Introduction to Neural Network Algorithm

Yinghao Wu
Department of Systems and Computational Biology
Albert Einstein College of Medicine
Fall 2014

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

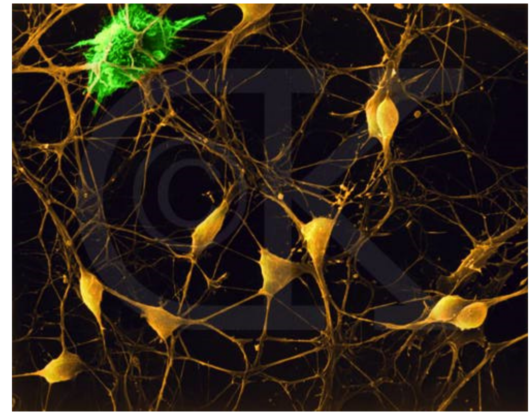
Biological Inspiration

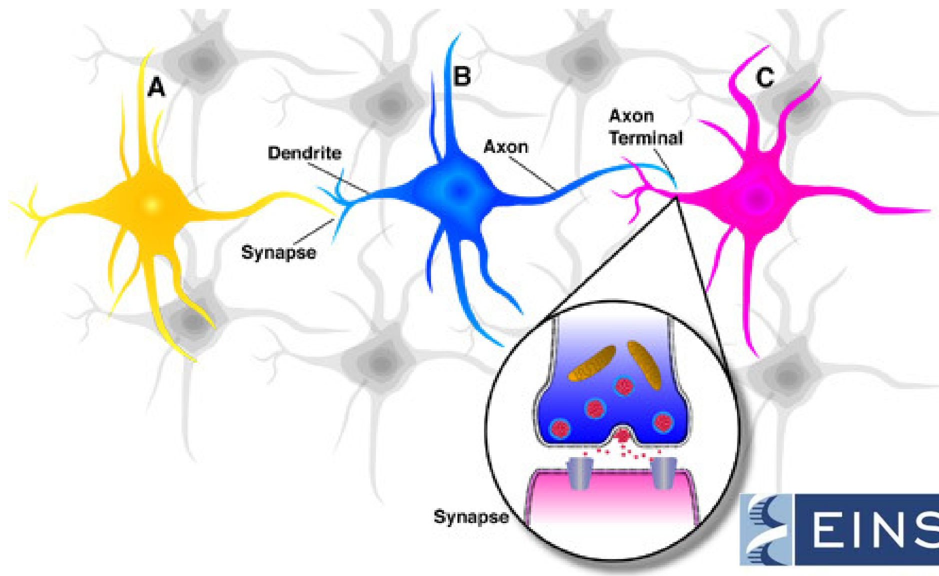
Idea : To make the computer more robust, intelligent, and learn, …
Let's model our computer software (and/or hardware) after the brain

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Neurons in the Brain

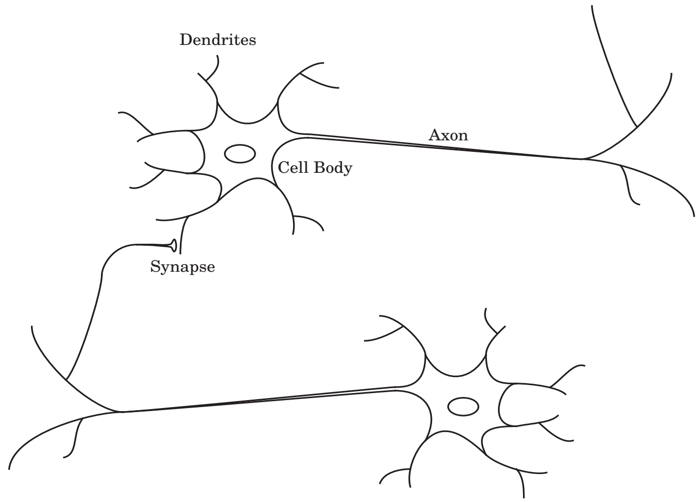
- Although heterogeneous, at a low level the brain is composed of neurons
  - A neuron receives input from other neurons (generally thousands) from its synapses
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that travels that travels from the body, down the axon, to the next neuron(s)

