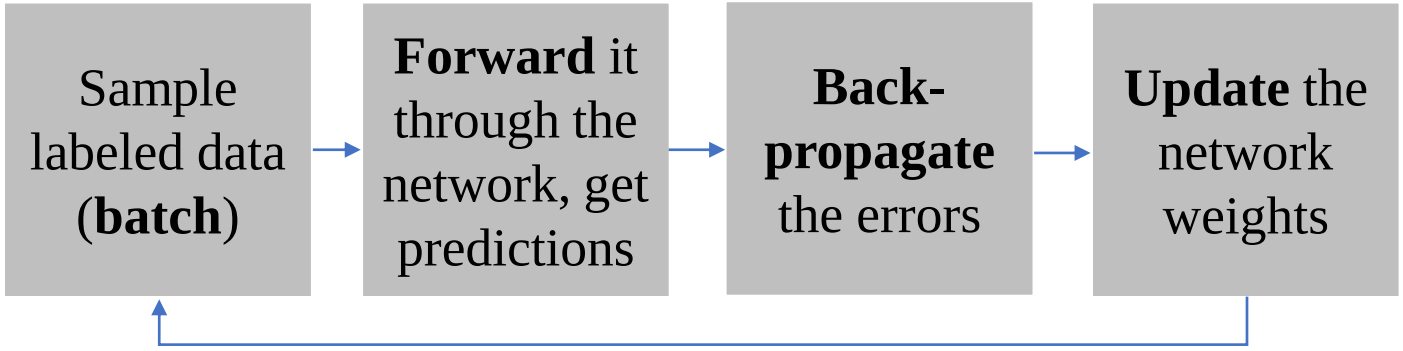
- Comes around 1989-1990
- Any function can be expressed as a neural network with one hidden layer to desired accuracy.

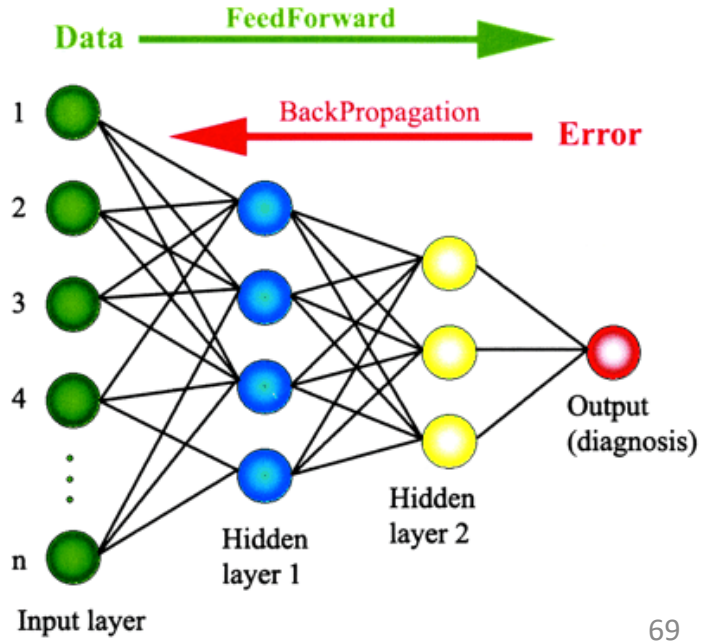


Copyright ACDSD, CSIR-NEIST

11.3.5 Parameter Learning

- Initially started with any random values for weight (generally smaller values)
- Apply ***forwardpropagation*** and find out calculated output
- Check the expected output and hence calculate the error ***(loss function)***
  - If no error then no needed to do anything - model is ready
  - If error then go to ***backpropagation algorithm***
    - Calculate the error of cost at the last level
    - From last layer update weights using ***gradient Descent*** up to second layer
    - Recursively doing that until error of cost minimized to 0





Copyright ACDSD, CSIR-NEIST

11.3.5 Learning Properties

Pros

- If a problem can be solved with a separation hyperplane, then the set of weights is found in finite iterations and solves the problem correctly.
- Able to learn any arbitrary function.
- Works well with noisy data.

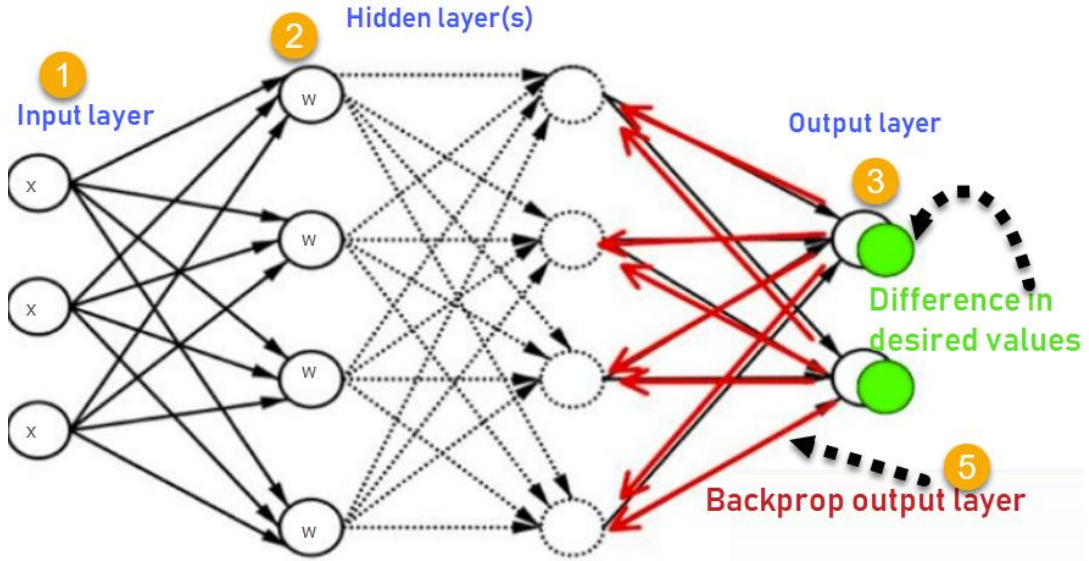
Cons

- Best suited for supervised learning
- It's a black box! (*can't see how it's making decisions?)*
- Training can take thousands of iterations (epocs) which is slow! $O(w^3)$
- Could be slow to converge (Note: Training a three node net is NP-Complete!).
- May converges (Gradient descent) to some local error minimum, Perhaps not global minimum.
- Can get different results (using different initial weights).

Copyright ACDSD, CSIR-NEIST

11.3.6 Backpropagation

- Discovered and rediscovered throughout 1960-1970.
- Werbos (1982) first used it in ANN to train the network.
- Popularized through the work of Rumelhart et. al.[1986].



- A gradient descent search through the parameter space to minimize the sum-of-squares error.
- A case of supervised learning to train FFNN using a nonlinear, differentiable activation function.
  - To optimize the weights so that network learn how to correctly map arbitrary inputs to outputs.
  - In other words, to adjust internal links (how to distribute the “blame” or the error).

Copyright ACDSD, CSIR-NEIST

11.3.6 Backpropagation

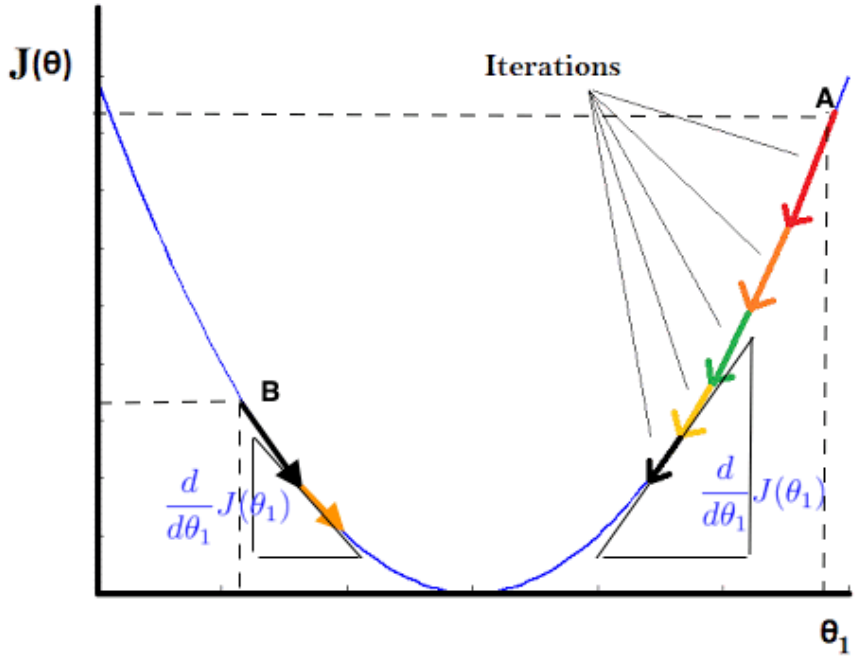
function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
  inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
    network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      /* *Propagate the inputs forward to compute the outputs* */
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\ a_i$
          $a_j \leftarrow g(in_j)$
      /* *Propagate deltas backward from output layer to input layer* */
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\ \Delta[j]$
      /* *Update every weight in network using deltas* */
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

Copyright ACDSD, CSIR-NEIST

11.3.7 Gradient Descent

- Discovered in 1847 by Cauchy to compute the orbit of heavenly bodies.
- Later in 1970-80 the back propagation algorithm is used in conjunction with gradient descent.

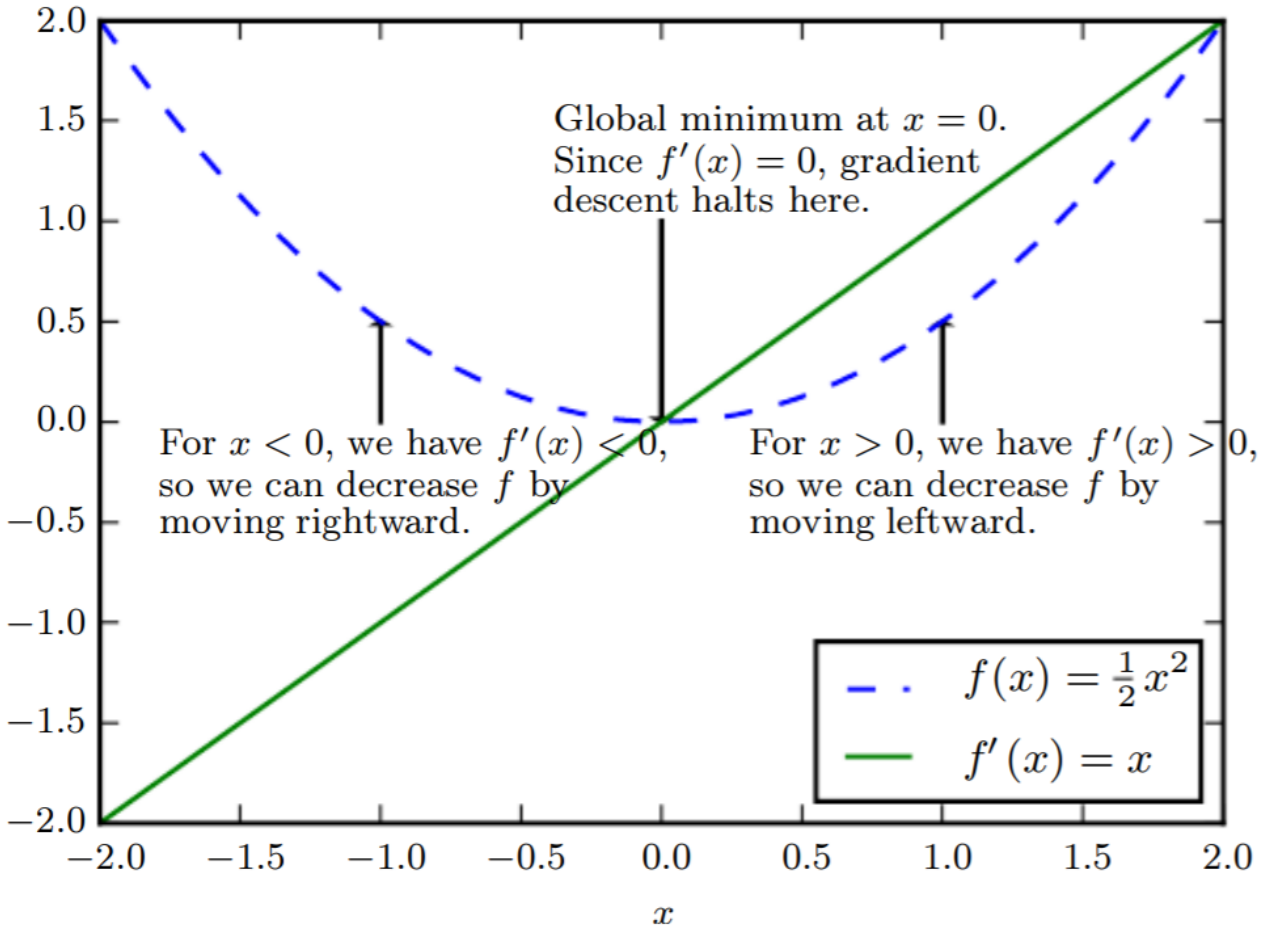


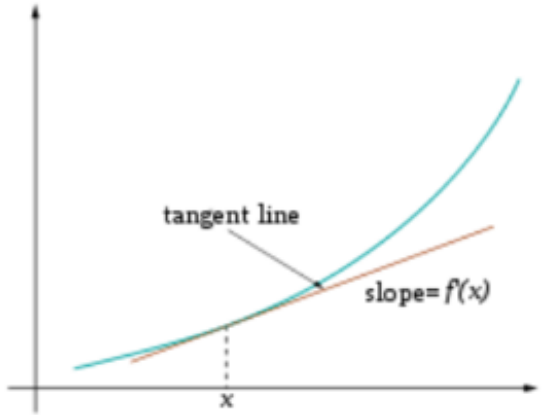
- It is an optimization algorithm for finding the minimum of a function f(x) by altering x.
- To find a local minimum, we take steps proportional to the negative of the gradient.
- Subtracting a fraction of the gradient moves us towards the (local) minimum of the cost function

Copyright ACDSD, CSIR-NEIST

11.3.7 Gradient Descent

- We usually phrase most optimization problems in terms of minimizing f (x).
- Maximization may be accomplished via a minimization algorithm by minimizing −f(x).

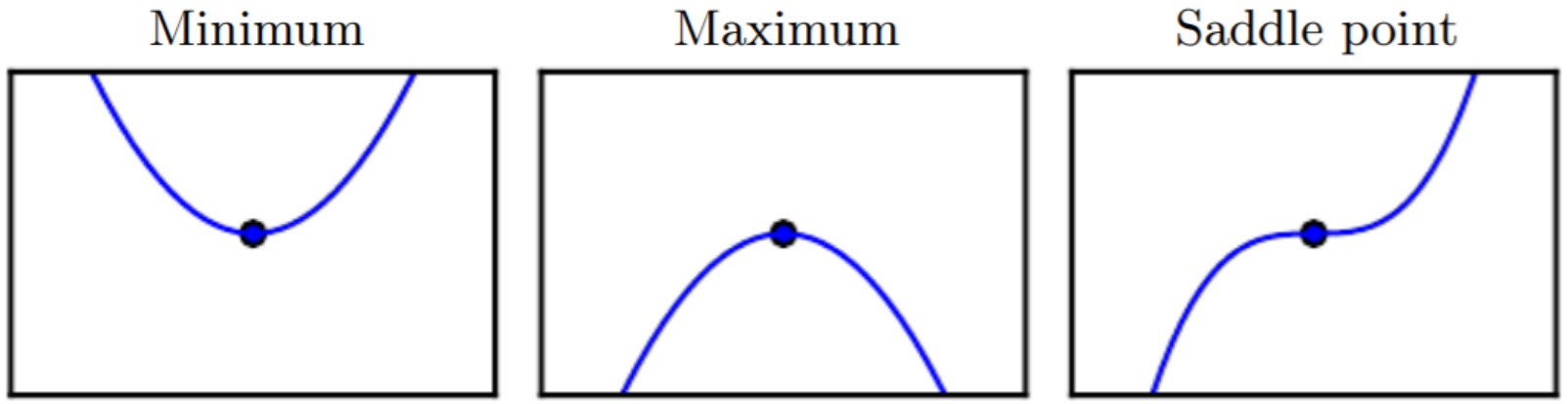




Copyright ACDSD, CSIR-NEIST

Concept of Critical Points (A point with zero slope)

- Local minimum point - lower than the neighboring points
- Local maximum point - higher than the neighboring points
- Saddle point - has neighbors that are both higher and lower than the point itself.



Copyright ACDSD, CSIR-NEIST

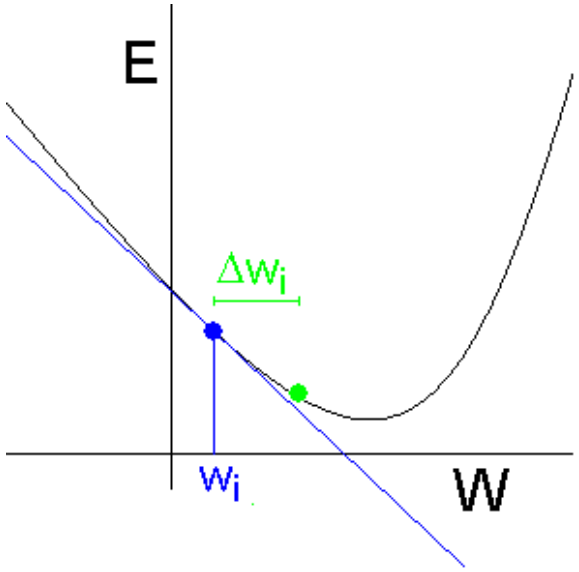
11.3.7 Gradient Descent

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

Learning rate Gradient

$$\frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

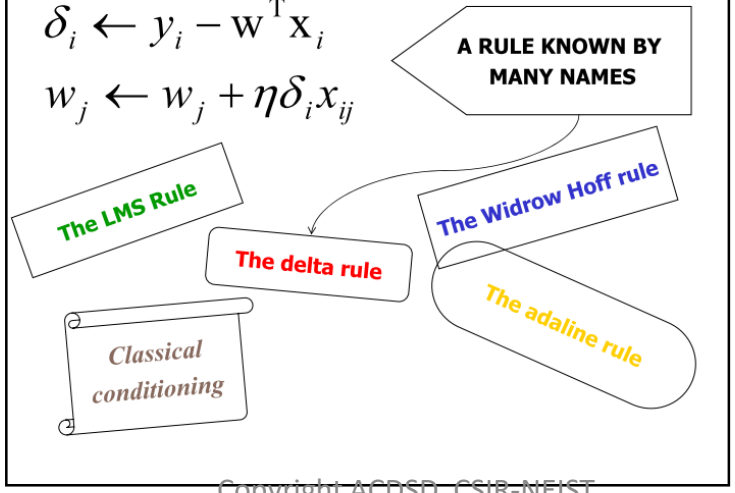


$$w_j \leftarrow w_j + 2\eta \sum_{k=1}^{R} \delta_k x_{kj}$$

- Require a convex loss function for gradient descent training.
- Gradient is given by: rate at which error changes as weight change

$$\delta_i \leftarrow y_i - \mathrm{w}^{\mathrm{T}} \mathrm{x}_i$$

$$w_j \leftarrow w_j + \eta \delta_i x_{ij}$$



$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{R} \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$= \sum_{k=1}^{R} 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k$$

...where...

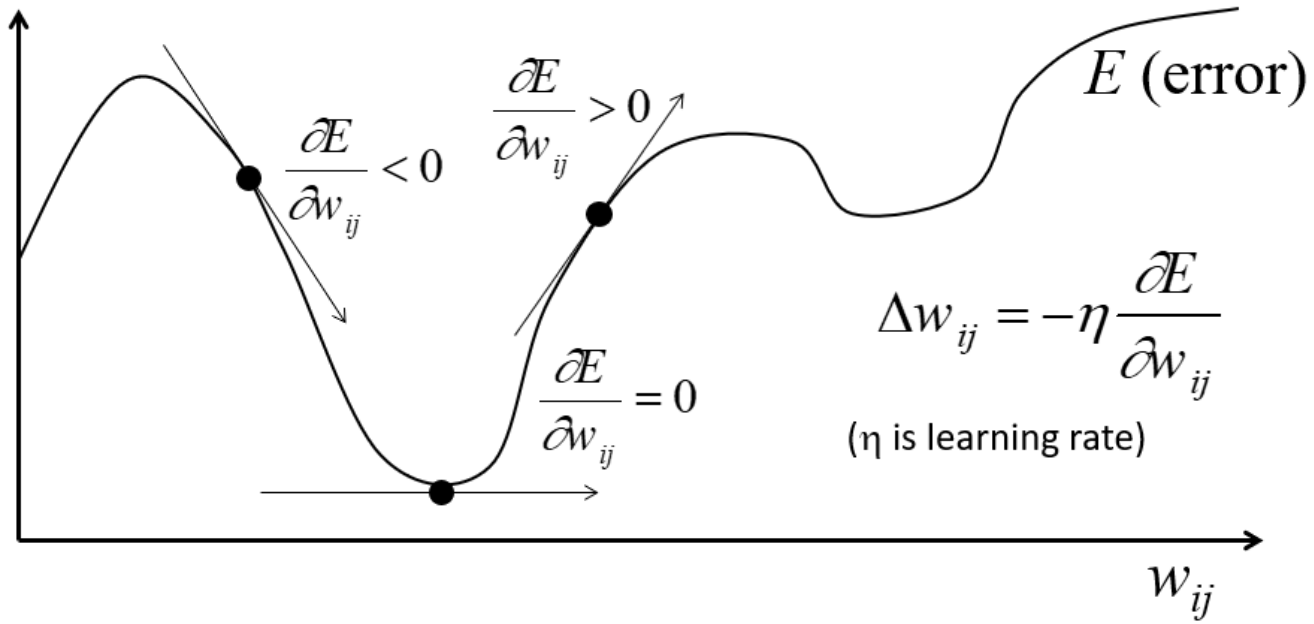
$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

Copyright ACDSD, CSIR-NEIST

11.3.7 Gradient Descent



Gradient $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$

Training rule: $\Delta w_i = -\eta \nabla E[\vec{w}]$ $\quad w_i \leftarrow w_i + \Delta w_i$

i.e., $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Copyright ACDSD, CSIR-NEIST

11.3.7 Gradient Descent

Batch mode Gradient Descent: [Full Batch or Mini Batch]

- Do until satisfied:

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Incremental mode (stochastic) Gradient Descent: [Online Gradient Descent]

- Do until satisfied:
- - For each training example d in D

1. Compute the gradient $\nabla E_d[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Note: On-line is faster than batch where as batch more efficient.

Copyright ACDSD, CSIR-NEIST

Gradient Descent for Sigmoidal unit:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d}(t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right)$$

$$= -\sum_{d}(t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}$$

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1-o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w}\cdot\vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

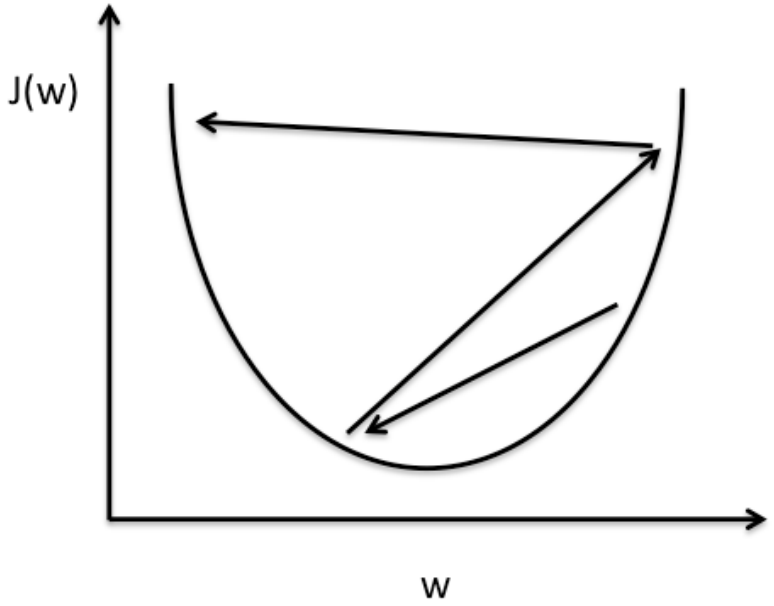
$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{i,d}$$

Copyright ACDSD, CSIR-NEIST

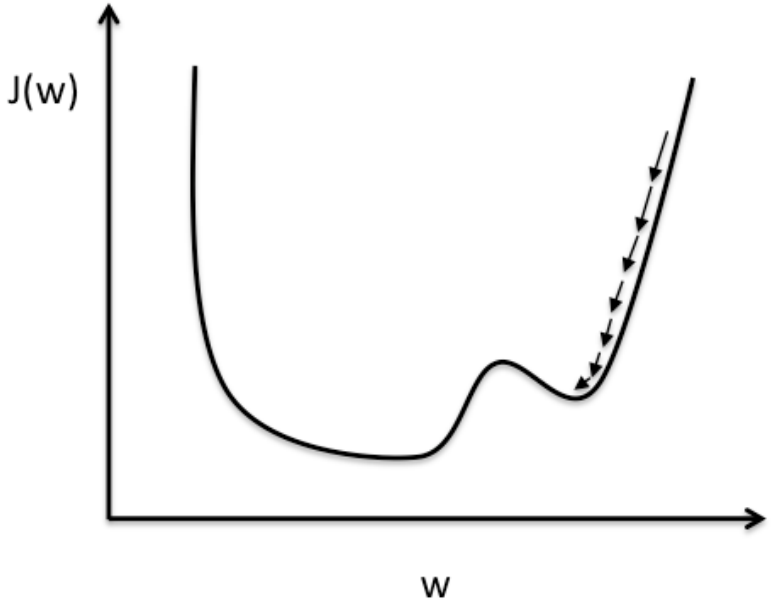
11.3.7 Gradient Descent

Learning Rate

- Used to control the amount of weight adjustment at each step of training.
- Learning rate ranging from 0 to 1 *(typically set = 0.1)* determines the rate of learning in each step.



Large Learning Rate (η)
Overshooting

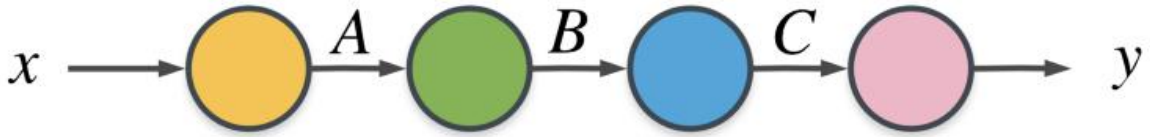


Small Learning Rate (η)
Many iterations until convergence and trapping in local minima

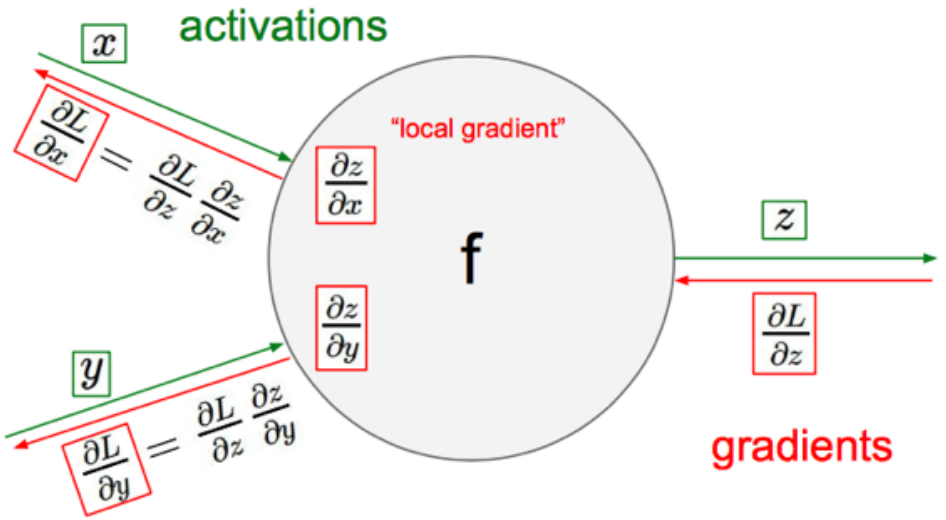
Copyright ACDSD, CSIR-NEIST

Chain Rule

- Backpropagation is just repeated application of chain rule applied recursively through each node.



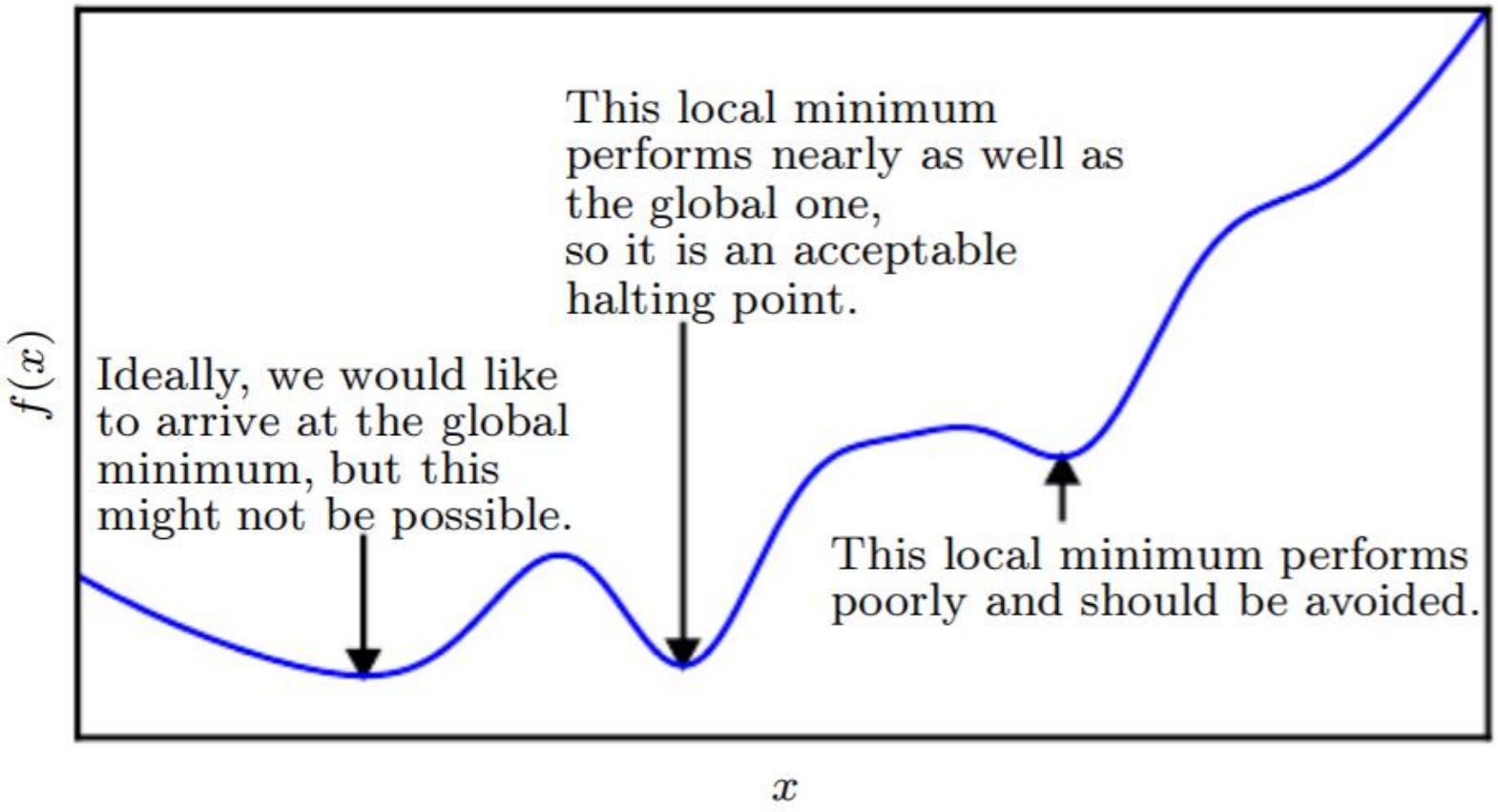
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial C} \times \frac{\partial C}{\partial B} \times \frac{\partial B}{\partial A} \times \frac{\partial A}{\partial x}$$



- Start by computing the delta term for the output layer and then we go back a layer and compute the delta terms for the before hidden layer and so on.
- So we're sort of back propagating the errors from the output layer to next to input layer.

Copyright ACDSD, CSIR-NEIST

Concept of Local and Global Minimum



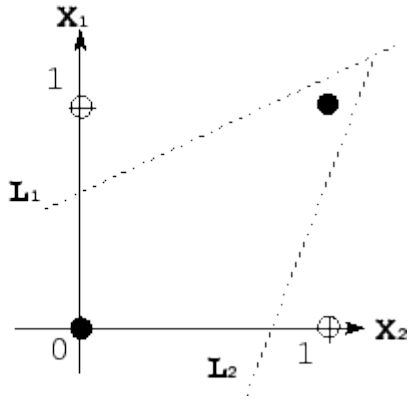
Copyright ACDSD, CSIR-NEIST

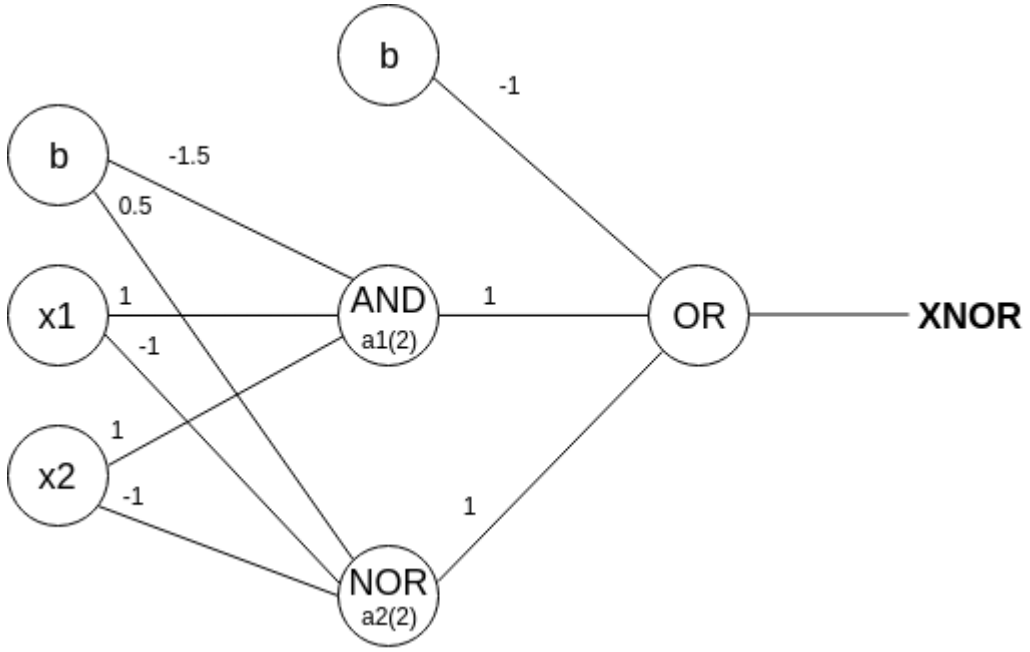
Boolean XOR

- Single layer NN can only represent linear decision boundary.
- XOR can not be implemented by linear decision boundary.
- NN with multiple layers of units used to compute complex functions like XOR/XNOR function.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$



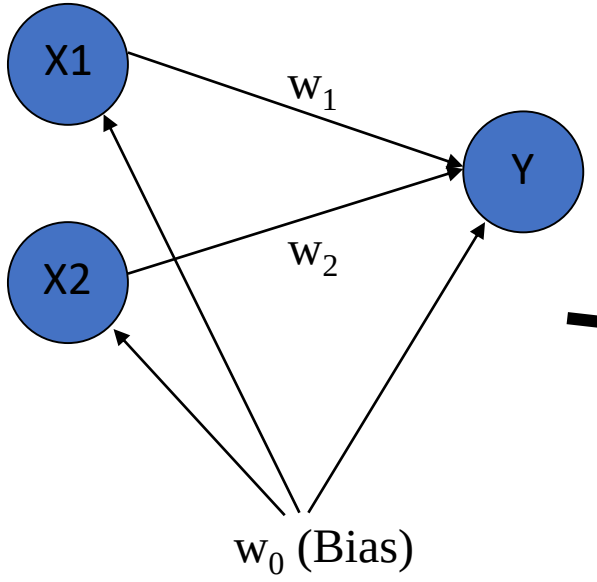


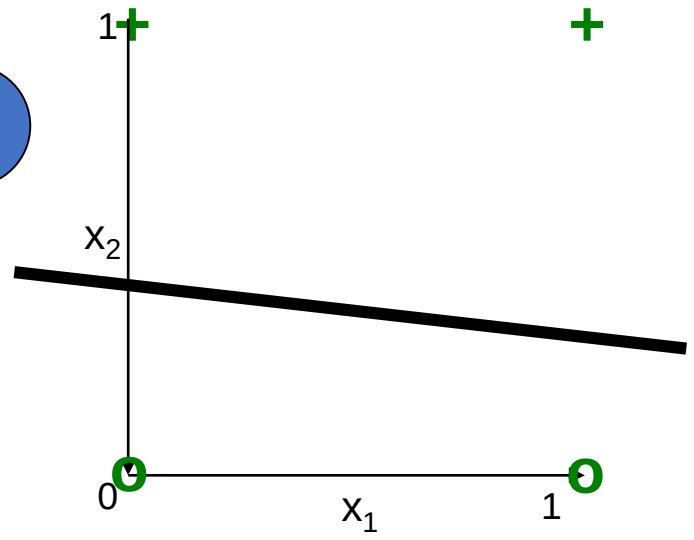
Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

- Need to design NN considering following condition with given input and desired output –

Input		Output
X1	**X2**	**Y**
0	0	0
0	1	1
1	0	0
1	1	1





Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

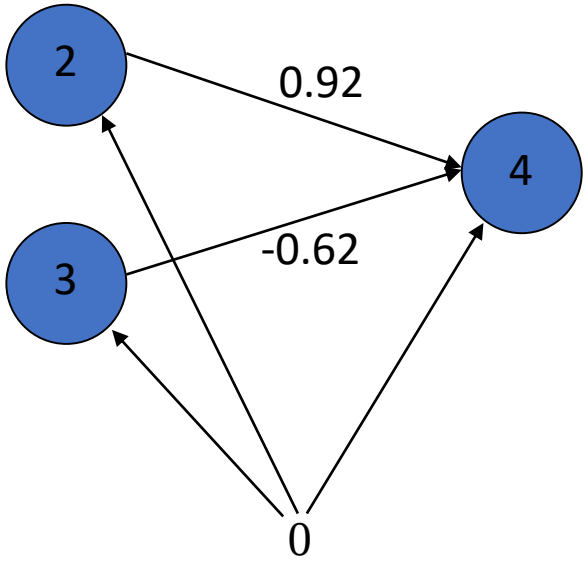
- Initialization:

$W_1(0) = 0.92$

$W_2(0) = 0.62$

$W_0(0) = 0.2236$

$\alpha = 0.1$



- Let us see the parameters are updated in each epoch.

Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

Training – epoch 1:

$\text{out1} = \text{sign}(0.92*0 + 0.62*0 - 0.22) = \text{sign}(-0.22) = 0$

$\text{out2} = \text{sign}(0.92*1 + 0.62*0 - 0.22) = \text{sign}(0.7) = 1$ (Incorrect)

$\text{out3} = \text{sign}(0.82*0 + 0.62*1 - 0.32) = \text{sign}(0.5) = 1$

$\text{out4} = \text{sign}(0.82*1 + 0.62*1 - 0.32) = 1$

$W_1(1) = 0.92 + 0.1 * (0 - 1) * 1 = 0.82$
$W_2(1) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
$W_0(1) = 0.22 + 0.1 * (0 - 1) * (-1) = 0.32$

Training – epoch 2:

$\text{out1} = \text{sign}(0.82*0 + 0.62*0 - 0.32) = \text{sign}(0.32) = 0$

$\text{out2} = \text{sign}(0.82*1 + 0.62*0 - 0.32) = \text{sign}(0.5) = 1$ (Incorrect)

$\text{out3} = \text{sign}(0.72*0 + 0.62*1 - 0.42) = \text{sign}(0.3) = 1$

$\text{out4} = \text{sign}(0.72*1 + 0.62*1 - 0.42) = 1$

$W_1(2) = 0.82 + 0.1 * (0 - 1) * 1 = 0.72$
$W_2(2) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
$W_0(2) = 0.32 + 0.1 * (0 - 1) * (-1) = 0.42$

Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

Training – epoch 3:

$\text{out1} = \text{sign}(0.72*0 + 0.62*0 - 0.42) = 0$

$\text{out2} = \text{sign}(0.72*1 + 0.62*0 - 0.42) = 1$ (Incorrect)

$W_1(3) = 0.72 + 0.1 * (0 - 1) * 1 = 0.62$
$W_2(3) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
$W_0(3) = 0.42 + 0.1 * (0 - 1) * (-1) = 0.52$

$\text{out3} = \text{sign}(0.62*0 + 0.62*1 - 0.52) = 1$

$\text{out4} = \text{sign}(0.62*1 + 0.62*1 - 0.52) = 1$

Training – epoch 4:

$\text{out1} = \text{sign}(0.62*0 + 0.62*0 - 0.52) = 0$

$\text{out2} = \text{sign}(0.62*1 + 0.62*0 - 0.52) = 1$(Incorrect)

$W_1(4) = 0.62 + 0.1 * (0 - 1) * 1 = 0.52$
$W_2(4) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
$W_0(4) = 0.52 + 0.1 * (0 - 1) * (-1) = 0.62$

$\text{out3} = \text{sign}(0.52*0 + 0.62*1 - 0.62) = 0$ (Incorrect)

$W_1(4) = 0.52 + 0.1 * (1 - 0) * 0 = 0.52$
$W_2(4) = 0.62 + 0.1 * (1 - 0) * 1 = 0.72$
$W_0(4) = 0.62 + 0.1 * (1 - 0) * (-1) = 0.52$

$\text{out4} = \text{sign}(0.52*1 + 0.72*1 - 0.52) = 1$

Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

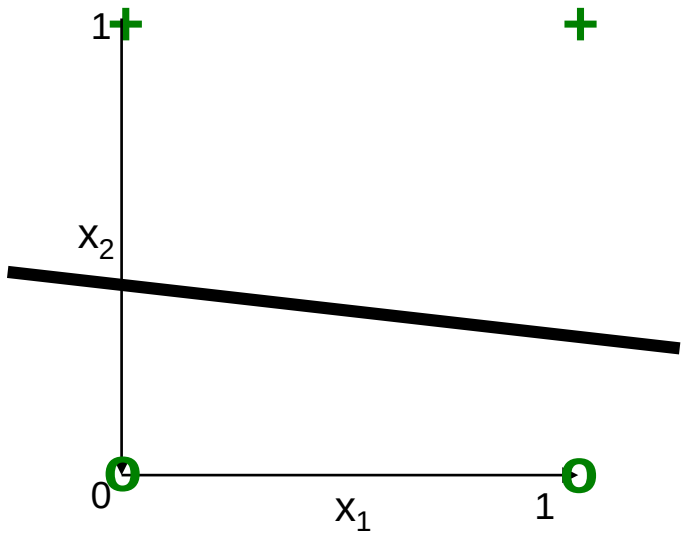
Finally:

$$\text{out1} = \text{sign}(0.12*0 + 0.82*0 - 0.42) = 0$$

$$\text{out2} = \text{sign}(0.12*1 + 0.82*0 - 0.42) = 0$$

$$\text{out3} = \text{sign}(0.12*0 + 0.82*1 - 0.42) = 1$$

$$\text{out4} = \text{sign}(0.12*1 + 0.82*1 - 0.42) = 1$$

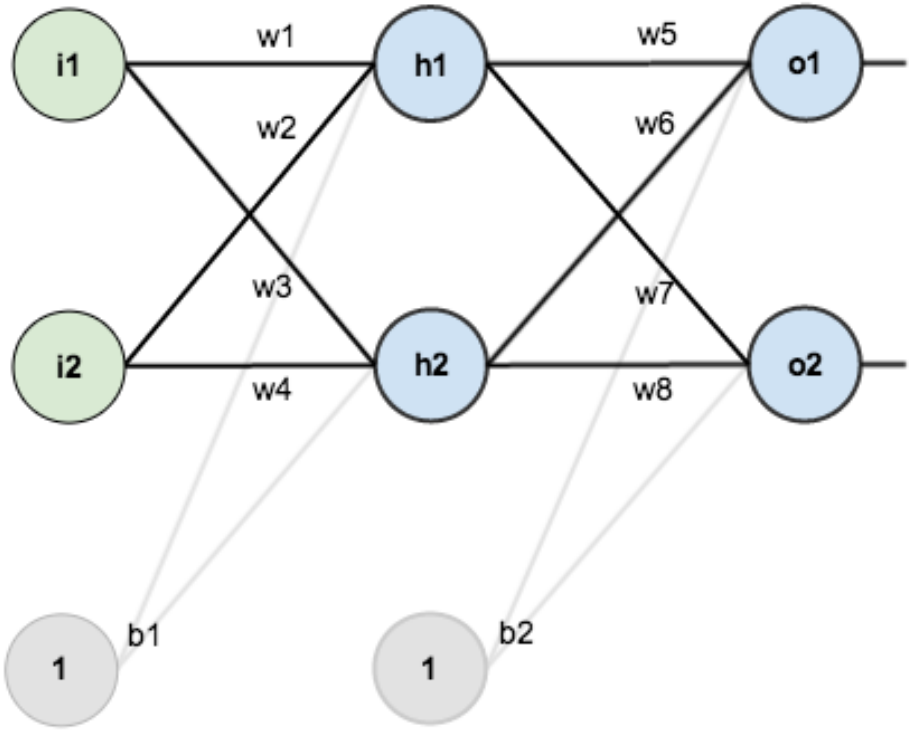


Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

Example *(https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/)*

- Given inputs 0.05 and 0.10
- Expected outputs 0.01 and 0.99



Copyright ACDSD, CSIR-NEIST

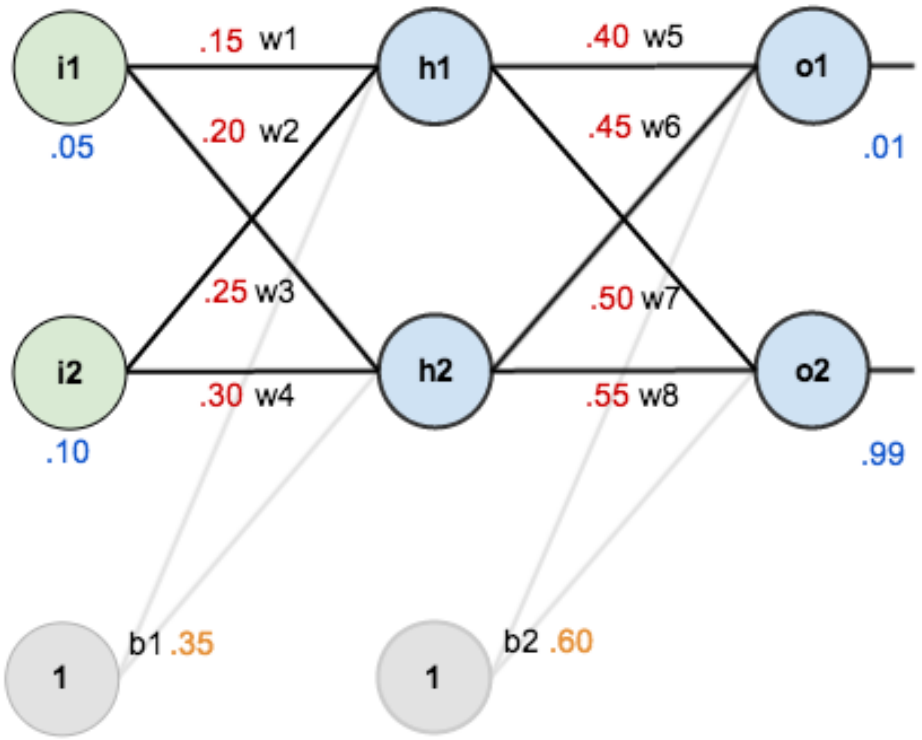
Forward Propagation

- The initial weights, the biases are as shown.