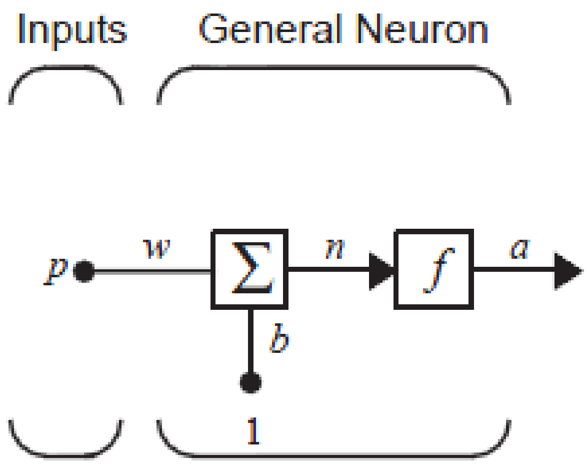




EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Neuron Model





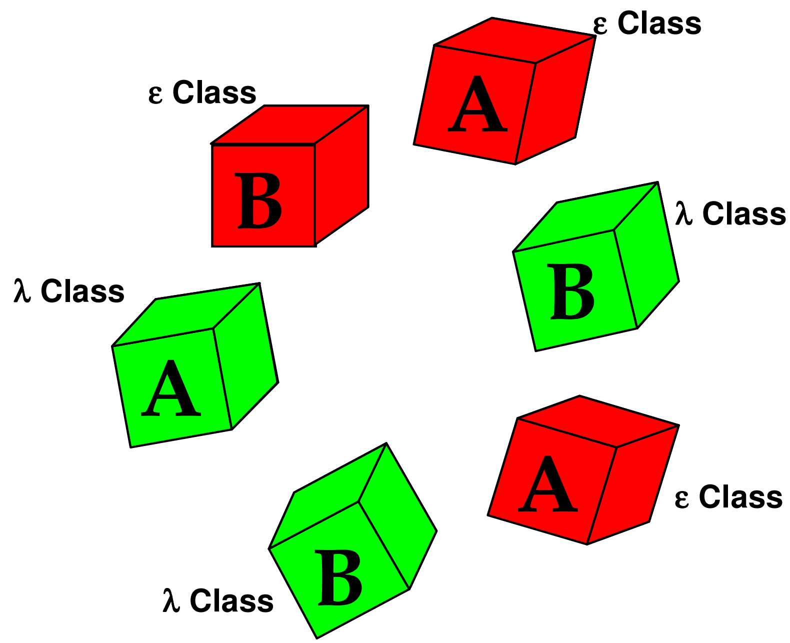
the weight “w” corresponds to the strength of a synapse

the cell body is represented by the summation and the transfer function

the neuron output “a”represents the signal on the axon

Supervised Learning

- It is based on a labeled training set.
- The class of each piece of data in training set is known.
- Class labels are pre-determined and provided in the training phase.



Albert Einstein College of Medicine of Yeshiva University

Unsupervised Learning

- Input : set of patterns P, from n-dimensional space S, but little/no information about their classification, evaluation, interesting features, etc.

  It must learn these by itself! :)

- Tasks:
  - Clustering - Group patterns based on similarity
  - Vector Quantization - Fully divide up S into a small set of regions (defined by codebook vectors) that also helps cluster P.
  - Feature Extraction - Reduce dimensionality of S by removing unimportant features (i.e. those that do not help in clustering P)

Supervised Vs Unsupervised

- Task performed
  - Classification
  - Pattern Recognition
- NN model :
  - Preceptron
  - Feed-forward NN

"What is the class of this data point?"

- Task performed
  - Clustering
- NN Model :
  - Self Organizing Maps

"What groupings exist in this data?"

"How is each data point related to the data set as a whole?"

Applications

- Aerospace
  - High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors
- Automotive
  - Automobile automatic guidance systems, warranty activity analyzers
- Banking
  - Check and other document readers, credit application evaluators
- Defense
  - Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
- Electronics
  - Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling

Applications

- Financial
  - Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction
- Manufacturing
  - Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process systems
- Medical
  - Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Neural Networks

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

Contd..