  w1=.15 w2=.20 w3=.25 w4=.30

  w5=.40 w6=.45 w7=.50 w8=.55

- Activation Function: Sigmoid



$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h_1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}} = \frac{1}{1 + e^{-0.3775}} = 0.5933$$

$$out_{h_2} = 0.5969$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$

$$net_{o_1} = 0.4 * 0.5933 + 0.45 * 0.5969 + 0.6 * 1 = 1.1059$$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}} = \frac{1}{1 + e^{1.1059}} = 0.7514$$

$$out_{o_2} = 0.7729$$

Copyright ACDSD, CSIR-NEIST

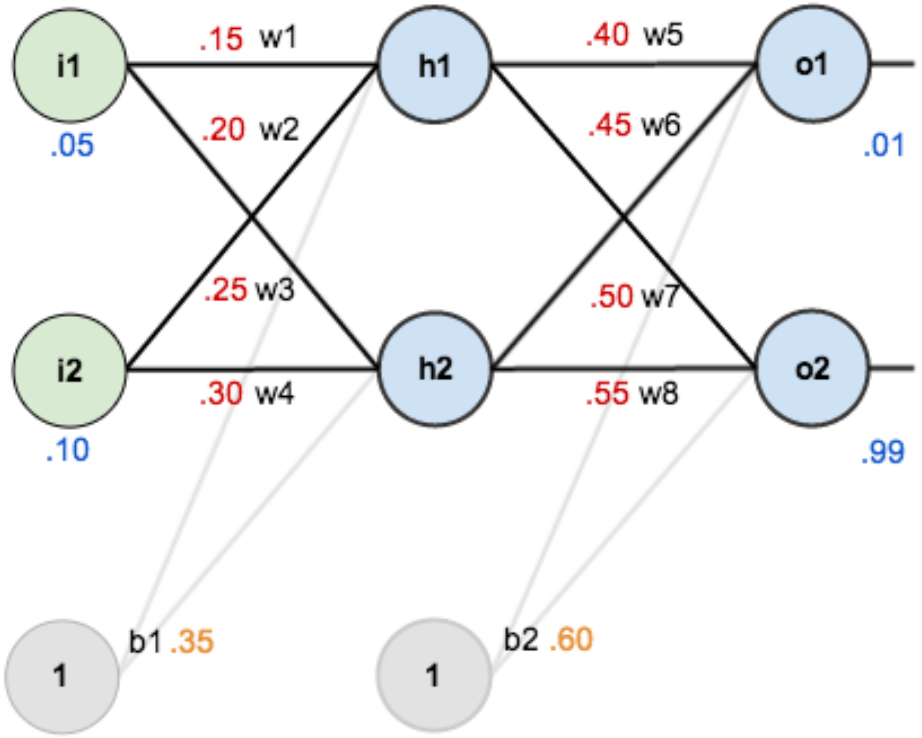
Calculating total error:

$$E_{total} = \frac{1}{2}\sum_{i=1}^{n}(target - output)^2$$

$$E_{total_{o_1}} = \frac{1}{2}\sum_{i=1}^{n}(\text{target}_{o_1} - \text{output}_{o_1})^2$$

$$= \frac{1}{2}(0.01 - 0.7514)^2 = 0.2748$$

$E_{total_{o_2}}$=0.0236



E_{total}=$E_{total_{o_1}}$ + $E_{total_{o_2}}$=0.2748+0.0236=0.2984

Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

The backward pass: *Consider* w_5

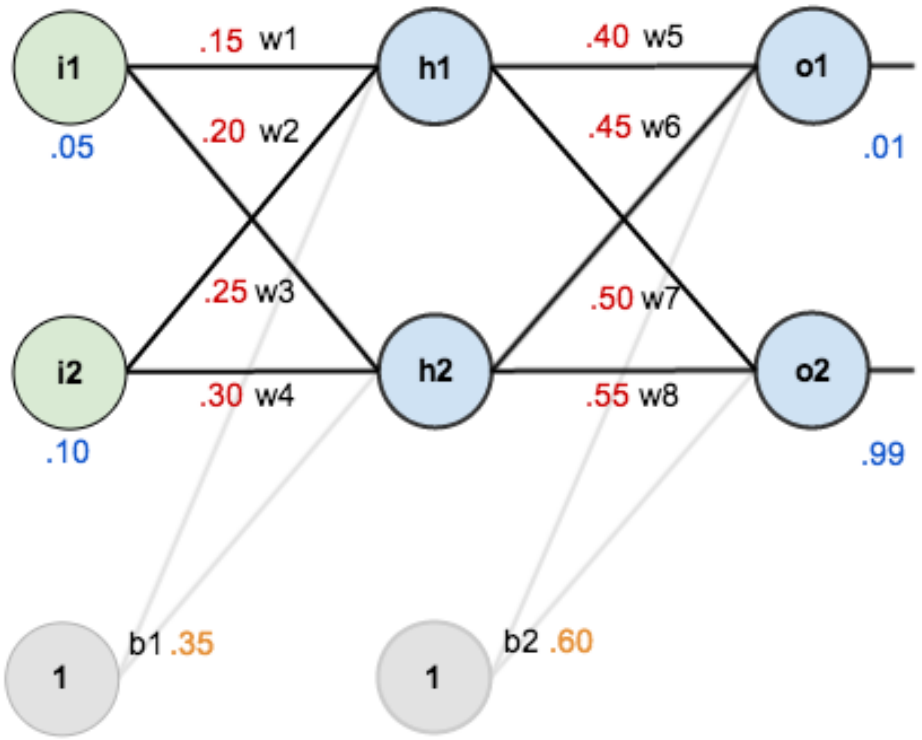
- How much a change in w_5 effects total error $\frac{\partial E_{total}}{\partial w_5}$
- That means the gradient w.r.t. w_5

- Applying chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

That means

- How much the total error E_{total} change w.r.t. output o_1
- How much the output of o_1 change w.r.t. total net input
- How much the total net input of o_1 change w.r.t. w_5



Copyright ACDSD, CSIR-NEIST

The backward pass: *Consider* w_5

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(\text{target}_{o_1} - \text{output}_{o_1})^2 + \frac{1}{2}(\text{target}_{o_2} - \text{output}_{o_2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o_1}} = 2 * \frac{1}{2}(\text{target}_{o_1} - \text{output}_{o_1})^{2-1} * (-1) + 0$$

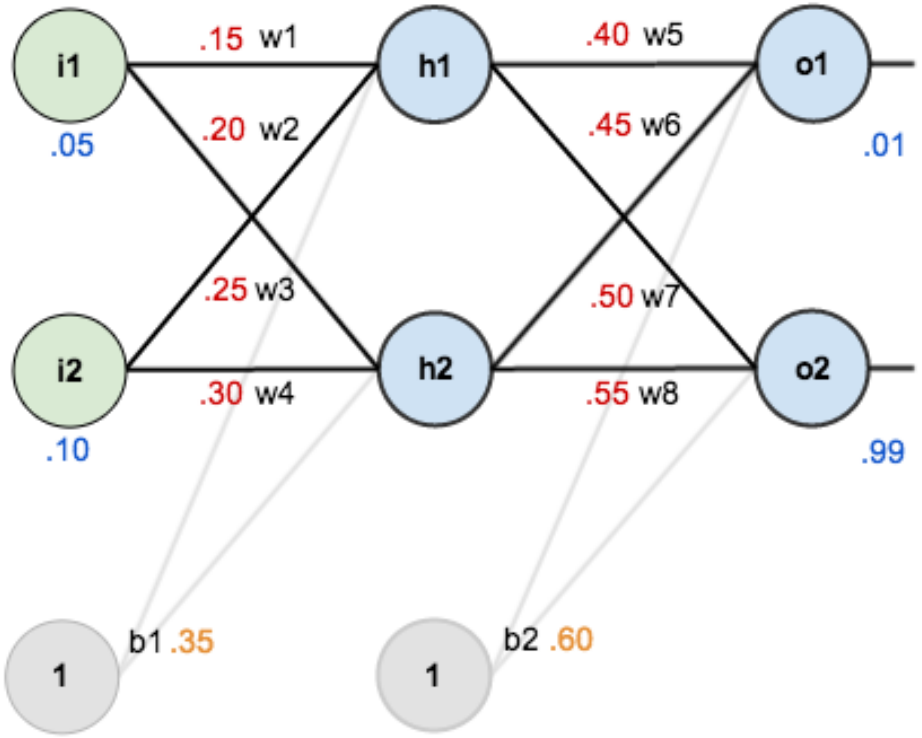
$$\frac{\partial E_{total}}{\partial out_{o_1}} = -(\text{target}_{o_1} - \text{output}_{o_1}) = -(0.01 - 0.7514) = 0.7414$$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1 - out_{o_1}) = 0.7514(1 - 0.7514) = 0.1868$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial w_5} = 1 * out_{h_1} * w_5^{1-1} + 0 + 0 = out_{h_1} = 0.5933$$



Putting it all together

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.7414 * 0.1868 * 0.5933 = 0.0822$$

Copyright ACDSD, CSIR-NEIST

Updating the weight:

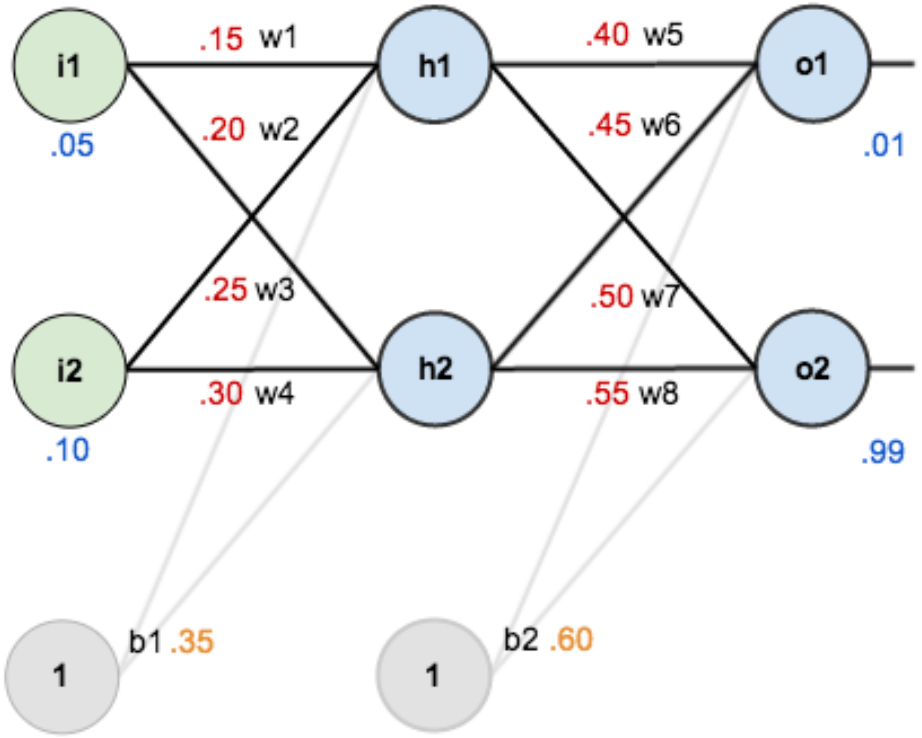
$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.0822 = 0.3589$$

- *Similarly updating the weights* w_6, w_7, w_8

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$



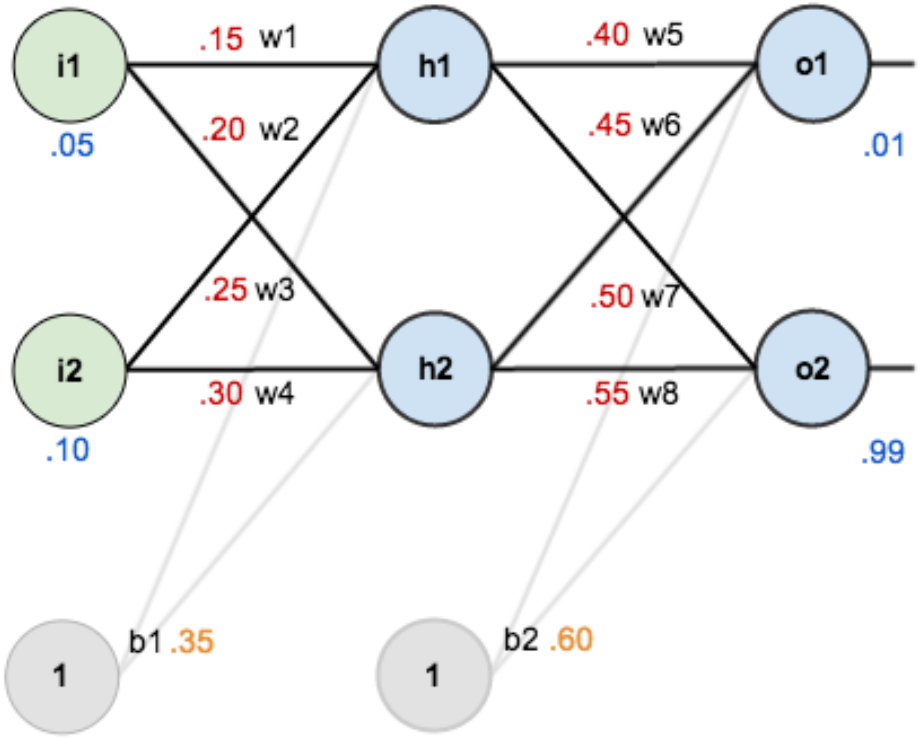
Copyright ACDSD, CSIR-NEIST

Hidden Layers: (calculating backpropagation for new values of updating the weights w_1, w_2, w_3, w_4)

Consider w_1

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



- The output of each hidden layer neuron contributes to the output (therefore error) of multiple output neurons.
- out_{h1} affects both out_{o1} *and* out_{o2} so needs to take into consideration effect of on the both output neurons.

Copyright ACDSD, CSIR-NEIST

Hidden Layers: (calculating backpropagation for new values of updating the weights w_1, w_2, w_3, w_4)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}} = 0.1385 * 0.40 = 0.0554$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = -0.0190$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}} = 0.0554 - 0.0190 = 0.0364$$

$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}}$$

$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = out_{h_1}(1 - out_{h_1}) = 0.5933(1 - 0.5933) = 0.2413$$

$$net_{h_1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h_1}}{\partial w_1} = i_1$$

$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{h_1}} = 0.7414 * 0.1868 = 0.1385$$

$$net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial out_{h_1}} = w_5 = 0.40$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}} = 0.1385 * 0.40 = 0.0554$$

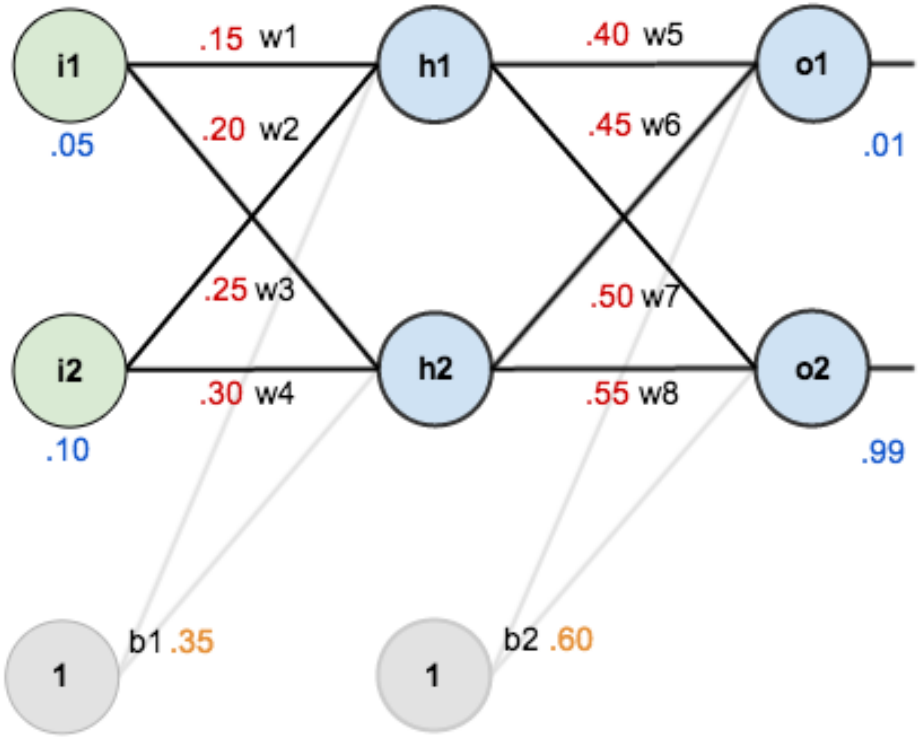
Putting it all together

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1} = 0.0364 * 0.2413 * 0.05 = 0.00044$$

Copyright ACDSD, CSIR-NEIST

Updating the weight:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.00044 = 0.1498$$



- *Similarly updating the weights w_2, w_3, w_4*

$w_2^+ = 0.1996$
$w_3^+ = 0.2498$
$w_4^+ = 0.2995$

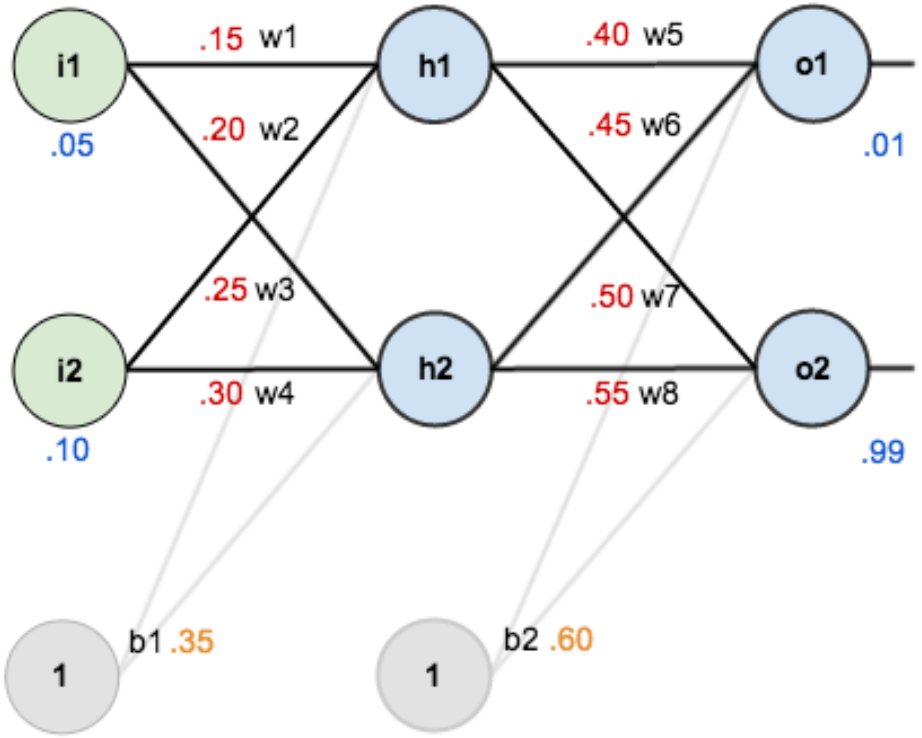
- Finally we have updated all the weights.
- Calculating new output values.
- New error is 0.2984

Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

Iterative actions:

- Earlier error was 0.2984
- New error (after first round) is 0.2910
- Not much improvement.

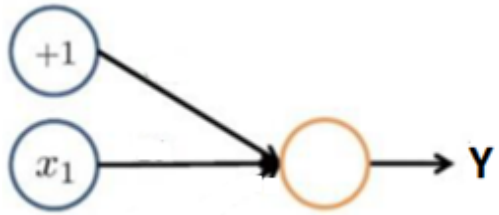


- But, repeating this process like 10,000 times.
- The error reduces to very low value like 0.000000035
- Accordingly new outs will be changed.

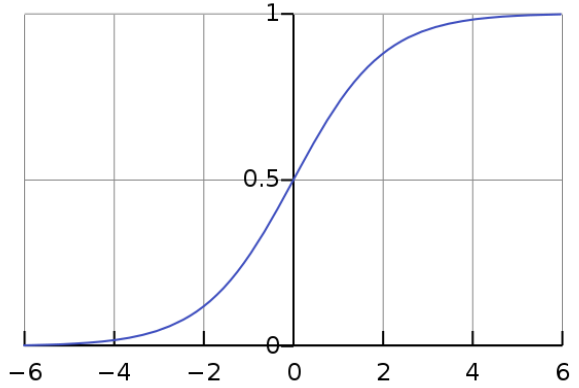
Copyright ACDSD, CSIR-NEIST

11.3.8 Examples

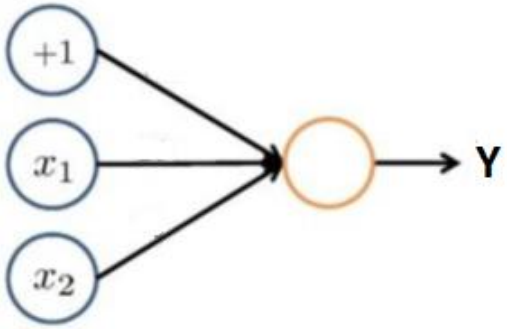
Boolean NOT



x1	Y = g(10 - 20 x1)
0	g(10) = 1
1	g(-10) = 0

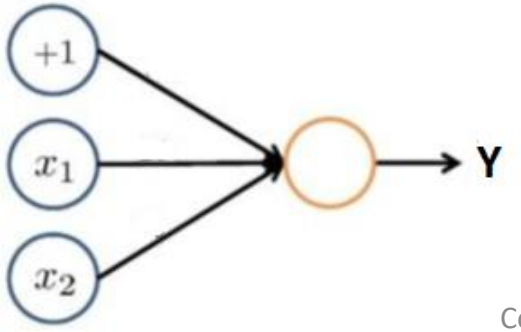


Boolean AND



x1	x2	Y = g(-30 + 20 x1 + 20 x2)
0	0	g(-30) = 0
0	1	g(-10) = 0
1	0	g(-10) = 0
1	1	g(10) = 1

Boolean OR

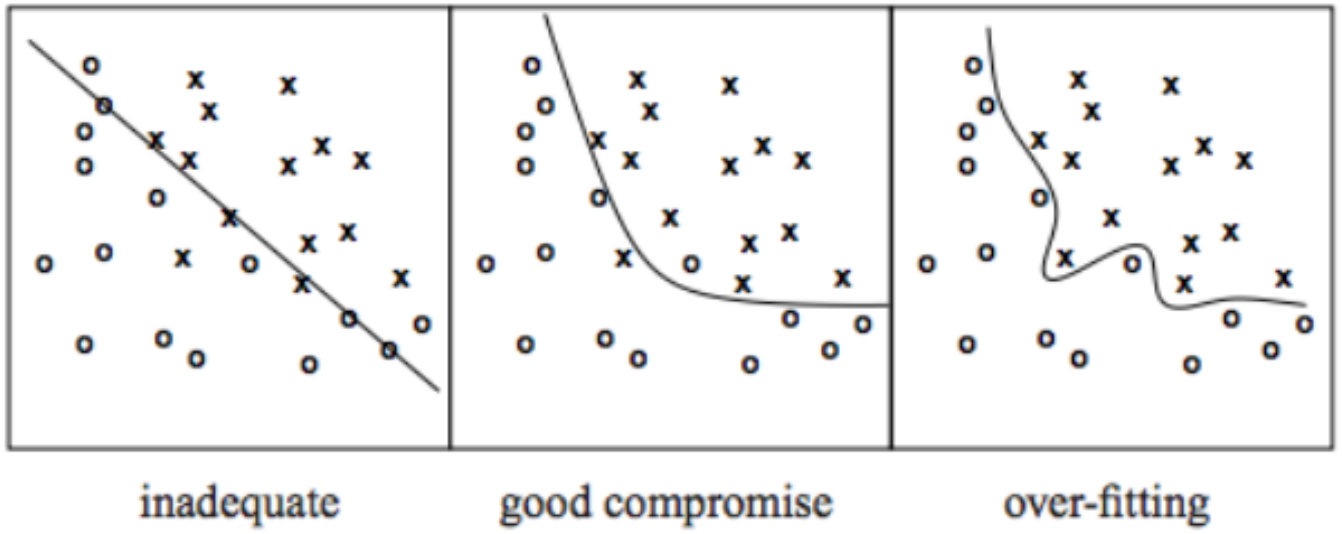


x1	x2	Y = g(-10 + 20 x1 + 20 x2)
0	0	g(-10) = 0
0	1	g(10) = 1
1	0	g(10) = 1
1	1	g(10) = 1

Copyright ACDSD, CSIR-NEIST

11.3.9 Generalization

- Neural networks have two important aspects to fulfill:
  - They must learn decision surfaces correctly (zero error) from training data.
  - They must be able to generalize i.e. deal appropriately with new data.
- Zero error is possible, but so is more extreme overfitting.
- This trade-off between learning and generalization of a NN is called over-fitting.
- Usually we want our neural networks to learn well and also to generalize well.



Occam's Razor (1300s): *"plurality should not be assumed without necessity"* - The simplest model which explains the majority of the data is usually the best.

Copyright ACDSD, CSIR-NEIST

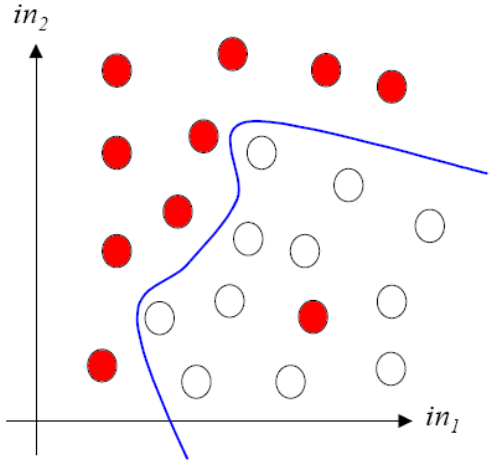
11.3.10 Learning vs. Generalization

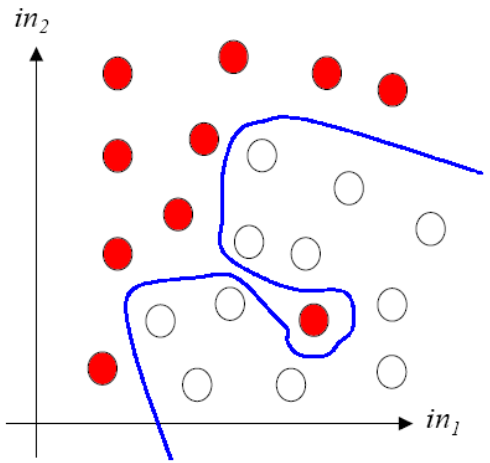
Generalization in classification

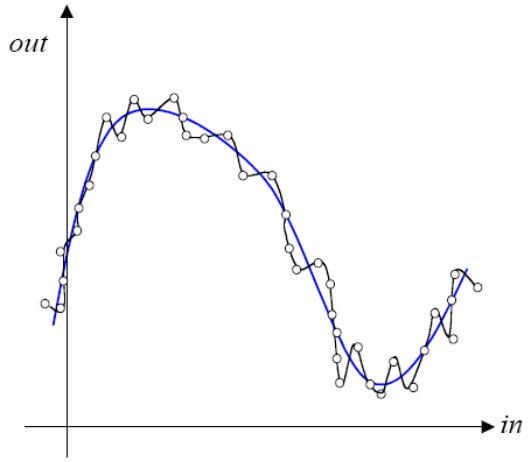
- Allowing ANN output curve not classify all training data totally accurately (allowing misclassification) likely to lead better generalization.

Generalization in regression

- Allowing ANN output curve not passing through all training data (allowing larger error) likely to lead better generalization.





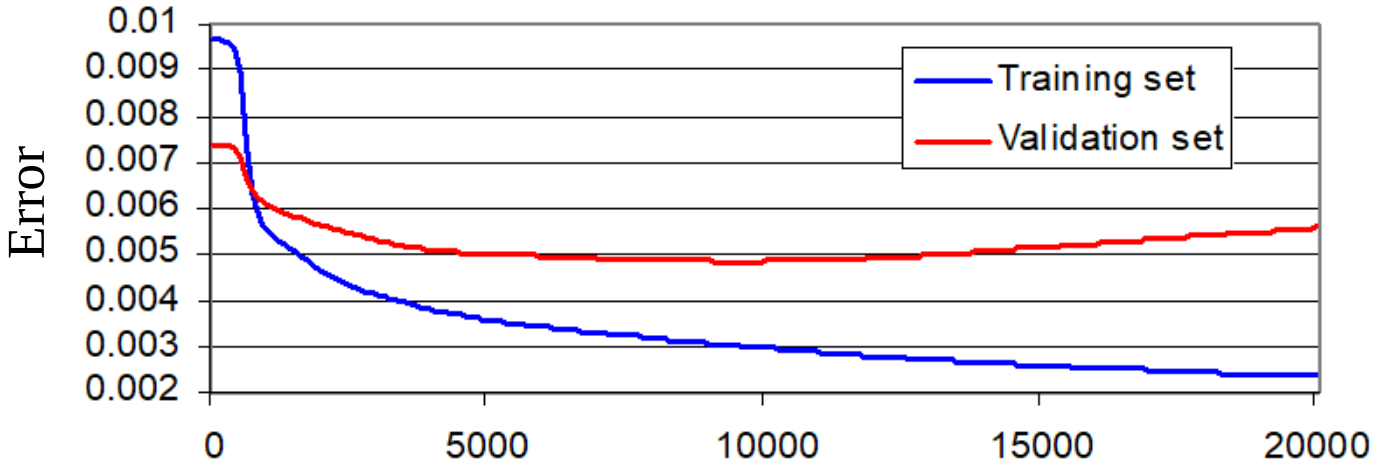


- Training data may contain errors which is not a good thing since it reduces generalization accuracy.
- Thus non-perfect learning is better in this case!
- However, injecting artificial noise (jitter) into the inputs during training improves generalization.

Copyright ACDSD, CSIR-NEIST

11.3.11 Overfitting

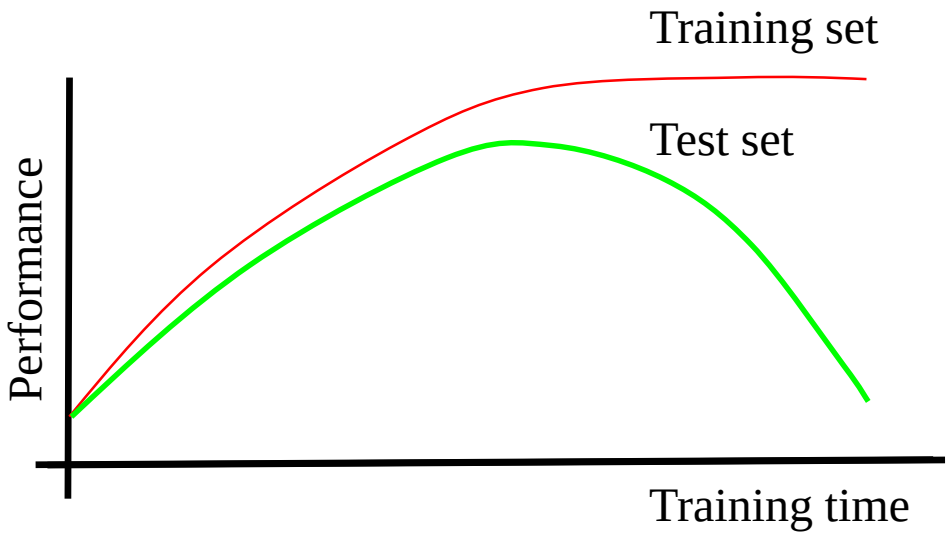
- Due to overtraining (training too much!)
- Overfittings may fit the training data well, even outliers but fail to generalize to new data.
- But loose performance on test sets (new data)



Underfitting (high bias)

Number of weight updates

Overfitting (high variance)



Copyright ACDSD, CSIR-NEIST

11.3.12 Estimation of Generalization Error

Methods for estimating generalization errors

- Single-sample statistics
  - Finding error on each data on generalization.
- Split-sample or hold-out validation (60-20-20)
  - Reserve some data as a "test set", which must not be used during training.
  - The test set must represent the cases that the ANN should generalize to.
  - A re-run with the random test set provides an unbiased estimate of the generalization error.
  - Disadvantage is that it reduces the amount of data available for both training and validation.
- Cross-validation (e.g., leave one out)
  - In k-fold cross-validation the data is divided into k subsets, and the network is trained k times, each time leaving one subset out for computing the error.
  - "Crossing" makes an improvement over split-sampling allowing all data used for training.
  - Disadvantage is that the network must be re-trained many times (k times in k-fold crossing).
- Bootstrapping
  - Works on random sub-samples (random shares) chosen from the full data set.
  - Any data item may be selected any number of times for validation.
  - The sub-samples are repeatedly analyzed.

Copyright ACDSD, CSIR-NEIST

Techniques to reduce overfitting in ANN

Empirical

- Dropout
- DropConnect
- Hand Engineering

Explicit

- Early Stopping
- Number of Parameters
- Weight Decay

Copyright ACDSD, CSIR-NEIST

11.3.13 Regularization Methods

- The "fully-connectedness" of these networks makes them prone to overfitting data.

Dropout (Technique patented by Google)

- Randomly dropping out units (both hidden and visible) along with connections during training.
- Each unit retained with fixed probability p, independent of other units.
- Only the reduced network is trained on the data in that stage.
- The removed nodes are then reinserted into the network with their original weights.
- Decreases overfitting by avoiding training all nodes and improves training speed.

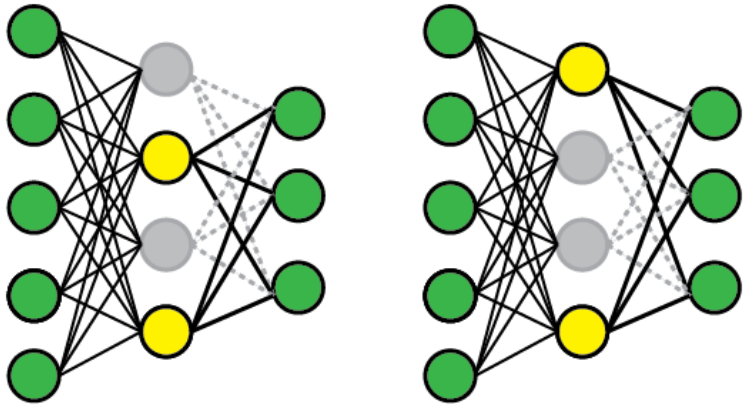
DropConnect

- Generalization of dropout where connection, rather than output unit, dropped with probability 1-p.
- Each unit thus receives input from a random subset of units in the previous layer.
- In other words, the fully connected layer with DropConnect becomes a sparsely connected layer.

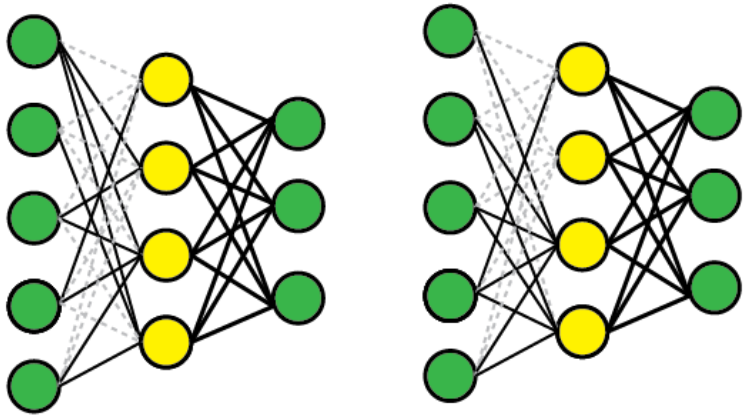
Copyright ACDSD, CSIR-NEIST

11.3.13 Regularization Methods

Dropout (Technique patented by Google)



DropConnect



Note:

- Hyper-parameter p to be chosen (tuned) usually 0.5.
- For input nodes, this is lower because information is directly lost when input nodes are ignored.

Copyright ACDSD, CSIR-NEIST

11.3.14 Batch Normalization

- A technique for improving the speed, performance and stability of ANN introduced in 2015.
- Proposed to solve internal covariate shift.

  "During training as the parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly, such that the current layer needs to constantly readjust to new distributions."
- It is used to normalize the input layer by adjusting and scaling the activations.
- The network can use higher learning rate without vanishing or exploding gradients.
- It regularizes the network such that it is easier to generalize and it is thus unnecessary to use dropout to mitigate overfitting.

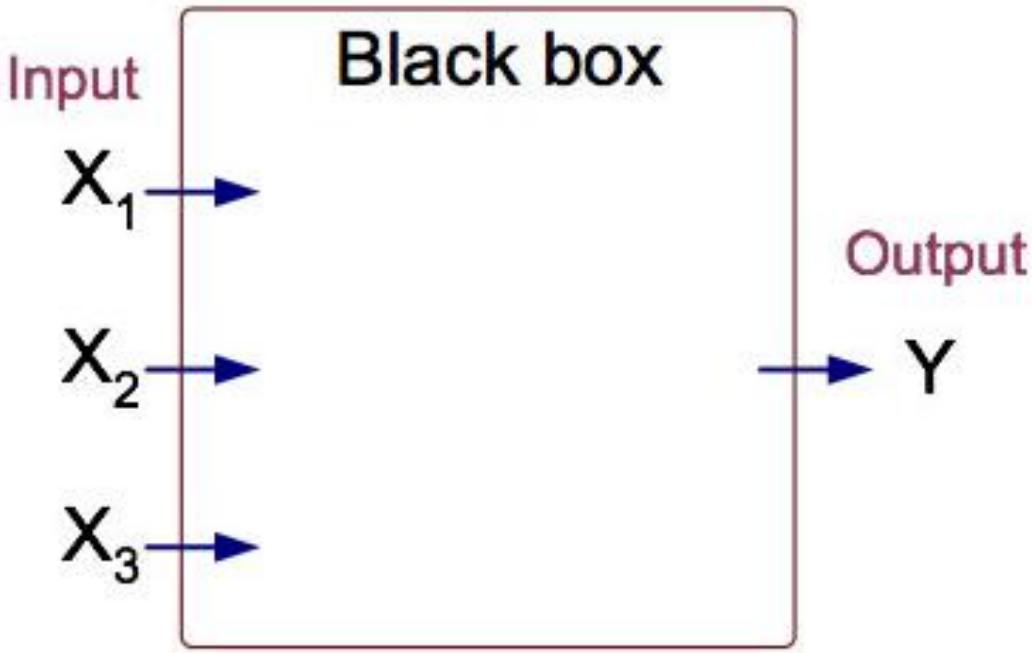
Criticism

- While the effect of batch normalization is evident, the reasons remain under discussion.
- Recently, some study shown that batch normalization does not reduce internal covariate shift, but rather smooths the objective function to improve the performance.

Copyright ACDSD, CSIR-NEIST

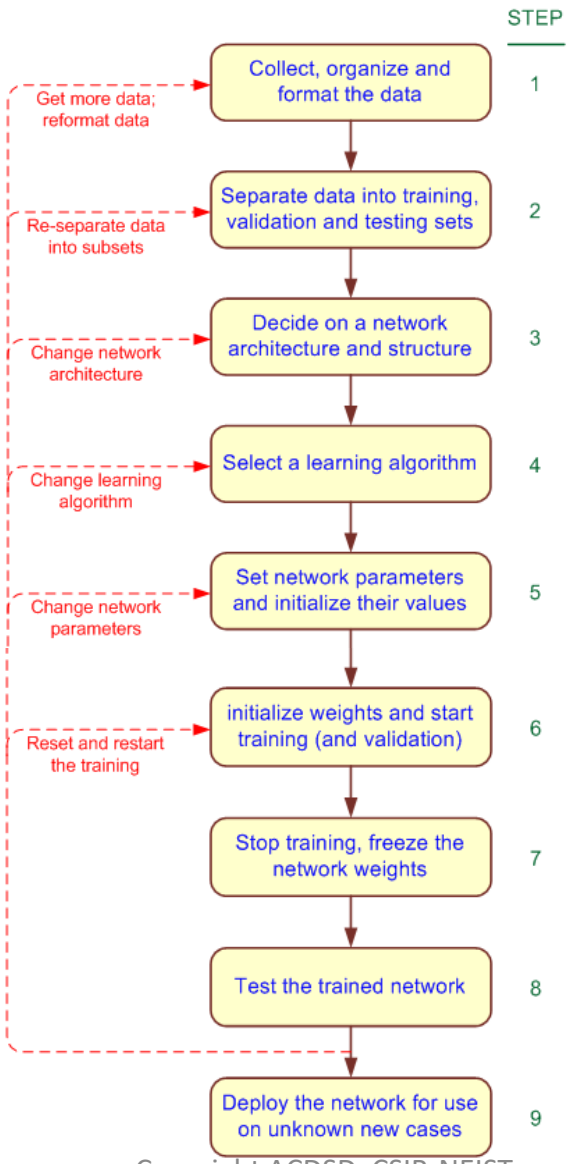
11.3.15 Black Box

- The non-linearities in the feature extraction make interpretation of hidden layers very difficult.
- We cannot say that much about the fitting, the weights and the model.
- Human understanding of what the network does could be limited.
- Suffice to say that the training algorithm has converged and therefore the model is ready to use.
- This leads to Neural Networks being treated as **black boxes**.



Copyright ACDSD, CSIR-NEIST

11.3.16 Summary



Copyright ACDSD, CSIR-NEIST

The Topics Covered In This Section

11.4.1 Introduction
11.4.2 Architecture
11.4.3 Components
11.4.4 Kohonen Map
11.4.5 Two Step Approach
11.4.6 Summary

Copyright ACDSD, CSIR-NEIST

11.4.1 Introduction

What is a Self Organizing Map?

- Neural Net for unsupervised learning.
- Assuming that class membership is broadly defined by the input patterns sharing common features and that the network will be able to identify those features across the range of input patterns.
- It performs dimensionality reduction and clustering both.

Learning

- Based on competitive learning, in which the output neurons compete amongst themselves to be activated with the result that only one is activated at any one time.
- This activated neuron is called a winner-takesall neuron or simply the winning neuron.

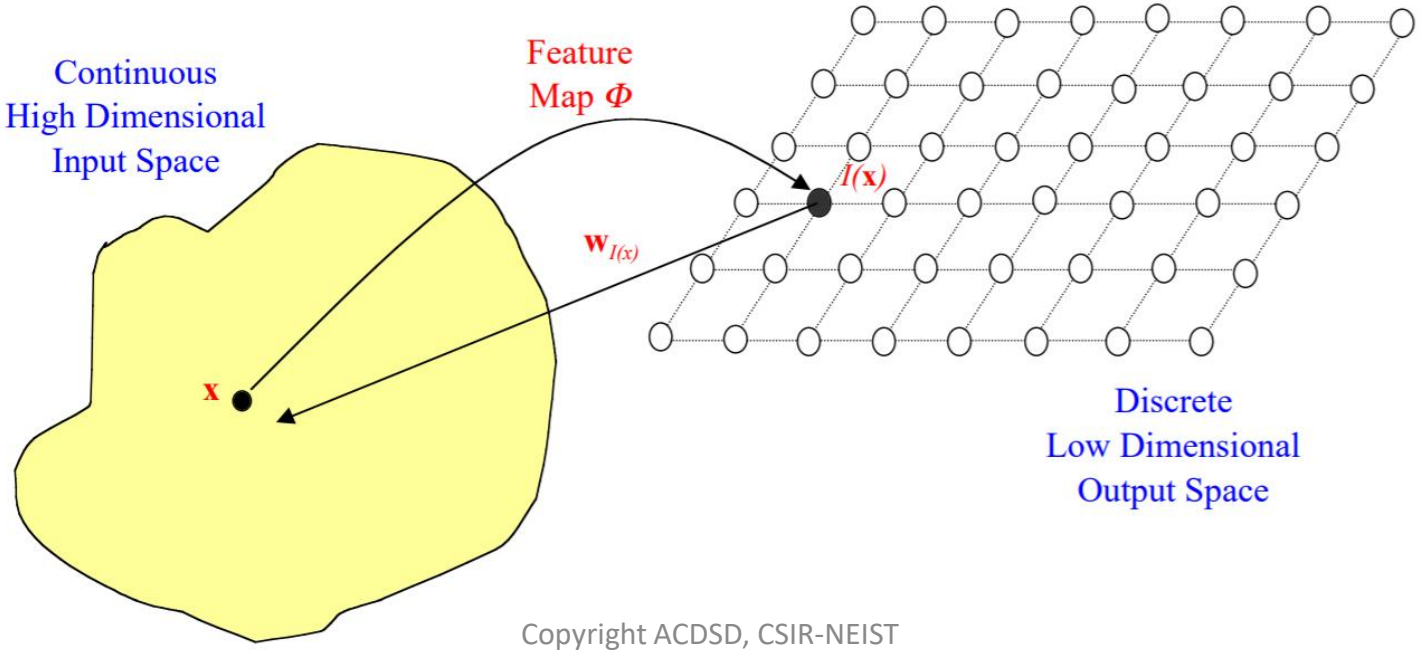
Why Self Organization?