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

Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$
  2. An adder function (linear combiner) for computing the weighted sum of the inputs:

     (real numbers)

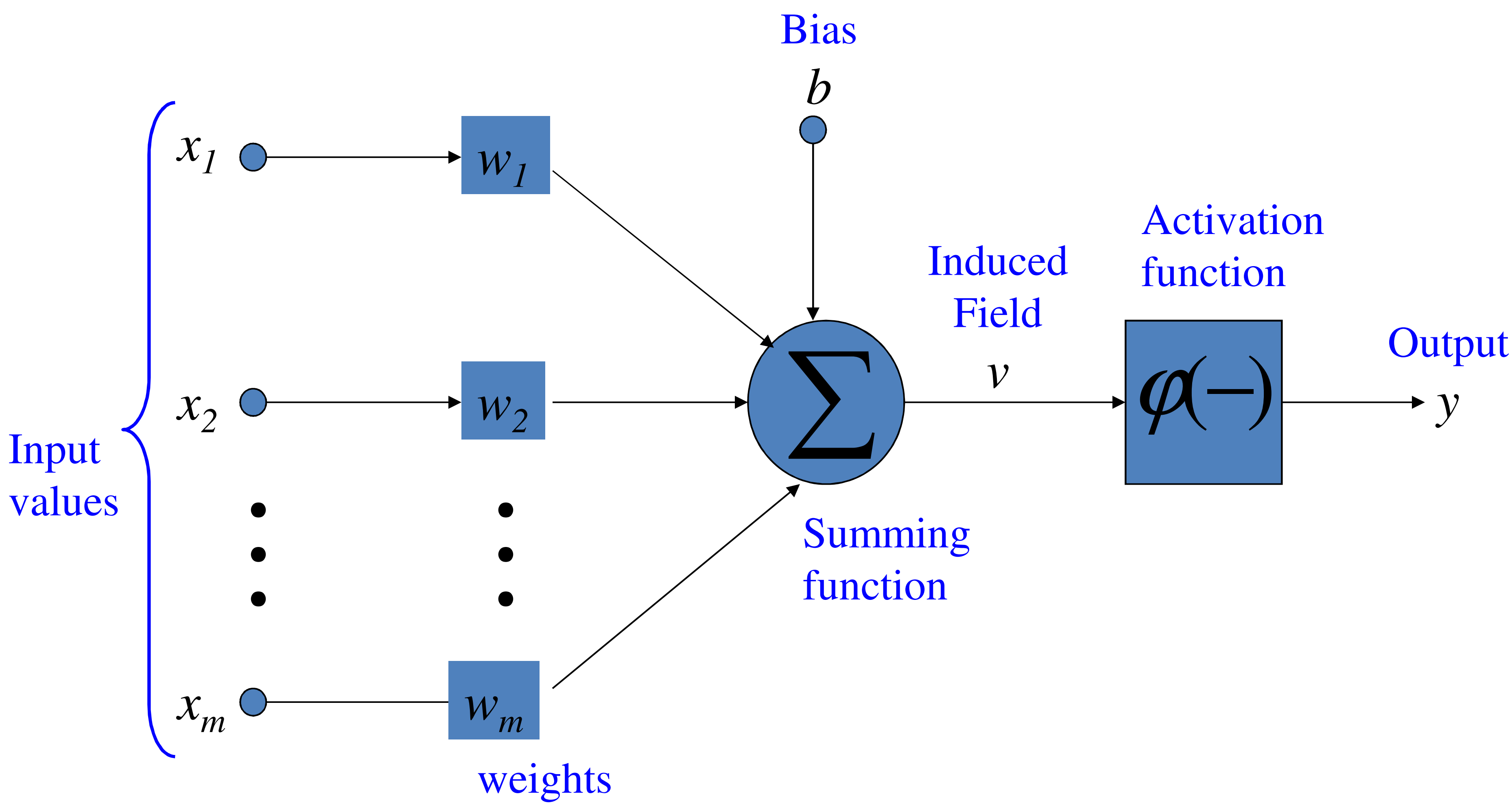
$$u = \sum_{j=1}^{m} w_j x_j$$

  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$y = \varphi(u + b)$$

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

The Neuron Diagram



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Neuron Models

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

Examples:

- step function:

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

- ramp function:

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v - c)(b - a)/(d - c)) & \text{otherwise} \end{cases}$$