- Competition implemented by having lateral inhibition connections (negative feedback paths) between neurons resulting the neurons are forced to organize themselves.

Copyright ACDSD, CSIR-NEIST

11.4.2 Architecture

- The goal is to transform an input pattern of arbitrary dimension into a 1 or 2 dimensional map.
- Therefore set up SOM by placing neurons at the nodes of a 1 or 2 or higher dimensional lattice.
- The neurons become selectively tuned to various input pattern class during competitive learning.
- The locations of the neurons so tuned (i.e. the winning neurons) become ordered and a meaningful coordinate system for the input features is created on the lattice.
- We can view this as a non-linear generalization of principal component analysis (PCA).



Copyright ACDSD, CSIR-NEIST

11.4.3 Components

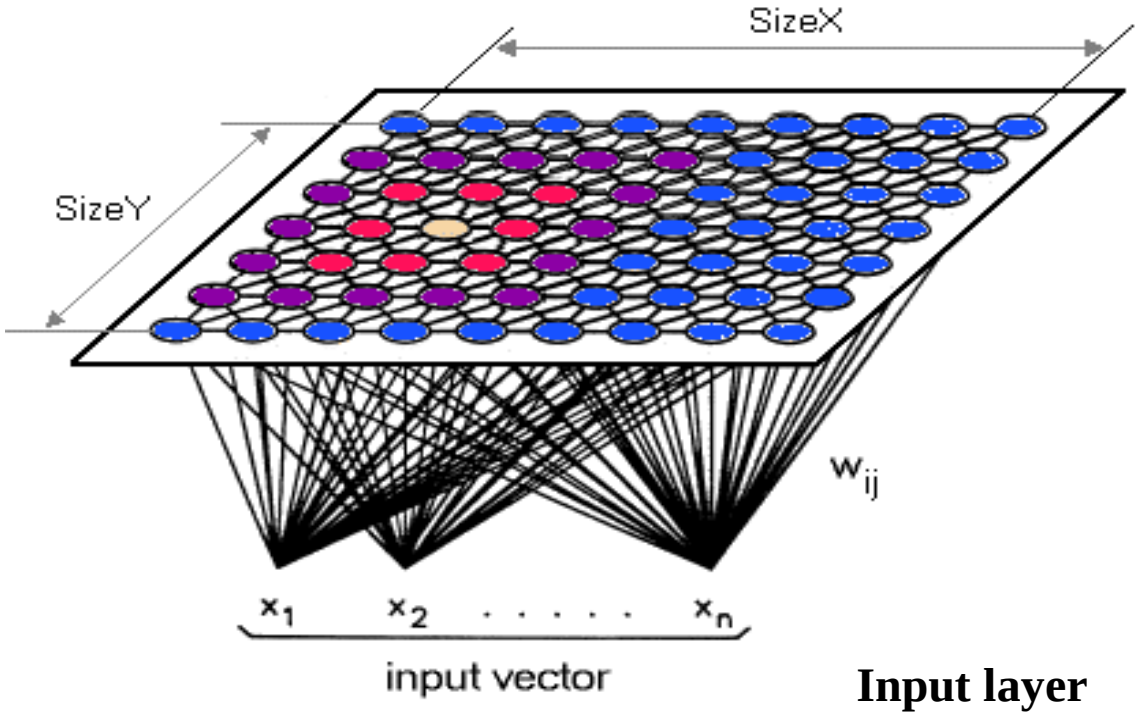
The self-organization process involves four major components:

- **Initialization:** All the connection weights are initialized with small random values.
- **Competition:** For each input pattern, the neurons compute their respective values of a discriminant function which provides the basis for competition. The particular neuron with the smallest value of the discriminant function is declared the winner.
- **Cooperation:** The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.
- **Adaptation:** The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights, such that the response of the winning neuron to the subsequent similar input pattern is enhanced.

Copyright ACDSD, CSIR-NEIST

11.4.4 Kohonen Map

- Particular kind of SOM developed by professor Kohonen [1982].
- It has a a single computational layer arranged in rows and columns.
- Each neuron is fully connected to all the source nodes in the input layer.



Computational layer

- Each neuron (node) corresponds to a set of instances from the dataset.
- Each neuron (node) is associated a vector of weights (codebook) which describes the profile of the neuron.
- Positions of the neurons in the map are important i.e.
  - Two neighboring neurons have similar codebook
  - A set of contiguous neurons correspond to a particular pattern in the data.

Input layer

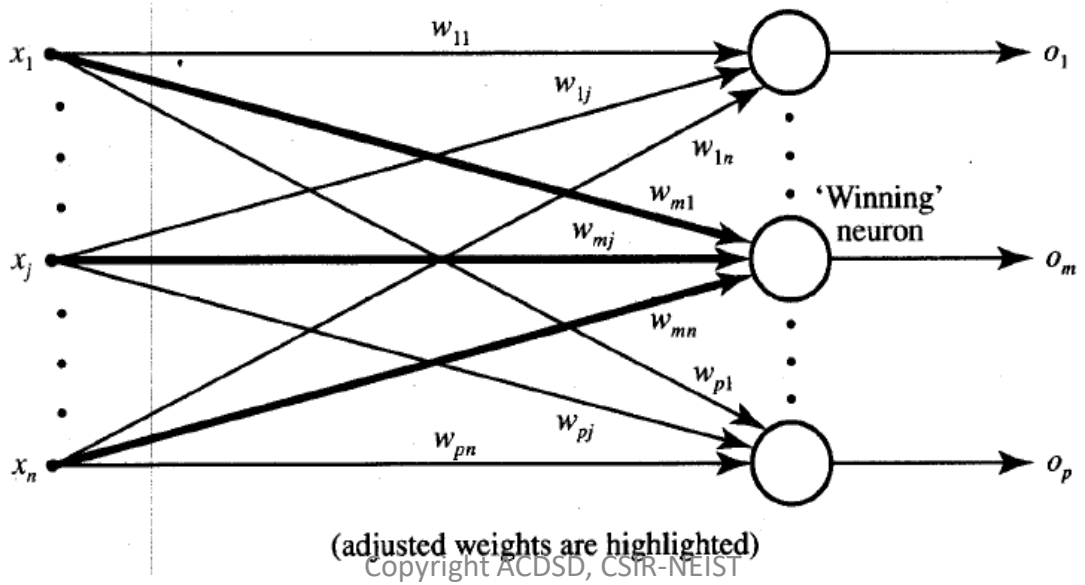
Input space, description of the dataset into the original representation space (vector with n values for n variables).

Copyright ACDSD, CSIR-NEIST

11.4.4 Kohonen Map

Algorithm

1. Initialize each node's weights
2. Present a randomly selected input vector to the lattice
3. Competition - Determine most resembling winning node based on Euclidean distance
4. Cooperation - Determine the topological neighboring nodes
5. Adaptation - Adjusted the winning and neighboring nodes (learning rate)
6. Repeat steps 2-5 for until a stopping criteria is reached



Copyright ACDSD, CSIR-NEIST

11.4.4 Kohonen Map

Scale the variables to avoid the over influence of the variables with high variance

```
Input: dataset, size and topology of the map
Output: the codebook for each output node

1. Randomize the map's nodes weight
2. Select randomly one instance
3. Find the closest node: best matching unit
4. The codebook of this node is updated
5. The codebooks of the adjacent nodes are
   also updated, by not to the same degree
6. Reduce the intensity of the update
   progressively
7. Repeat 1 to 6 for Tmax iterations
```

(1) Initialization phase

(2) We process all instances from the data. One instance may be processed several times.

(3) Competition phase. A distance measure must be defined between the codebook and the description of the instances.

(4) Update the weights (codebook).

(5) Cooperation phase. This is what ensures the similarity of weights between contiguous nodes. The size of the neighborhood to be considered is reduced progressively. Node: if we ignore the neighborhood structure, the algorithm corresponds to the K-means approach.

(6) Adaptation. At first, high learning rate, move quickly to the solution; at the end, small learning rate, to avoid oscillations.

11.4.5 Two Step Approach

- Perform a pre-clustering using the SOM method which can process very large database.
- Start the HAC (Hierarchical Agglomerative Clustering) from these pre-clusters.
- Often (not always), the adjacent nodes of the topological map belong to the same final cluster.

Advantage

- HAC dendrogram gives clear understanding and identification of hierarchy of nested partitions.

Disadvantage

- HAC requires the calculation of distances between each pair of individuals (distance matrix).
- This is too time consuming on large datasets (in number of observations).

Copyright ACDSD, CSIR-NEIST

11.4.6 Summary

Pros

- Performs both dimensionality reduction, data visualization and cluster analysis (clustering).
- This is a nonlinear approach for dimensionality reduction (vs. PCA for instance).
- Numerous visualization possibilities
- The two-step approach (with HAC) for clustering is especially attractive.

Cons

- Processing time may be long on very large bases (we need to pass several times the individuals).
- Visualization and interpretation of codebooks becomes difficult when number of variables is high.

Copyright ACDSD, CSIR-NEIST

The Topics Covered In This Section

11.5.1 Introduction
11.5.2 Architecture
11.5.3 Updating
11.5.4 Energy Function
11.5.5 Learning
11.5.6 Exercises

Copyright ACDSD, CSIR-NEIST

11.5.1 Introduction

- A form of recurrent NN popularized by John Hopfield[1982], but described earlier by Little[1974].
- Serve as content-addressable ("associative") memory systems with binary threshold nodes.
- They are guaranteed to converge to a local minimum and, therefore, may converge to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum).

Applications

- Applies to solving complex computational problems (e.g., optimization, association).
- Act as “autoassociative” memories to store patterns.
- Used for pattern association, pattern categorization, pattern classification etc.

Copyright ACDSD, CSIR-NEIST

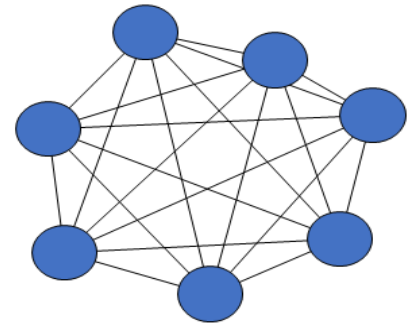
11.5.2 Architecture

- Hopfield nets have highly interconnected neurons (All neurons connected to each other).
- Every pair of units *i* and *j* in the net has a connection described by the connectivity weight w_{ij}
- In other words, Hopfield net can be formally described as a complete undirected graph G(V, f)

  where V is a set of McCulloch-Pitts neurons

  and f: $V^2 \rightarrow R$ is a function that links pairs of units to a real value, the connectivity weight.

- Hopfield net units (McCulloch-Pitts neurons) are binary threshold units, take values 1, -1. (or 0,1)
- The value is determined by whether or not the units' input exceeds their threshold Θ
- The connections in a Hopfield net typically have the following restrictions:
  - $w_{ij} = 0$ for all i (No unit has a connection with itself)
  - $w_{ij} = w_{ij}$ for all i (Connections are symmetric)



Completely Connected

Note: The constraint that weights are symmetric guarantees that the energy function decreases monotonically while following the activation rules.

Copyright ACDSD, CSIR-NEIST

11.5.3 Updating

Updating unit is performed using the following rule:

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

- w_{ij} is the strength of the connection weight from unit j to unit i (the weight of the connection)
- s_i is the state of unit i
- Θ_i is the threshold of unit i

Updates in the Hopfield network can be performed in two different ways:

- **Asynchronous**: Only one unit is updated at a time. This unit can be picked at random, or a pre-defined order can be imposed from the very beginning.
- **Synchronous**: All units are updated at the same time. This requires a central clock to the system in order to maintain synchronization. (less realistic)

Copyright ACDSD, CSIR-NEIST

11.5.3 Updating

Neurons "attract or repel each other" in state-space

- The weight between two units has a powerful impact upon the values of the neurons.
- Consider the connection weight w_{ij} between two neurons i and j.
- If $w_{ij} > 0$ the updating rule implies that:
  - when $s_j = 1$, the contribution of j in the weighted sum is positive. Thus, s_i is pulled by j towards its value $s_i = 1$
  - when $s_j = -1$ the contribution of j in the weighted sum is negative. Then again, s_i is pushed by j towards its value $s_i = -1$

- Thus, the values of neurons i and j will converge if the weight between them is positive. Similarly, they will diverge if the weight is negative.

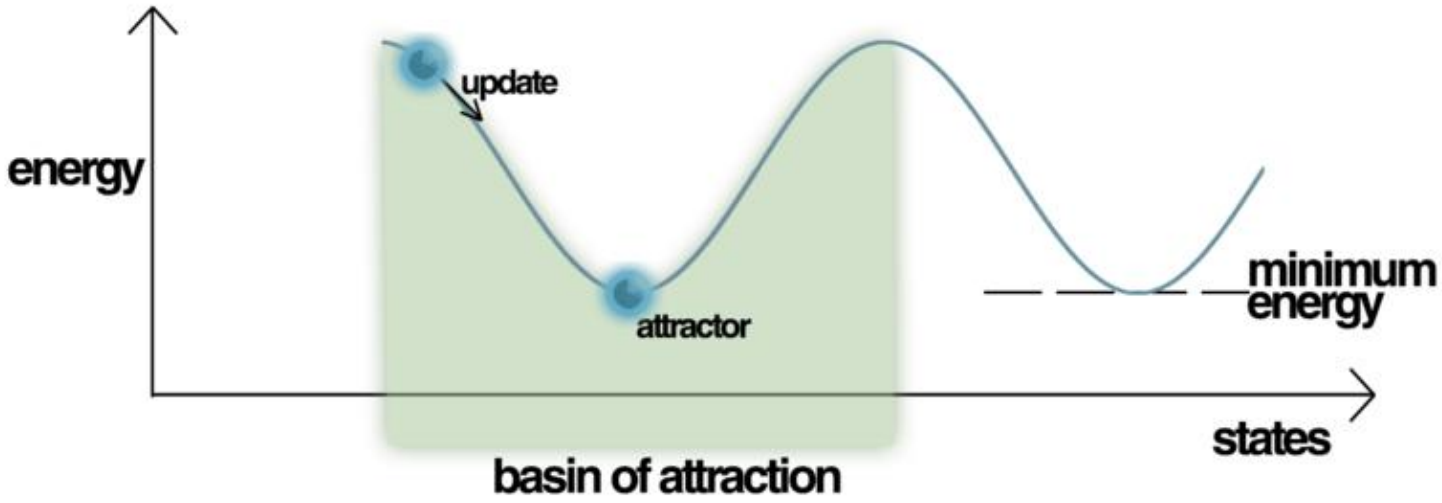
Copyright ACDSD, CSIR-NEIST

11.5.4 Energy Function

- Hopfield nets have a scalar value associated with each state of the network referred to as energy (E) of the network (analogous to gradient descent error function)).

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

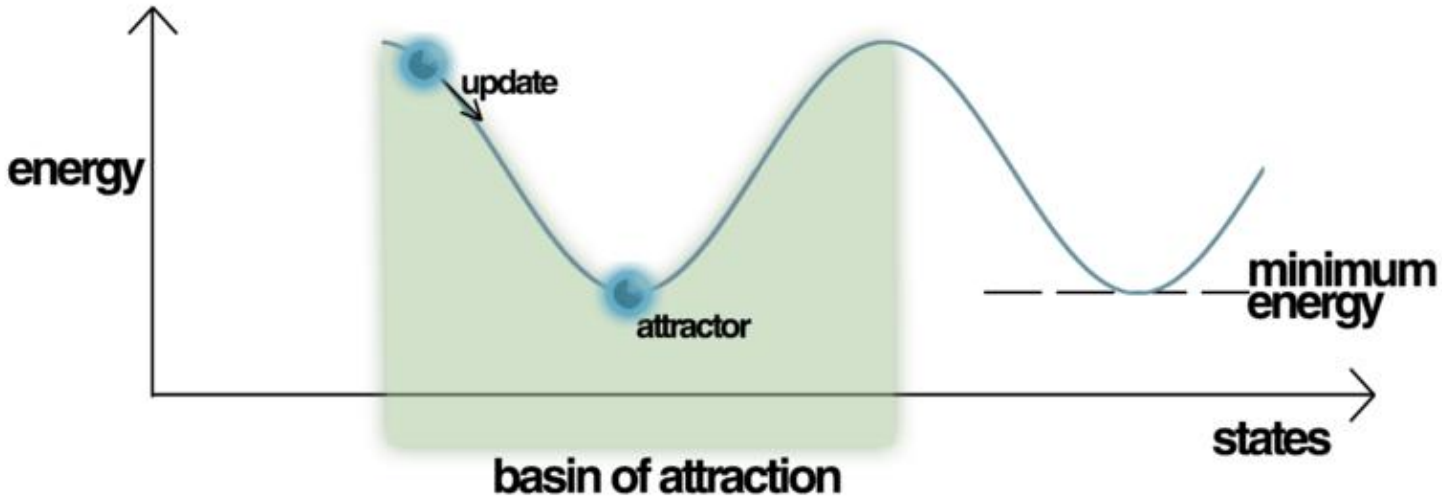
- Its called "energy" because it either decreases or stays the same upon network units being updated.
- Furthermore, under repeated updating the network will eventually converge to a state which is a local minimum in the energy function.
- Thus, if a state is a local minimum in the energy function it is a stable state for the network.



Copyright ACDSD, CSIR-NEIST

11.5.4 Energy Function

- Hopfield showed that asynchronous updating in symmetric networks minimizes an "energy" function and leads to a stable final state for a given initial state.
- Asynchronous updating of outputs
  - If s_i is initially –1 and $\sum_j w_{ij}$ sj> Θ_i then s_i becomes +1
    - Change in E = -1/2 $\sum_j$ (wijsj + wji sj)+ □$_i$ = - $\sum_j w_{ij}$ sj + Θ_i< 0
  - If s_i is initially +1 and $\sum_j w_{ij}$ sj< Θ_i, then s_i becomes -1
    - Change in E = 1/2 $\sum_j$ (wij sj + wji sj) − □$_i$= $\sum_j w_{ij}$ sj - Θ_i< 0



Copyright ACDSD, CSIR-NEIST

11.5.5 Learning

- In 1949 neuropsychologist Donald Hebb postulated how biological neurons learn:

> ***"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells such that A's efficiency as one of the cells firing B, is increased."***

- In other words if two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased.
- This rule is often supplemented by: If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

> ***"Neurons that fire together, wire together. Neurons that fire out of sync, fail to link".***

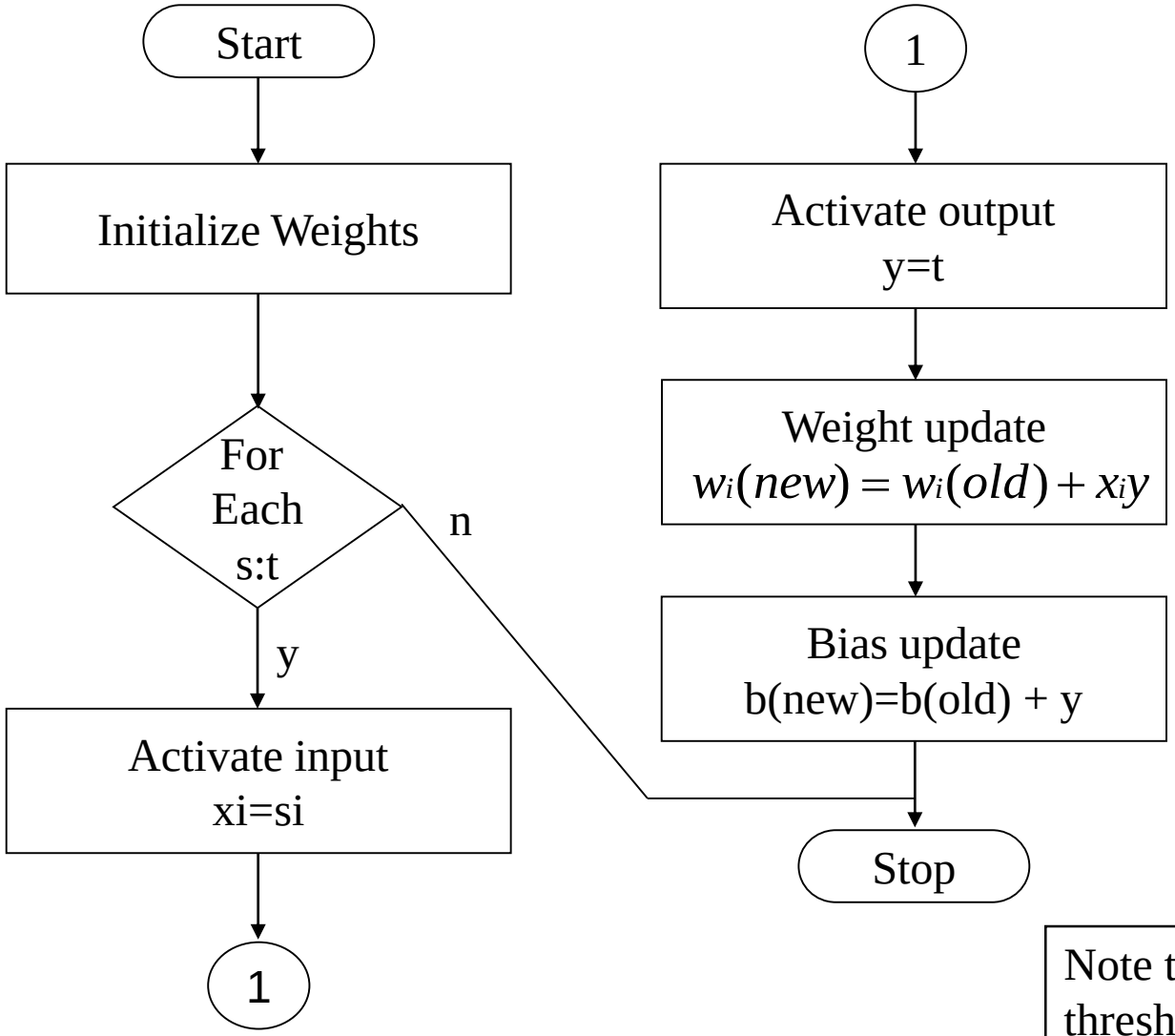
Algorithm

- Step 0: Initialize all weights (For simplicity, set weights and bias to zero)
- Step 1: For each input training vector do steps 2-4
- Step 2: Set activations of input units $x_i = s_i$
- Step 3: Set the activation for the output unit $y = t$
- Step 4: Update weights and bias $w_{ij}(new) = w_{ij}(old) + in_i out_j = w_{ij}(old) + x_i y_j$

$$b(new) = b(old) + y$$

Copyright ADDS, CSIR-NEIST

11.5.5 Learning



Note that, in contrast to Perceptron training, the thresholds of the neurons are never updated.

Copyright ACDSD, CSIR-NEIST

Worked Examples

Question 1: A drug company decided to deploy a new feed-forward neural network system to calculate the efficacy of a drug. Explain the following –

a) Suggest what should company have in hand in order to implement the system and use it?

b) Mention a problem associated with this requirement.

Answer:

a) The company should have historical data about all the medicines that will be used as training set.

b) The network will not be able to produce an accurate answer for a medicine very different from those in the training set.

Copyright ACDSD, CSIR-NEIST

Question 2: An ANN require lots of data to train it properly. But sometimes it is not possible to collect that amount of data to train the neural net. Considering that situation answer the following -

a) Is it feasible to go for splitting up of that few data into training set and a test set? Justify.

b) Give one better solution to tackle this situation to improve the efficiency NN performance?

Answer:

a) No, not a feasible idea

b) Augmentation

Copyright ACDSD, CSIR-NEIST

Question 3: A student wants to use ANN to automatically determine the species of mammal from a dataset having different features of 10 species of mammals. The features given in the dataset are 'length', 'breadth', 'weight', and 'color' in the numerical values. The dataset contains 800 different labeled samples of these 10 species. Answer the following regarding the ANN architecture that need to be designed to correctly classify the species of mammal and justify your answer –

a) How many input and output units should the ANN have?

b) Should the ANN use hidden units or not?

c) Should the ANN use feedforward connections, recurrent connections, both, or neither?

d) What activation function(s) should the neurons use?

e) What learning mechanism(s) should the ANN use?

Answer:

a) 4 input and 10 output units are required.

b) Yes, the ANN should have hidden units.

c) The ANN should use feed forward connections.

d) The hidden layer should use sigmoid function and output units should use softmax function.

e) Backpropagation learning mechanism is required.

Copyright ACDSD, CSIR-NEIST

Question 4: Consider three problems A, B, and C as follows –

Problem A: A regression problem

Problem B: A binary classification problem

Problem C: A multiclass (3 class) problem

Answer the following -

a) Is it possible to address all the above problems with a 3-3-1 neural net? If not, why?

b) What will be the learning algorithm, loss function & activation function used for each problem?

c) What necessary changes to make to address all the above problem by the given net?

Answer:

a) No, its not possible because the output unit is 1

b) Sigmoid will work on problem A and B but not C. For C we must use softmax at last layer.

c) For problem A and B it may work. But for C we have to increase the output layer units to 3.

Copyright ACDSD, CSIR-NEIST

Question 5: Calculate the number of parameters for the following ANN –

a) A fully-connected neural network with 5 hidden layers, each with 10 hidden units. The input is 20-dimensional and the output is a scalar.

b) A 5-5-5 architecture neural network?

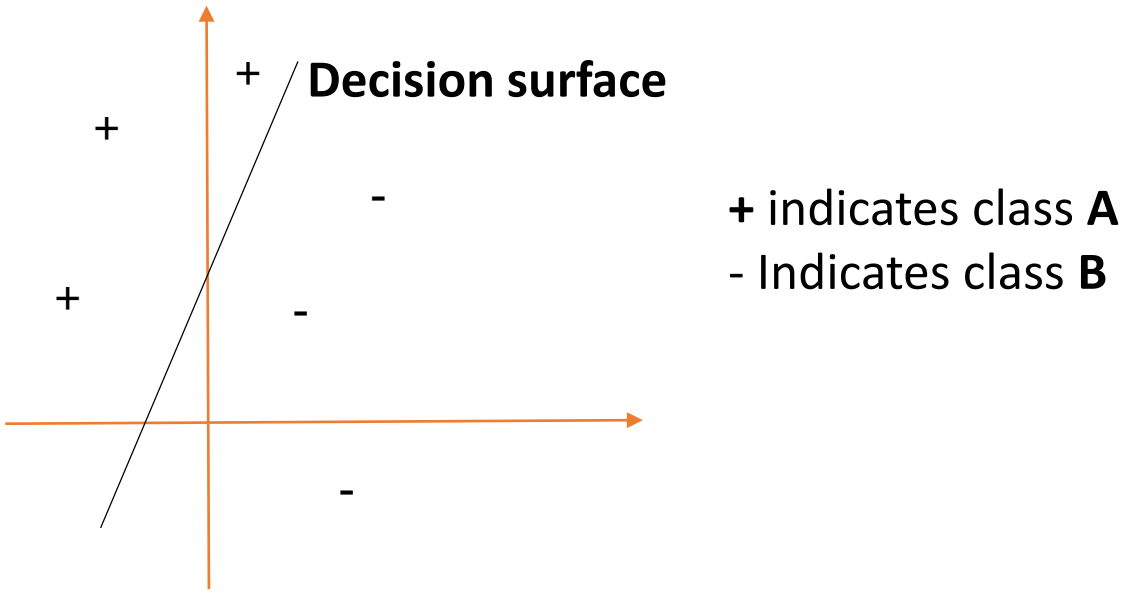
Answer:

a) (20+1)*10 + (10+1)*10*4 + (10+1)*1 = 661

b) 60

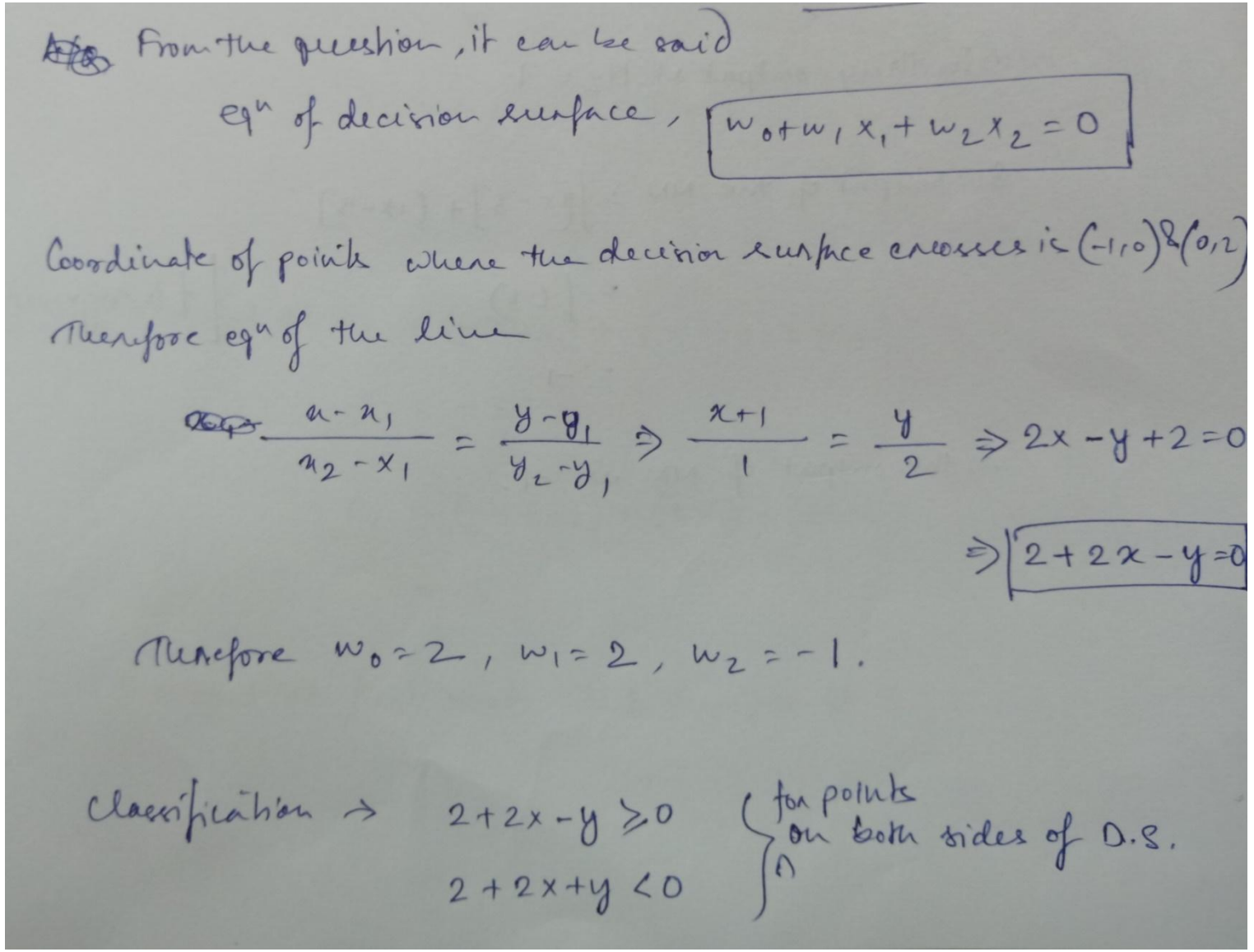
Copyright ACDSD, CSIR-NEIST

Question 6: Consider a dataset having 2 attributes ‘X1’ and ‘X2’ and 6 sample instances belonging to either class A or B. If the decision surface which is used to classify the data points correctly in a 2D space crosses X1 axis at -1 and X2 axis at 2 as shown in the figure. What will be the weights of w0, W1, W2 of the perceptron?



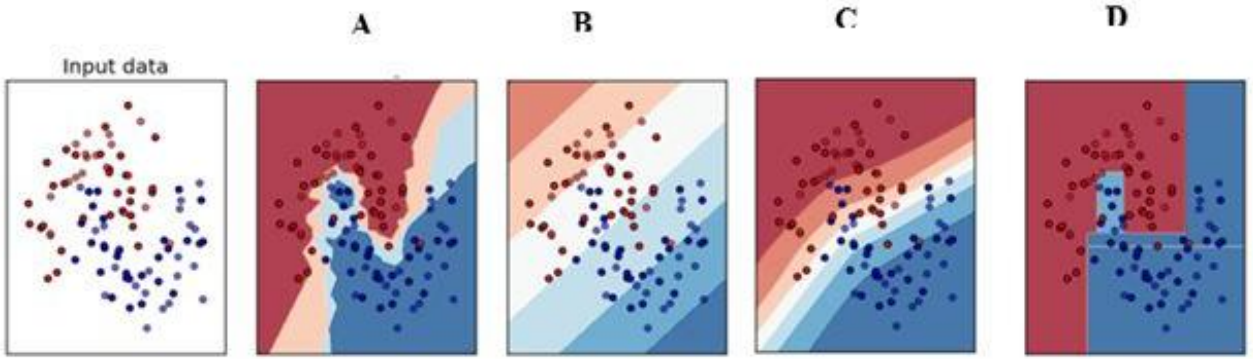
Copyright ACDSD, CSIR-NEIST

Solution:



Copyright ACDSD, CSIR-NEIST

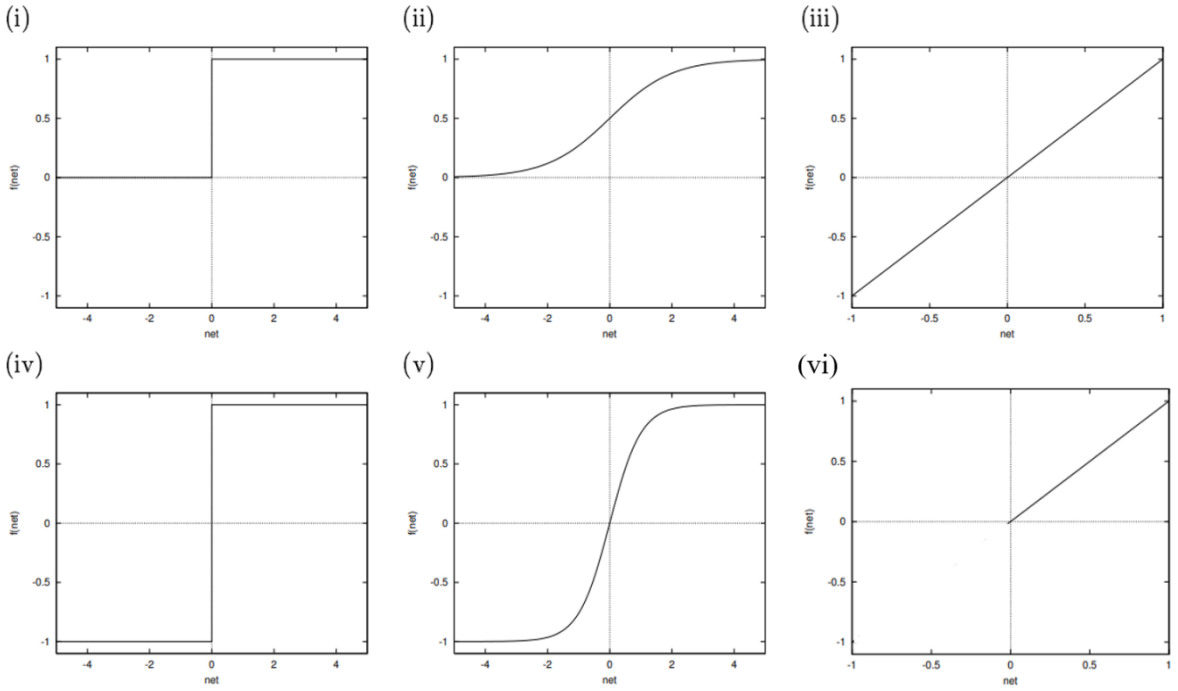
Question 7: Which of the following is a decision boundary of an artificial neural network? Justify your answer.



Answer: all are possible

Copyright ACDSD, CSIR-NEIST

Question 8: Identify each of the following activation function from the following –



Answer:

i. Unipolar step

ii. Unipolar sigmoid

iii. Linear

iv. Bipolar step

v. Bipolar sigmoid

vi. Relu

Copyright ACDSD, CSIR-NEIST

Question 9: For a NN assume the following –

- All the inputs to the network are negative numbers.
- The weights assigned from inputs units are all positive.
- Activation function used in each neurons is Relu function.

Answer the following -

a) What will be the strange thing you will observe in the NN in the training process?

b) How will you overcome the problem?

Answer:

a) Output at each unit will be 0 and therefore all of them behave like dead neuron.

b) Solution is to use different activation function depending upon state of the problem.

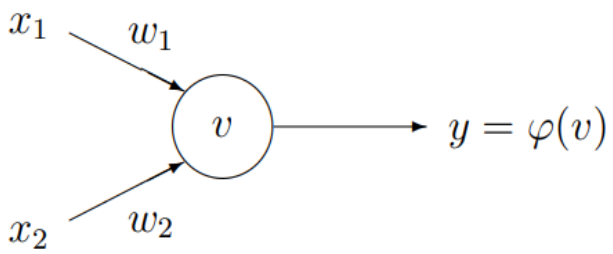
Copyright ACDSD, CSIR-NEIST

Question 10: You're solving a binary classification task. The final two layers in your network are a ReLU activation followed by a sigmoid activation. What will happen?

Answer: Using ReLU then sigmoid will cause all predictions to be positive.

Copyright ACDSD, CSIR-NEIST

Question 11: Consider a neural network as shown below



If the weights are w1 = 1 and w2 = 1

and the activation function is: $\phi(v) = 1$ if $v \geq 2$ or 0 otherwise

a) Test how the logical AND function works.

b) What will be the necessary changes in the weights to implement logical OR function.

c) Is it possible to do implement with above neural net? If not, why? Provide a solution to do this by bring some necessary changes in the architecture? Also mention why and how the modified architecture works perfectly?

Answer:

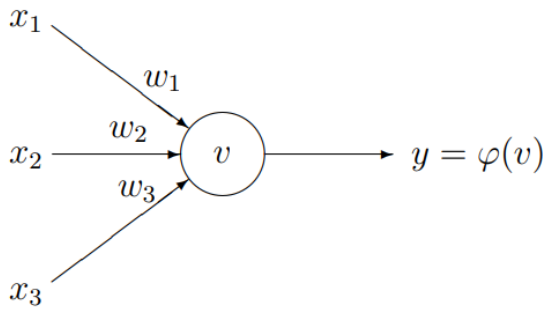
a) By given weights it follows AND function for any two binary inputs.

b) Weights need to change to 2 each (one possible solution)

c) Not ppossible with given network, we must have one layer because of non-linearity.

Copyright ACDSD, CSIR-NEIST

Question 12: Consider a neural network as shown below



Assume the three input nodes x = (x1, x2, x3) receive only binary signals (either 0 or 1).

a) How many different input patterns this node can receive?

b) What if the node had four inputs or Five?

c) What will be the formula that computes the number of binary input patterns for a given number of inputs?

Answer:

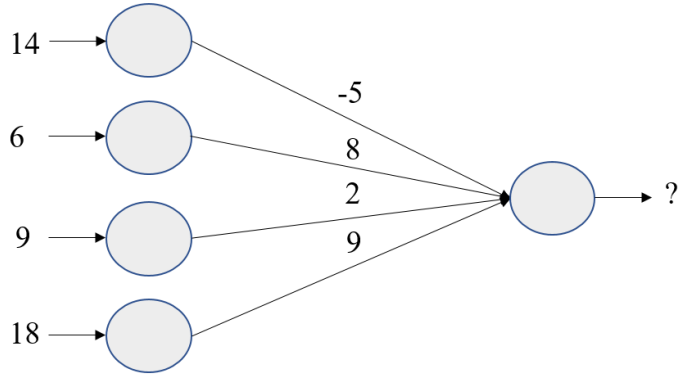
a) 8

b) For 4 its 16 and for 5 its 25

c) Generalise formula is 2^n (number of different inputs)

Copyright ACDSD, CSIR-NEIST

Question 13: A 4 inputs neural net has weights -5, 8, 2, and 9. The transfer function is linear with the constant of proportionality being equal to 3. The inputs are respectively 14, 6, 9, and 18. What will be the output of the NN in the feed forward phase (No need for backpropagation).

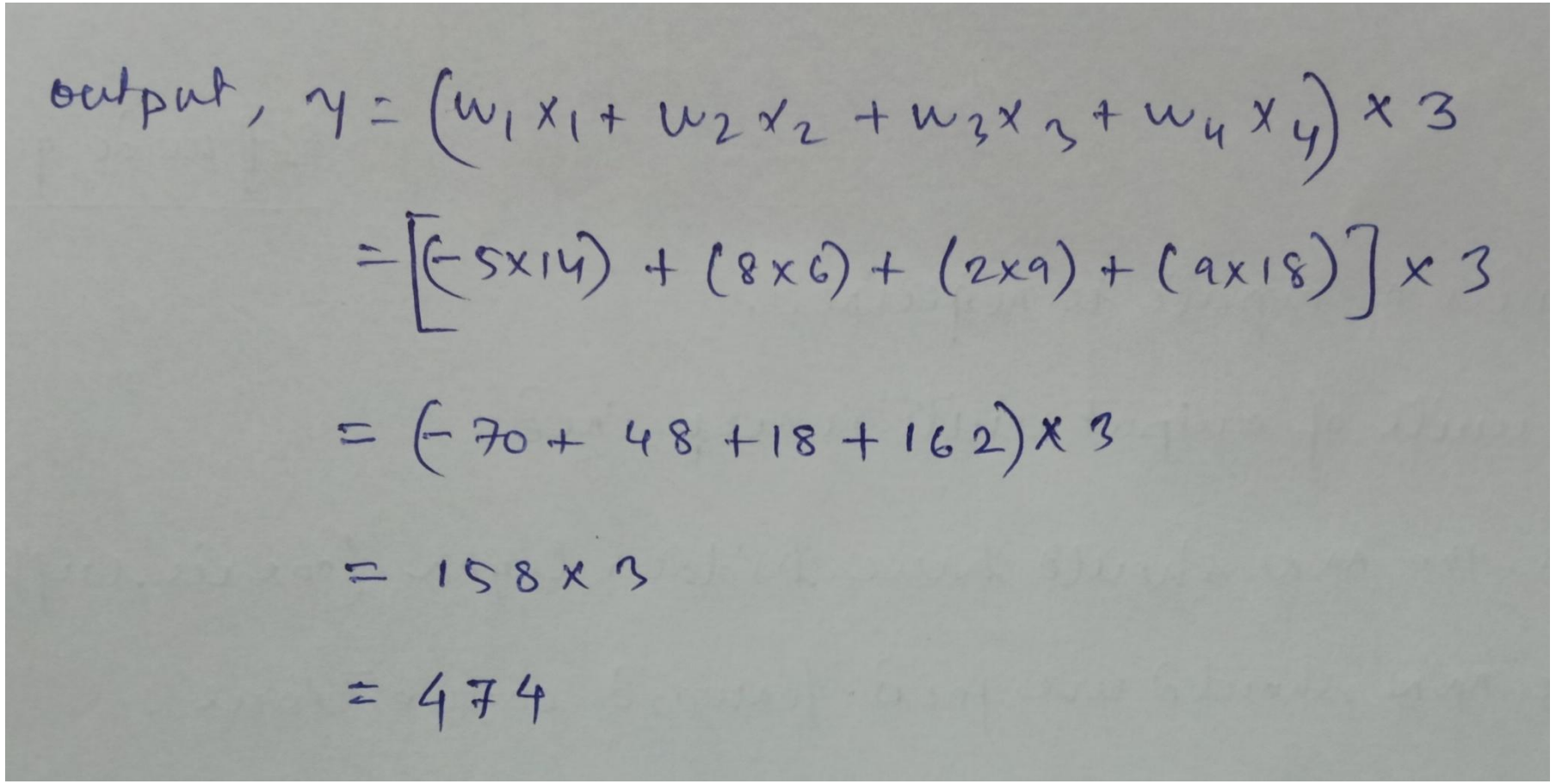


Answer:

Output, $Y = [w1x1 + w2x2 + w3x3 + w4x4] \times 3 = [(-5x14) + (8x6) + (2x9) + (9x18)] \times 3 = 474$

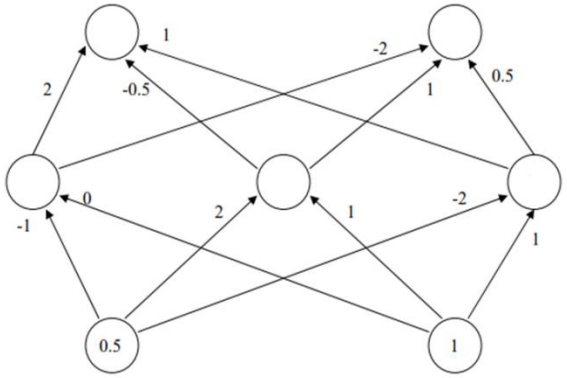
Copyright ACDSD, CSIR-NEIST

Solution:



Copyright ACDSD, CSIR-NEIST

Question 14: The following is a network of linear neurons, that is, neurons whose output is identical to their net input, $y_i = net_i$.

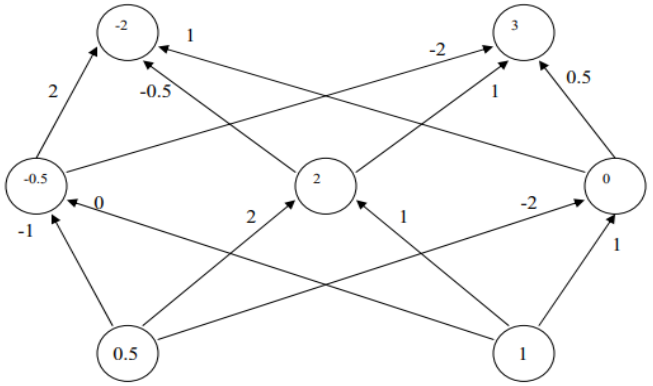


a) Compute the output of the hidden-layer and the output-layer neurons for the given input (0.5, 1) and enter those values into the corresponding circles.

b) What is the output of the network for the input (1, 2), i.e. the left input neuron having the value 1 and the right one having the value 2? Do you have to do all the network computations once again in order to answer this question? Explain why you do or do not have to do this.

Answer:

a)

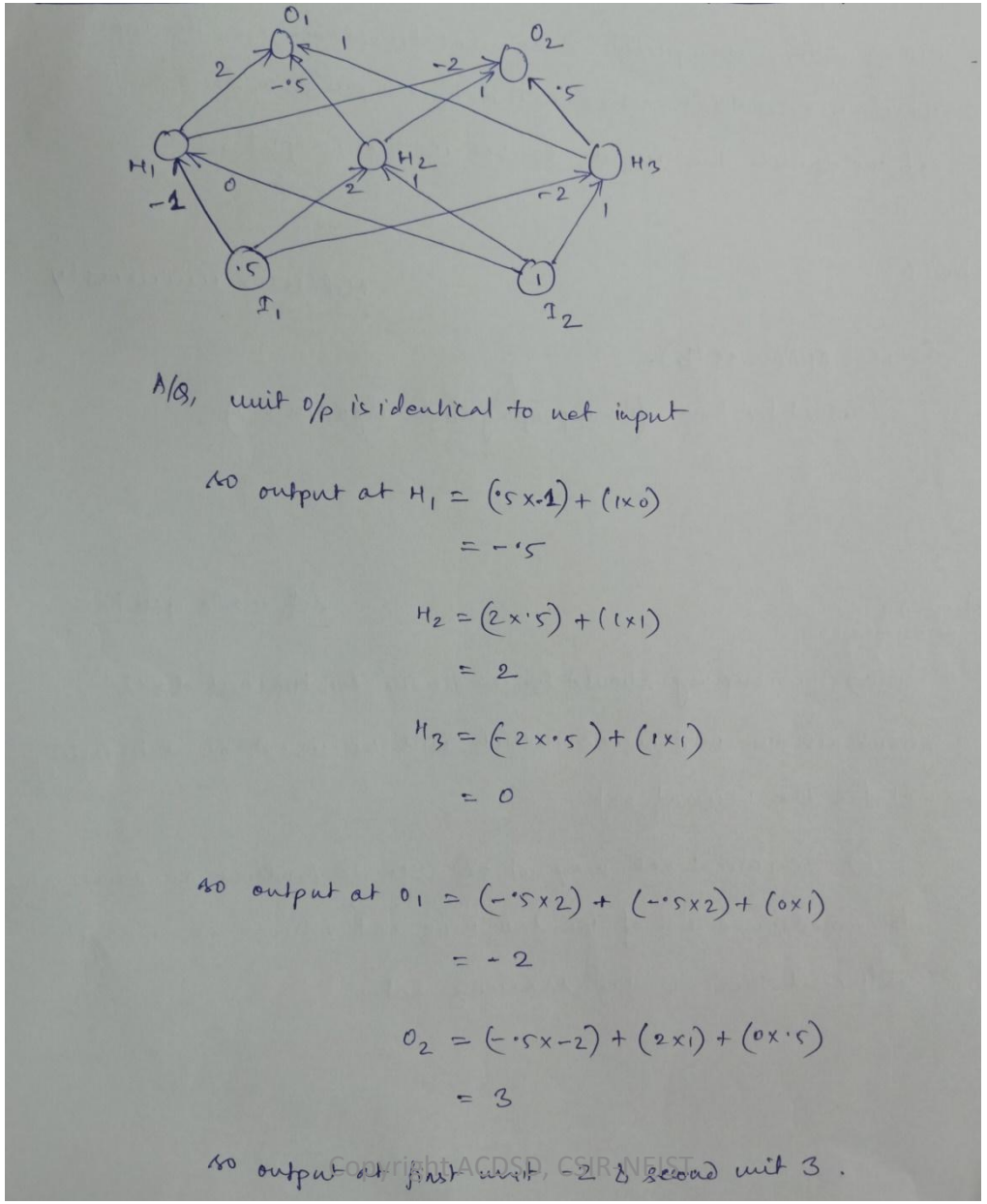


b) It is (-4, 6). Here we do not need to do all computations because every neuron computes a linear function on its inputs which means that entire network computes a linear function. If we double the input, double is the output.

Copyright ACDSD, CSIR-NEIST

Worked Examples

Solution:



A/Q, unit o/p is identical to net input

$\therefore$ output at $H_1 = (.5 \times -1) + (1 \times 0)$

$= -.5$

$H_2 = (2 \times .5) + (1 \times 1)$

$= 2$

$H_3 = (-2 \times .5) + (1 \times 1)$

$= 0$

$\therefore$ output at $O_1 = (-.5 \times 2) + (-.5 \times 2) + (0 \times 1)$

$= -2$

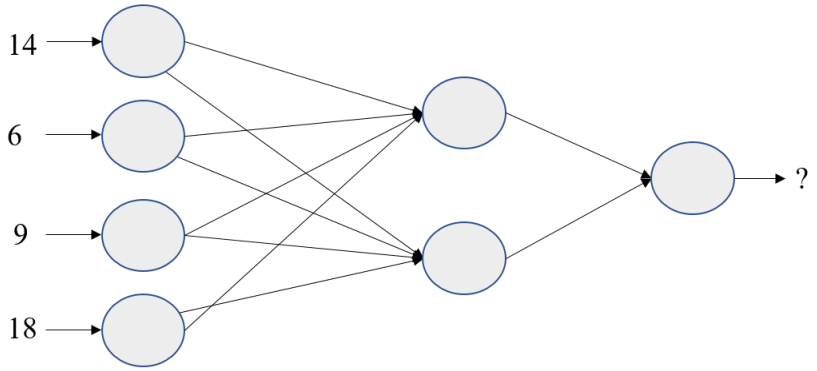
$O_2 = (-.5 \times -2) + (2 \times 1) + (0 \times .5)$

$= 3$

$\therefore$ output at first unit = -2 & second unit 3.

Copyright ACDSD, CSIR-NEIST

Question 15: Consider a neural net having 4 inputs and a hidden layer of 2 units and one output unit as shown below. The input values are given in the diagram. The 4 inputs has initial weights -5, 8, 2, and 9. Assume the weights from each input units to each next layer unit being same and also weights from each of the hidden units to output layer is 3. Calculate the output of the NN keeping the activation function as sigmoid.



Answer:

Output at H1 = φ [(14x-5) + (6x8) + (9x2) + (18x9)] = φ [168] = 1

Similarly output at H2 = 1

Therefore, output of the neural net = φ [(1x-3) + (1x-3)] = φ [-6] = -1

Copyright ACDSD, CSIR-NEIST

Worked Examples

Solution:

output at $H_1 = f[(4x-5)+(6x8)+(9x2)+(18x9)]$

$= f(168)$

$= 1$

Similarly, output at $H_2 = 1$

So, output of the NN $= f[1x-3]+[1x-3]$

$= f(-6)$

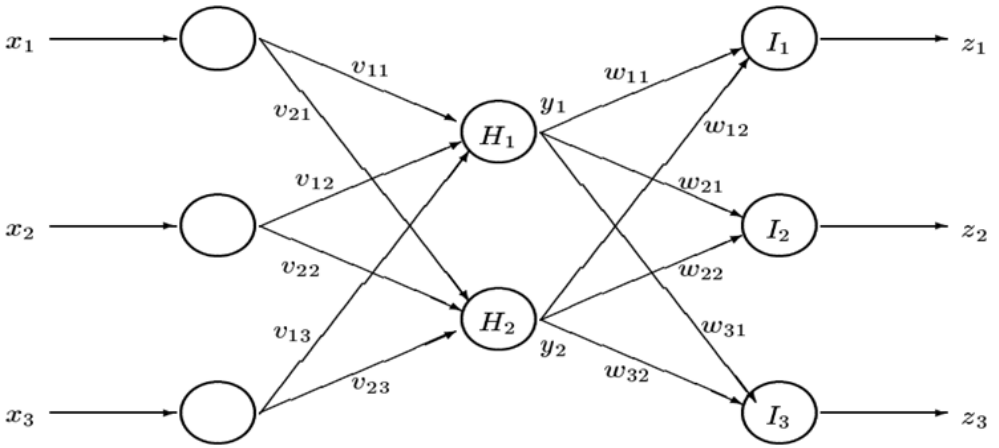
$= -1$

[f is sigmoid function]

So the output of NN is -1.

Copyright ACDSD, CSIR-NEIST

Question 16: A training pattern, consisting of an input vector X=[x1, x2, x3]^T and Z=[z1, z2, z3]^T, is presented to the following neural network.



Consider the weights vectors as

$$\mathbf{v}_1 = \begin{bmatrix} -2.0 \\ 2.0 \\ -2.0 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1.0 \\ 1.0 \\ -1.0 \end{bmatrix}, \quad \mathbf{w}_1 = \begin{bmatrix} 1.0 \\ -3.5 \end{bmatrix}, \quad \mathbf{w}_2 = \begin{bmatrix} 0.5 \\ -1.2 \end{bmatrix} \quad \text{and} \quad \mathbf{w}_3 = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}$$

If the network is tested with input vector X=[2, 3, 1]^T , Find out the output of the second hidden unit?

Answer: 4

H2 = [(2x1) + (3x1) + (1x-1)] = 4

Copyright ACDSD, CSIR-NEIST

Question 17: Using the notations used in class and the tutorial document, evaluate the value of the neural network with a 3-3-1 architecture (2-dimensional input with 1 node for the bias term in both the layers). The parameters are as follows –

$$\alpha = \begin{bmatrix} 1 & 0.2 & 0.4 \\ -1 & 0.3 & 0.5 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 0.3 & 0.4 & 0.5 \end{bmatrix}$$

Using linear function as the activation functions at both the layers and bias as +1 calculate the output of the network for an input of (0.8, 0.7) will be?

Answer:

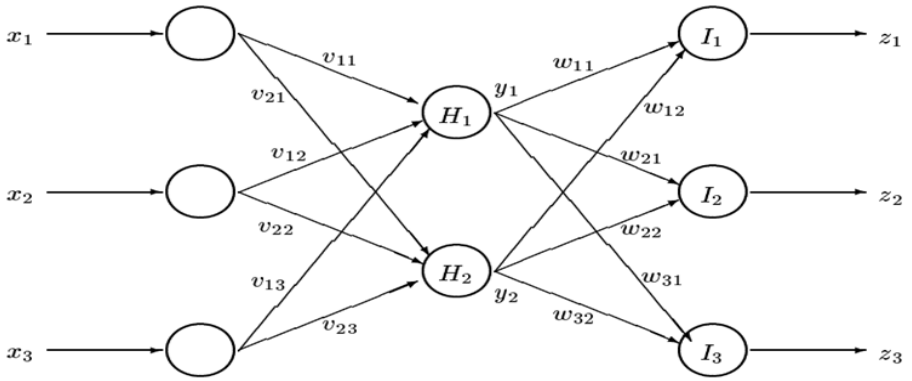
$$X = \begin{matrix} 1 \\ .8 \\ .7 \end{matrix}$$

Output of first layer can be written as $A = \alpha X$

Output of seond layer can be written as $B = \beta A$

Copyright ACDSD, CSIR-NEIST

Question 18: Consider the following neural network



a) What is the usual sequence of for training the network using the backpropagation algorithm?

b) In the Back-Propagation learning algorithm, what is the object of the learning? Does the Back-Propagation learning algorithm guarantee to find the global optimum solution?

Answer:

a) Calculate $z_k = f(I_k)$, update w_{kj} , calculate $y_i = f(H_j)$, update v_{ji}

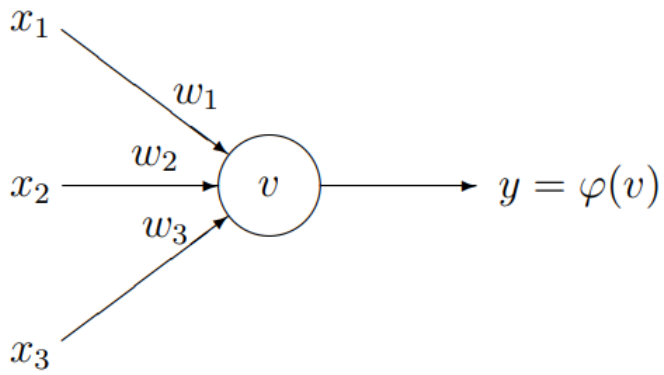
b) The Object is to learn the weights of the interconnections between the inputs and the hidden units and between the hidden units and the output units. The algorithms attempts to minimize the squared error between the network output values and the target values of these outputs. The learning algorithm does not guarantee to find the global optimum solution. It guarantees to find at least a local minimum of the error function.

Question 19: When the input is 2-dimensional, you can plot the decision boundary of your neural network and clearly see if there is overfitting. How do you check overfitting if the input is 10-dimensional?

Answer: Compute cost function in the val and training set. If there is a significant difference, then you have a variance problem.

Copyright ACDSD, CSIR-NEIST

Question 20: Consider a neural network as shown below



Suppose the weights corresponding to the three inputs are as follows -

w1 = 2

w2 = –4

w3 = 1

and the activation of the unit is given by the step-function: $\phi(v) = 1$ if $v \geq 0$ or 0 otherwise

Calculate what will be the output value y of the NN for each of the following 4 inputs:

1 0 0

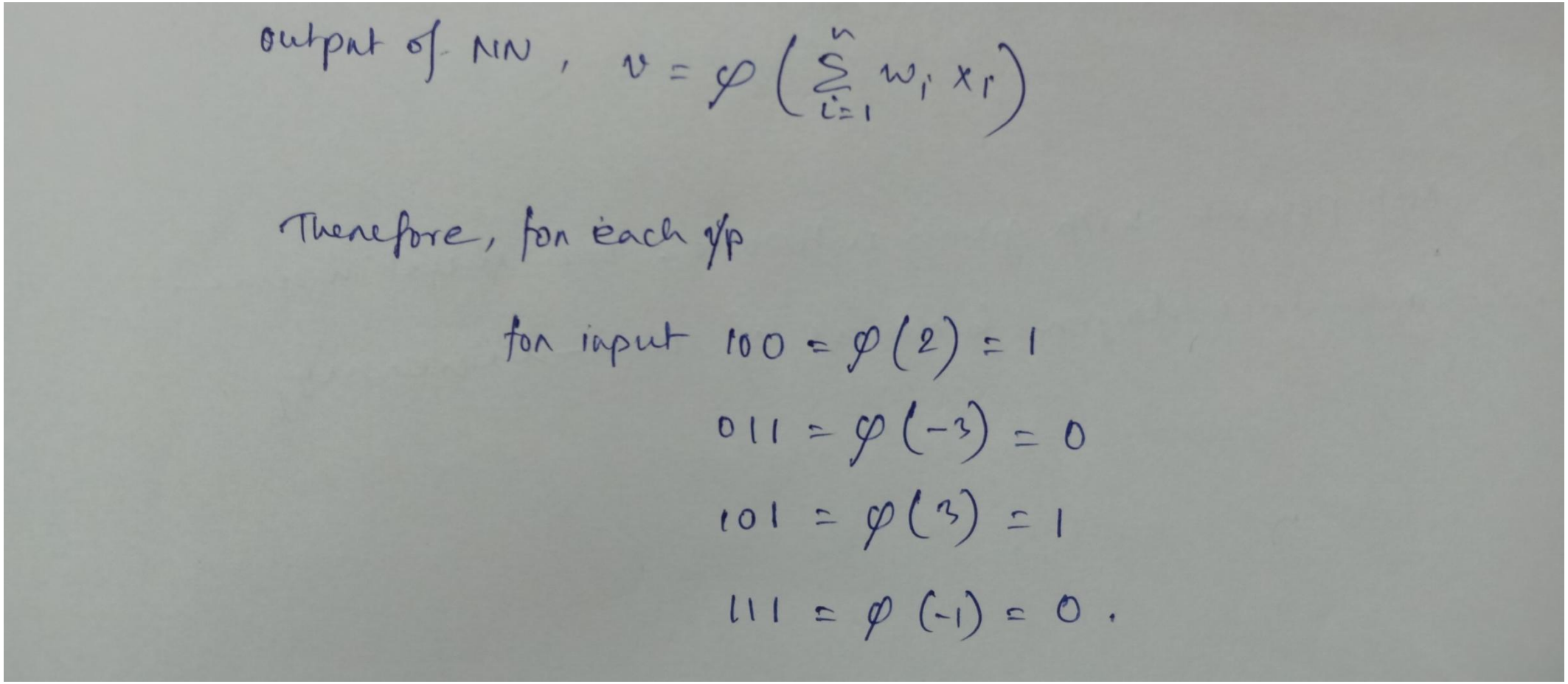
0 1 1

1 0 1

1 1 1

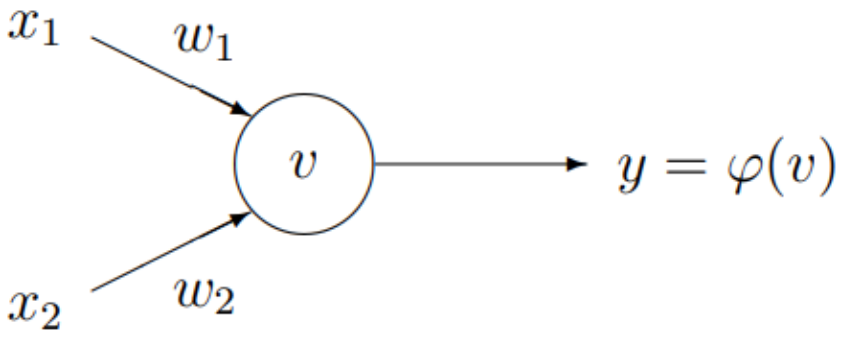
Copyright ACDSD, CSIR-NEIST

Solution:



Copyright ACDSD, CSIR-NEIST

Question 21: Consider a neural network as shown below



If the weights are w1 = 1 and w2 = 1

and the activation function is: $\phi(v) = 1$ if $v \geq 2$ or 0 otherwise

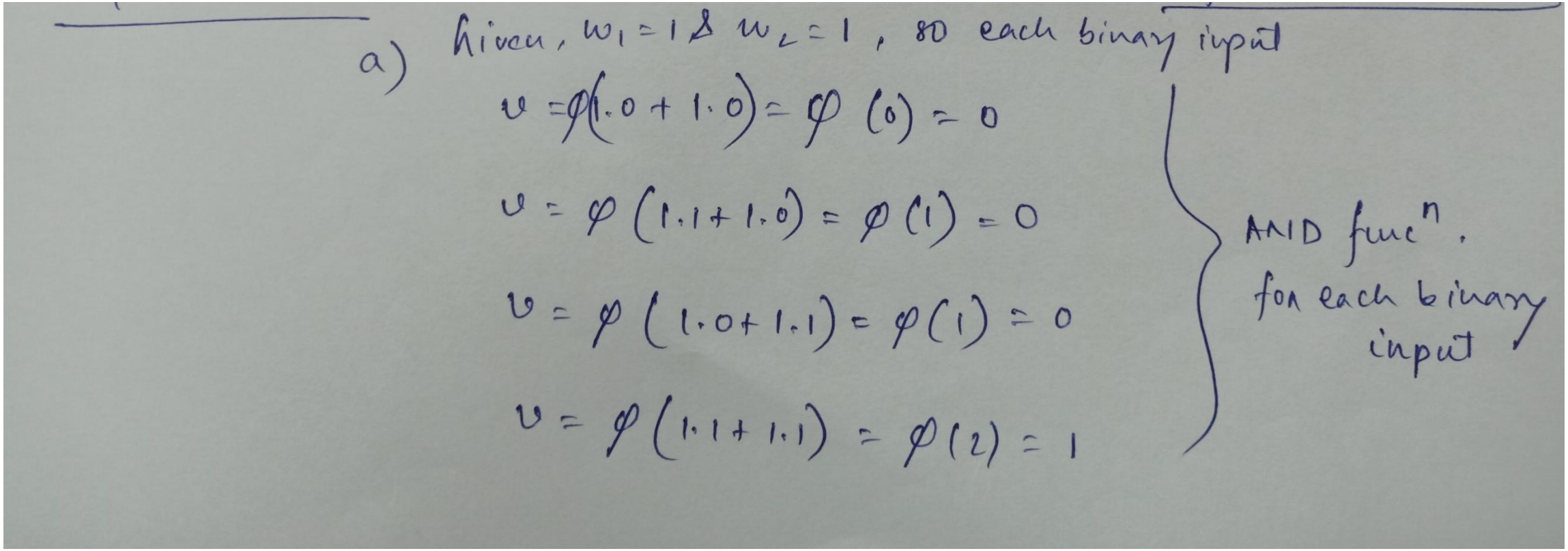
a) Test how the logical AND function works.

b) What will be the necessary changes in the weights to implement logical OR function.

c) Is it possible to do implement with above neural net? If not, why? Provide a solution to do this by bring some necessary changes in the architecture? Also mention why and how the modified architecture works perfectly?

Copyright ACDSD, CSIR-NEIST

Worked Examples



Copyright ACDSD, CSIR-NEIST

b) Try & guessing method, (Substitution)

Consider $w_1 = 2$, $w_2 = 2$

Then

$v = \varphi(2.0 + 2.0) = \varphi(0) = 0$

$v = \varphi(2.1 + 2.0) = \varphi(2) = 1$

$v = \varphi(2.0 + 2.1) = \varphi(2) = 1$

$v = \varphi(2.1 + 2.1) = \varphi(4) = 1$

} OR func

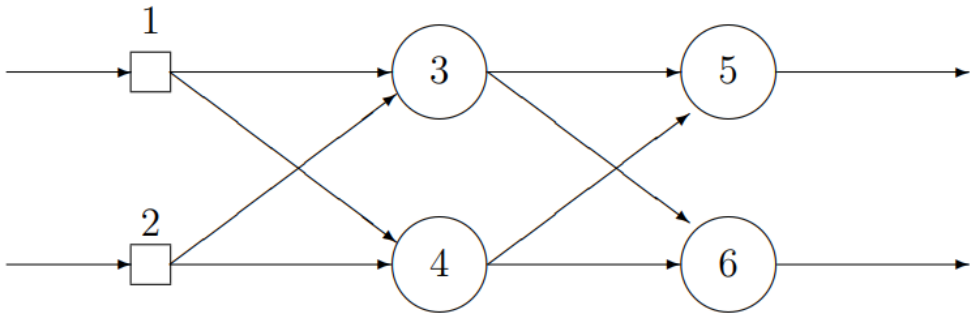
So modified weights as $w_1 = 2$ & $w_2 = 2$.

Copyright ACDSD, CSIR-NEIST

c) Not possible with given network. We must have a hidden layer because of non-linearity.

Copyright ACDSD, CSIR-NEIST

Question 22: The following is a feed-forward neural network with one hidden layer.



A weight on connection between nodes i and j is denoted by w_{ij}.

The following table lists all the weights in the network:

w13 = −2	w35 = 1	w23 = 3	w45 = −1
w14 = 4	w36 = −1	w24 = −1	w46 = 1

Each of the nodes 3, 4, 5 and 6 uses the activation function: $\phi(v) = 1$ if $v \geq 0$ or 0 otherwise

where v denotes the weighted sum of a node.

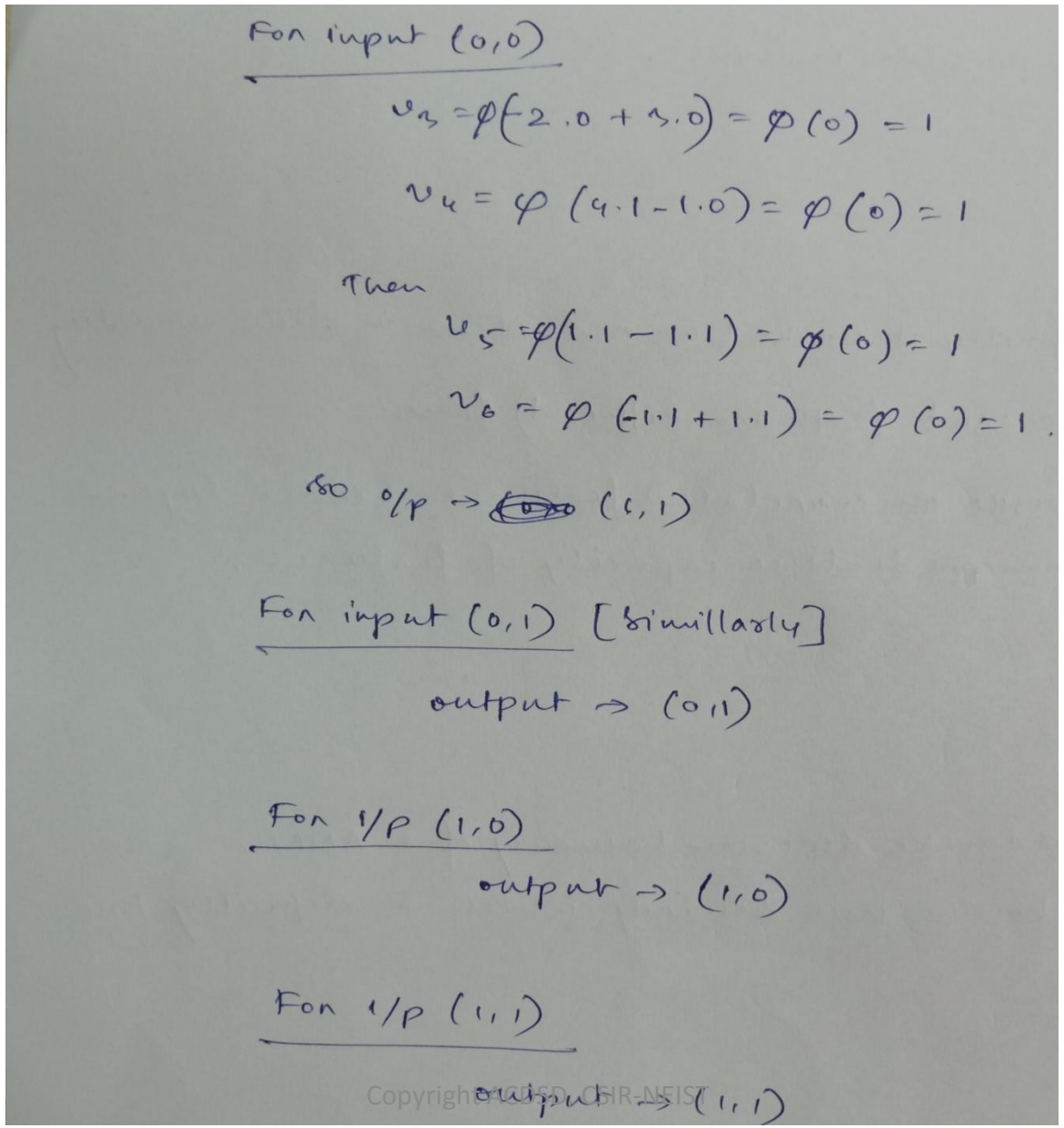
Calculate the output of the network (y5 and y6) for each of the input patterns:

0 0, 0 1, 1 0, and 1 1

Copyright ACDSD, CSIR-NEIST

Worked Examples

Solution:



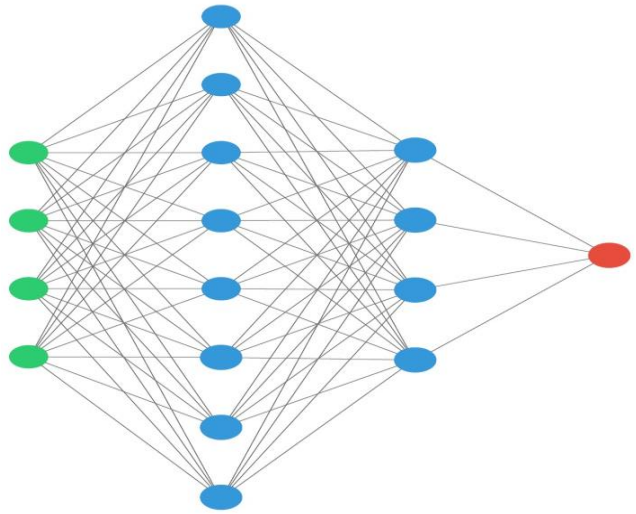
Copyright ACDS, CSIR-NEIST

Question 23: Assume you have defined a neural net architecture as shown below, and train it on a dataset of 1 million instances with 15 features and 4 classes in order to predict a new sample class. Answer the following –

a) Can we treat the hidden layers in the neural network as dimensionality reduction process.

b) If yes, can we use dimensionality reduction technique such as PCA instead of using hidden layer.

c) Would the network that uses a dimensionality reduction technique always give same output as network with hidden layer?



Copyright ACDSD, CSIR-NEIST

Solution:

a) Yes, definitely it works like reduction in dimensionality

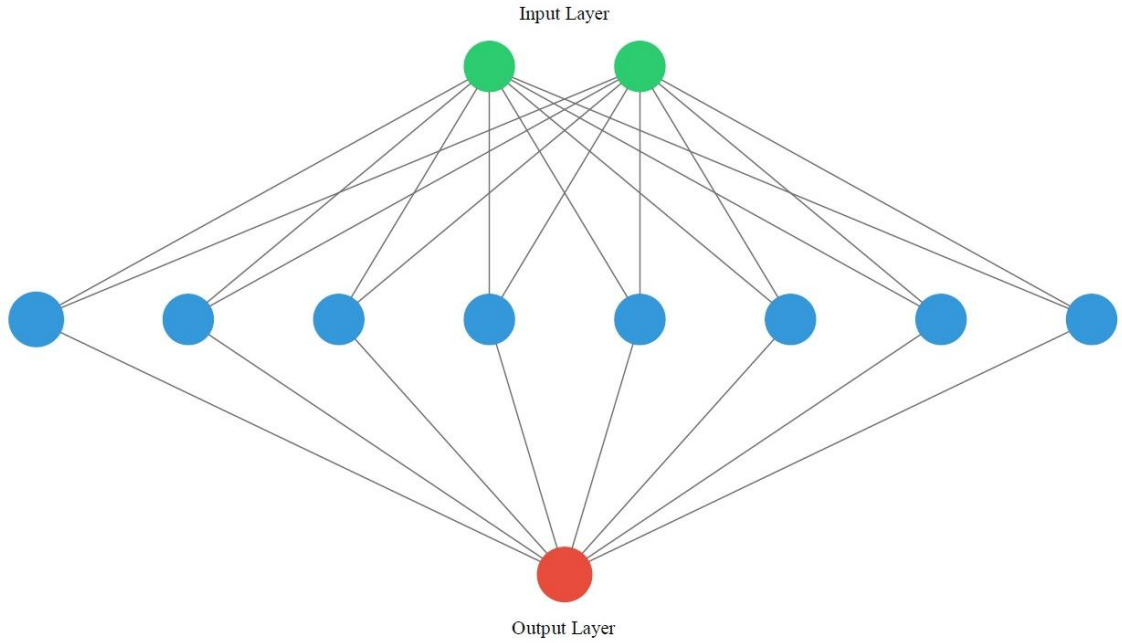
b) No, we can't (both are different)

c) PCA works on correlated features, hidden layer works on predictive capacity of features.

Copyright ACDSD, CSIR-NEIST

Question 24: For the following NN assume the following –

- All the inputs to the network are negative numbers.
- The weights assigned from inputs units are all positives.
- Activation function used in each neurons is Relu function.



Answer the following -

a) What will be the strange thing you will observe in the neural network in the training process?

b) How will you overcome the problem?

Copyright ACDSD, CSIR-NEIST

Solution:

a) A/Q, I/p → -ve

weights → +ve

so net o/p → always -ve

Now A/Q activation func → Relu

so output at each hidden layer becomes 0.

so all hidden neurons behave like dead neuron.

b) Solution is to use different activation func depending upon state of the problem.

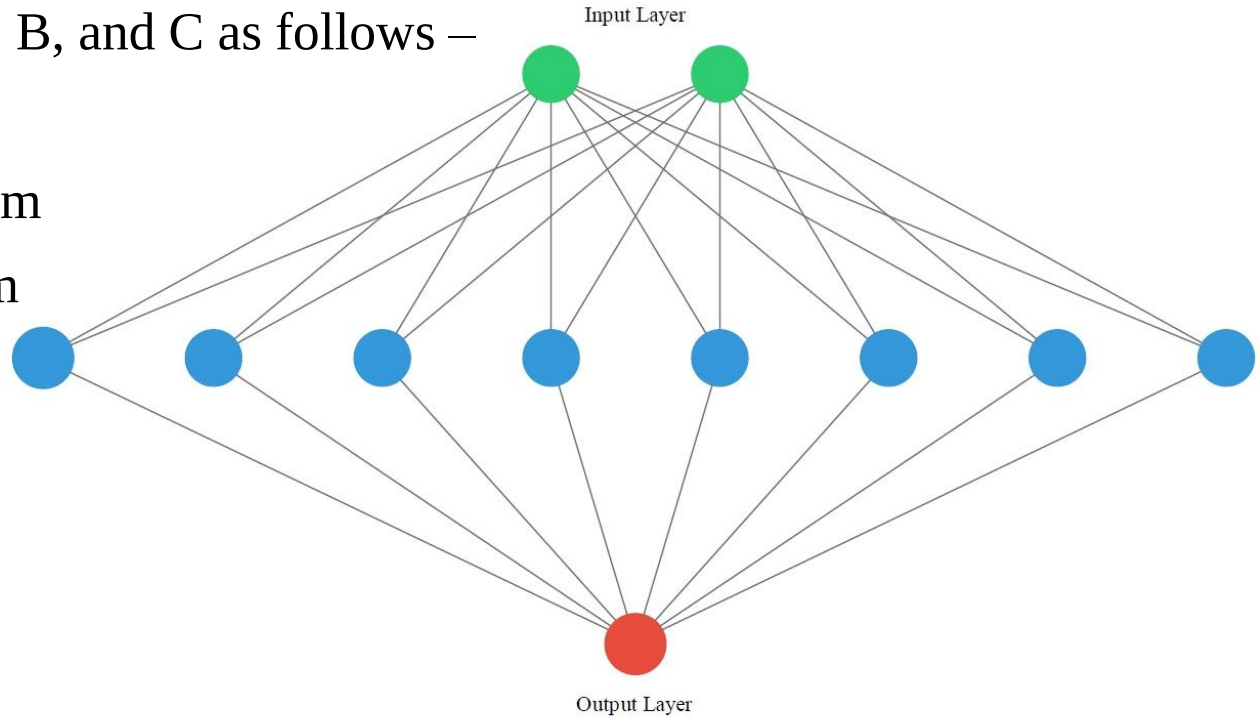
Copyright ACDSD, CSIR-NEIST

Question 25: Consider three problems A, B, and C as follows –

Problem A: A regression problem

Problem B: A binary classification problem

Problem C: A multiclass (3 class) problem



Answer the following -

a) Is it possible to address all the above problems with the given neural net? If not, why?

b) What will be the learning algorithm, loss function and activation function used for each of the problem?

c) What necessary changes need to make to address all the above problem by the given net?

Copyright ACDSD, CSIR-NEIST

Solution:

a) No its not possible,
because o/p unit given is 1.

b) Sigmoid will work on problems A & B but not C
For C we must use softmax at last layer

c) For A & B architecture may work.
But for C we have to increase o/p layer
units to 3.

Copyright ACDSD, CSIR-NEIST

Exercises

Question 1: Let's say you have a e-market website with a lot of traffic. You would like to build a network that computes the probability of a user clicking on a given advertisement on your website. What dataset do you need to train such a network?

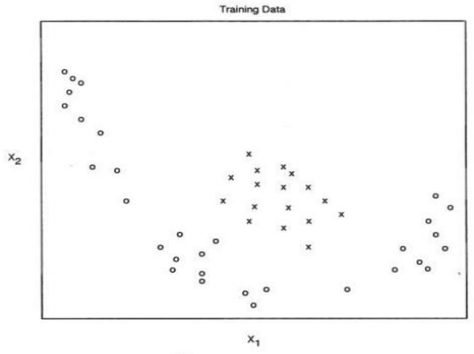
Question 2: You have already collected a dataset of 1 million examples from past visits. Your friend, who works in another company offers to let you use their dataset as it has similar descriptors. However, you are concerned about the impact of this different distribution dataset in the performance of your predictive system. Explain how you would use your friend's dataset.

Question 3: Answer the following –

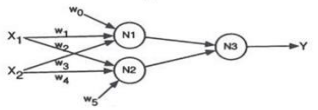
a) Why do the layers in a deep architecture need to be non-linear? Can the same be achieved through multiple linear layer?

b) Why is it impossible for a single binary perceptron to solve the XOR problem?

Question 4: Consider the following classification training data and the NN that classify the data correctly by setting proper weights.



Plot the decision boundaries of N1 and N2?



Copyright ACDSD, CSIR-NEIST

Question 5: The universal approximation theorem states that a neural network with a single hidden layer can approximate any continuous function (with some assumptions on the activation). Give one reason why you would use deep networks with multiple layers.

Question 6: You have defined a neural net architecture to train it on a dataset of 1 million instances with 15 features and 4 classes in order to predict a new sample class. Answer the following –

a) Can we treat the hidden layers in the neural network as dimensionality reduction process.

b) If yes, can we use dimensionality reduction technique such as PCA instead of using hidden layer.

c) Would the network that uses a dimensionality reduction technique always give same output as network with hidden layer?

Question 7: A student wanted to develop an autonomous system to predict whether a patient has a disease or not out of diseases A, B, C, D. He collected a set of historical data from a local hospital from patients who have visited in the last 10 years. He performs feature selection and extracted a set of features for each patient. Note that a patient may have one or more of these diseases among A, B, C, D. The student has decided to use a neural network to solve this problem among all. Which one of the following strategy he should follow and why –

a) Train a separate neural network for each of the diseases.

b) Train a single NN with one output neuron for each disease, but with a shared hidden layer.

Copyright ACDSD, CSIR-NEIST

Question 8: Taking an example of two layer perceptron perform the following -

a) Design a two-input perceptron that implements the Boolean function A∧¬B.

b) Design the two-layer network of perceptron that implements A XOR B.

Question 9: You're solving a binary classification task.

a) You first try a logistic regression. You initialize all weights to 0.5. Is this a good idea? Briefly explain why or why not.

b) Then, you try a 4-layer neural network. You initialize all weights to 0.5. Is this a good idea? Briefly explain why or why not.

Question 10: Considering the sigmoid activation function answer the following:

a) What would the gradient of the sigmoid be with respect to an input that is very large?

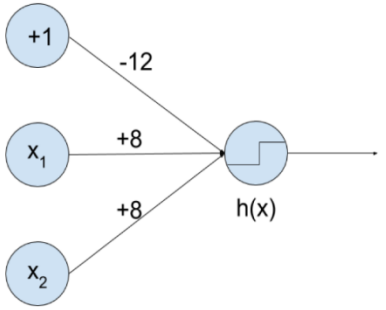
b) Why might this be a problem for training neural networks?

c) How does the ReLU activation ReLU(z) = max(0; z) solve this problem?

Copyright ACDSD, CSIR-NEIST

Question 11: You are given the following neural networks which take two binary valued inputs x1, x2 as either 0 or 1 and the activation function is the threshold function (h(x) = 1 if x > 0; 0 otherwise).

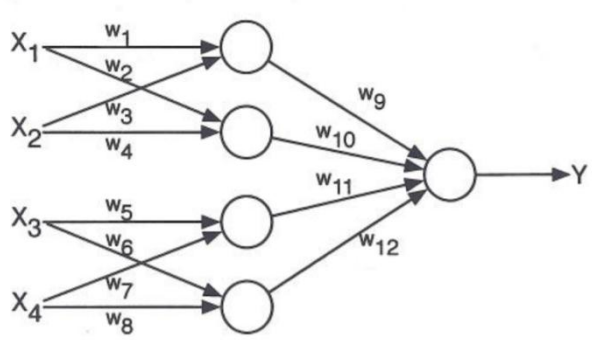
Compute the output for all the possible input?



Question 12: Following neural network uses linear activation function. The output of each unit is a constant multiplied by the net output.

a) Can you realize the same function implemented by above neural network with single layer neural network? If yes, how and if no, why?

b) Can the above NN be represented by regression? If yes, provide the equation?



Copyright ACDSD, CSIR-NEIST

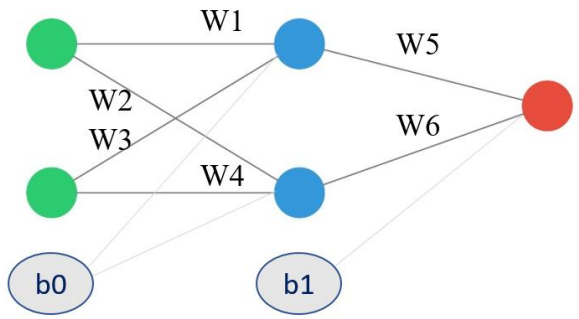
Question 13: We have a function which takes a two-dimensional input x = (x1, x2) and has two parameters w = (w1,w2) given by f(x,w) = σ(σ(x1w1)w2 + x2) where σ(x) is the sigmoid function. We use backpropagation to estimate the right parameter values. We start by setting both the parameters to 0. Assume that we are given a training point x1 = 1; x2 = 0; y = 5. Given this information answer the next two questions.

a) What is the value of $\frac{\delta f}{\delta w2}$?

b) If the learning rate is 0.5, what will be the value of w2 after one update using backpropagation algorithm?

Copyright ACDSD, CSIR-NEIST

Question 14: Consider the following 2-2-1 Feed Forward neural network –



Given inputs 0.05 and 0.10

Expected outputs 0.01 and 0.99

Activation function used - sigmoid

Perform the following -

a) Forward propagation – find out the actual output?

b) Calculate total error after forward propagation?

c) Perform backward pass and update the weights and find the new error (after first iteration)?

Copyright ACDSD, CSIR-NEIST

Question 15: You are searching the best learning rate for your model. You decide to test the following values between 0.01 and 1:

– learning rate = 0.01

– learning rate = 0.16

– learning rate = 0.21

– learning rate = 0.84

– learning rate = 0.94

Is that a good method? Explain why.

Question 16: You have two data sets of similar size for a binary classification task. However, one contains almost entirely positive examples, and the other contains only negative examples. You would like to use both sets to train your model. Describe a scenario in which combining these two data sets could lead to a failure of the model to learn?

Copyright ACDSD, CSIR-NEIST

Question 17: You want to solve a classification task. You first train your network on 20 samples. Training converges, but the training loss is very high. You then decide to train this network on 10,000 examples. Is your approach to fixing the problem correct? If yes, explain the most likely results of training with 10,000 examples. If not, give a solution to this problem.

Question 18: Consider the problem you are trying to solve has a small amount of data. Fortunately, you have a pre-trained neural network that was trained on a similar problem. What will be your approach to tackle the situation?

Question 19: After training a neural network, you observe a large gap between the training accuracy (100%) and the test accuracy (42%). Which of the following methods is commonly used to reduce this gap in ANN?

Question 20: You are asked to design neural network system for the following problems –

a) To diagnose breast cancer through historical data.

b) To detect brain tumor

What is the most appropriate evaluation metric among Accuracy, Precision, Recall that you will use in evaluating performance of the system for above problems. Explain your choice.

Note: Precision = True positive examples / Total predicted positive examples

Recall = True positive examples/Total positive examples

Copyright ACDSD, CSIR-NEIST

11.6 References

The Topics Covered in This Section

11.6.1 Books
11.6.2 Video Lectures

Copyright ACDSD, CSIR-NEIST

11.6.1 Books

- Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

➢ *This introduces basic concept of feed-forward neural networks from the perspective of statistical pattern recognition. The book examines techniques for modeling probability density functions and the properties and merits of the multi-layer perceptron and radial basis function network models.* ★★★★

- Neilson, M. (2015) *Neural Networks and Deep Learning*.

➢ *The book teaches about Neural networks, a beautiful biologically-inspired programming paradigm which enables a computer to learn from observational data and deep learning, a powerful set of techniques for learning in neural networks* ★★★★

- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media.

➢ *By using concrete examples, minimal theory, and two production-ready Python frameworks—Scikit-Learn and TensorFlow, this book provides an intuitive understanding of the concepts and tools for building intelligent systems.A range of techniques, starting with simple linear regression and progressing to deep neural networks have been covered here.* ★★★

Copyright ACDSD, CSIR-NEIST

- Mehotra, K., Mohan, C. K., and Ranka, S. (1997). *Elements of Artificial Neural Networks*. MIT Press.

➢ *This book provides a organized general introduction, focusing on a broad range of algorithms to use neural networks. Along with basic concepts, the book provides frequently used algorithm like back propagation, perceptron etc.* ★★★★

- Fausett, L. (1994). *Fundamentals of Neural Networks*. Prentice Hall.

➢ *Basic concepts of Neural Network, how they are used for pattern classification, pattern association along with advanced concepts like adaptive resonance theory and back propagation are covered in this book.* ★★★

Copyright ACDSD, CSIR-NEIST

11.6.2 Video Lectures

- Neural Networks for Machine Learning | Prof. Jeoffrey Hinton

https://www.youtube.com/watch?v=2fRnHVVLf1Y&list=PLiPvV5TNogxKKwvKb1RKwkq2hm7ZvpHz0

- Neural Networks and Applications | Prof. S. Sengupta

https://www.youtube.com/watch?v=xbYgKoG4x2g&list=PL3EA65335EAC29EE8

Copyright ACDSD, CSIR-NEIST

Thank You

Copyright ACDSD, CSIR-NEIST