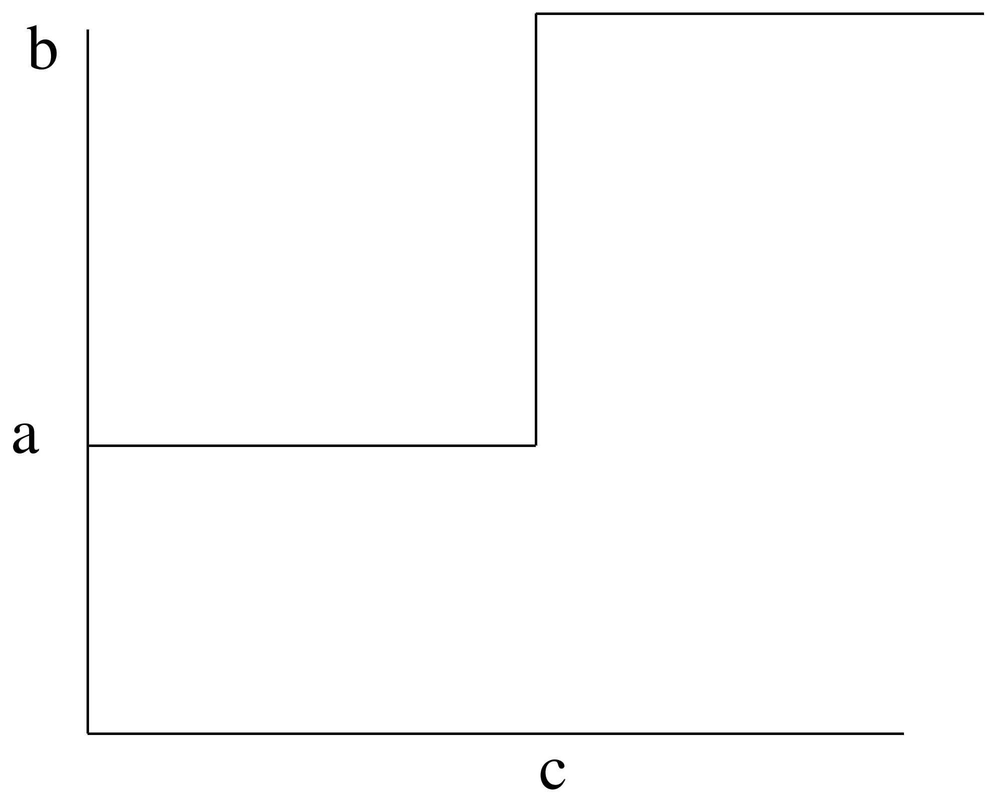
- sigmoid function with z,x,y parameters

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

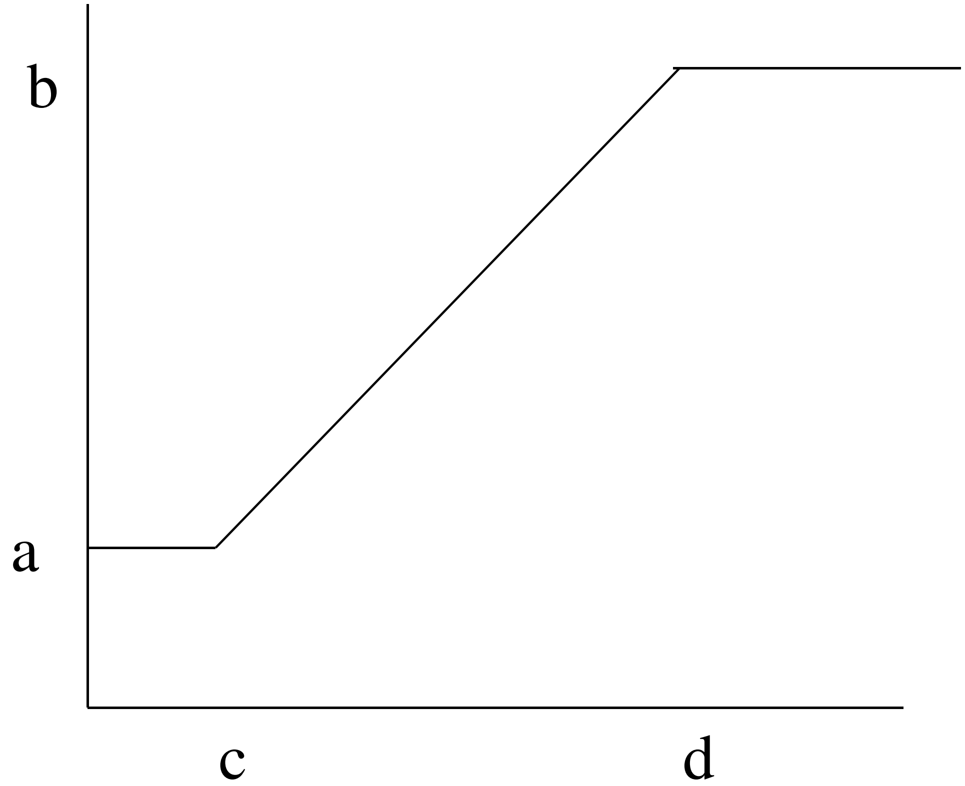
- Gaussian function:

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

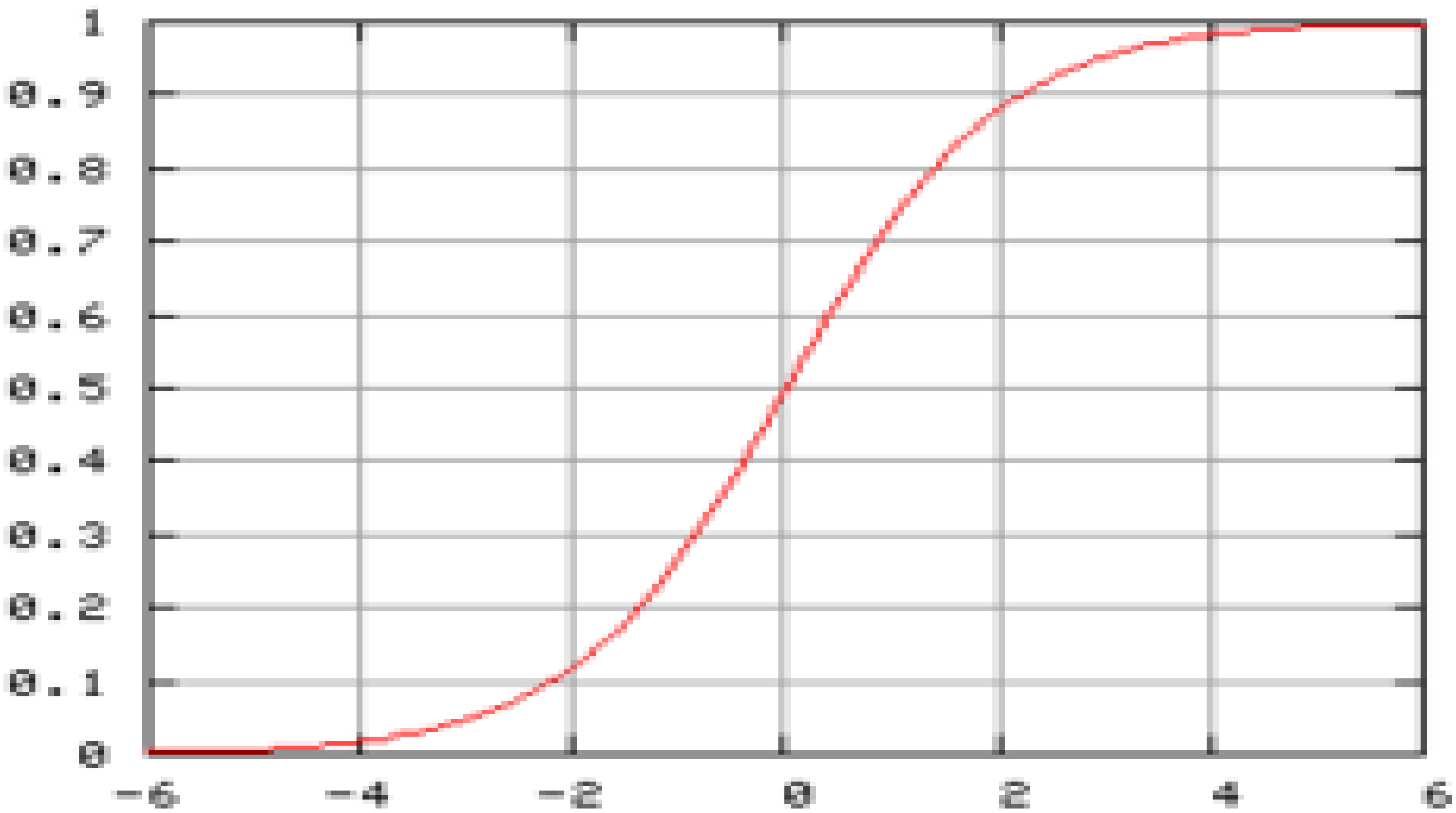
Step Function



Ramp Function



Sigmoid function



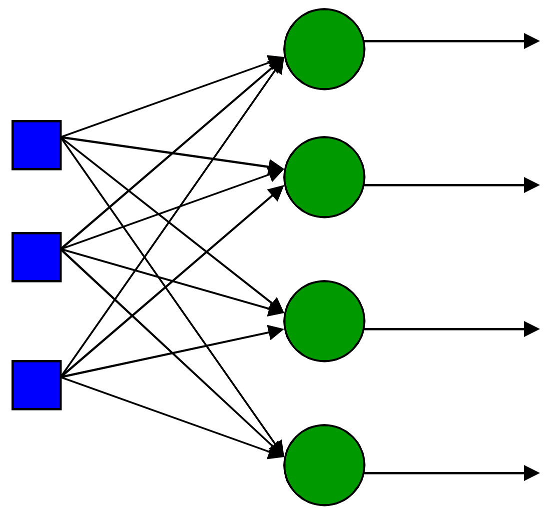
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer
of
source nodes



Output layer
of
neurons

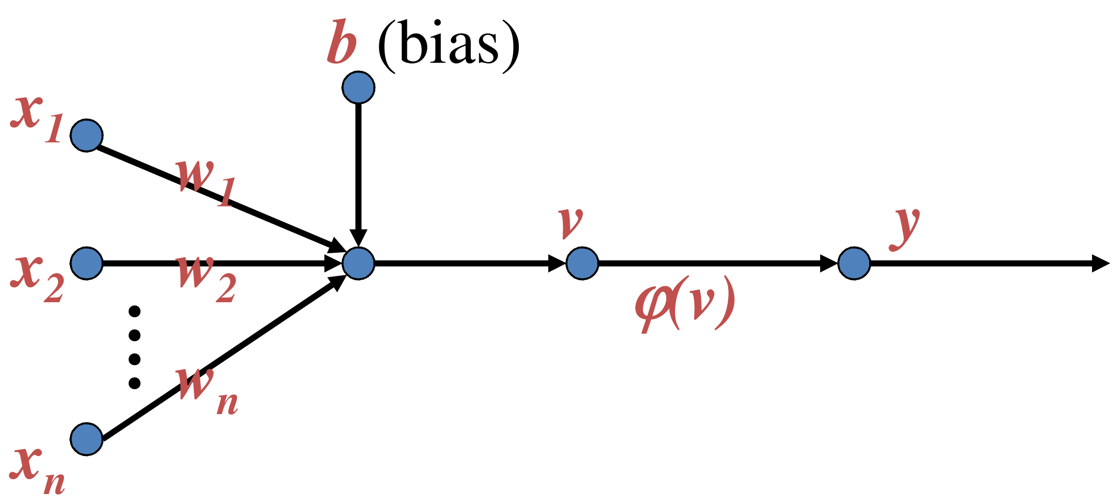
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Perceptron: Neuron Model

(Special form of single layer feed forward)

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

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



Perceptron for Classification

- The perceptron is used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- The perceptron can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.
- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class* C_1
  - *If the output is -1 then the input is assigned to* C_2

Learning Process for Perceptron

- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

  $wi \leftarrow wi + (a^* \ xi \ ^*e)$, where 'e' is error produced and 'a' ($-1 < a < 1$) is learning rate

  - 'a' is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.
  - Once the modification to weights has taken place, the next piece of training data is used in the same way.
  - Once all the training data have been applied, the process starts again until all the weights are correct and all errors are zero.
  - Each iteration of this process is known as an epoch.

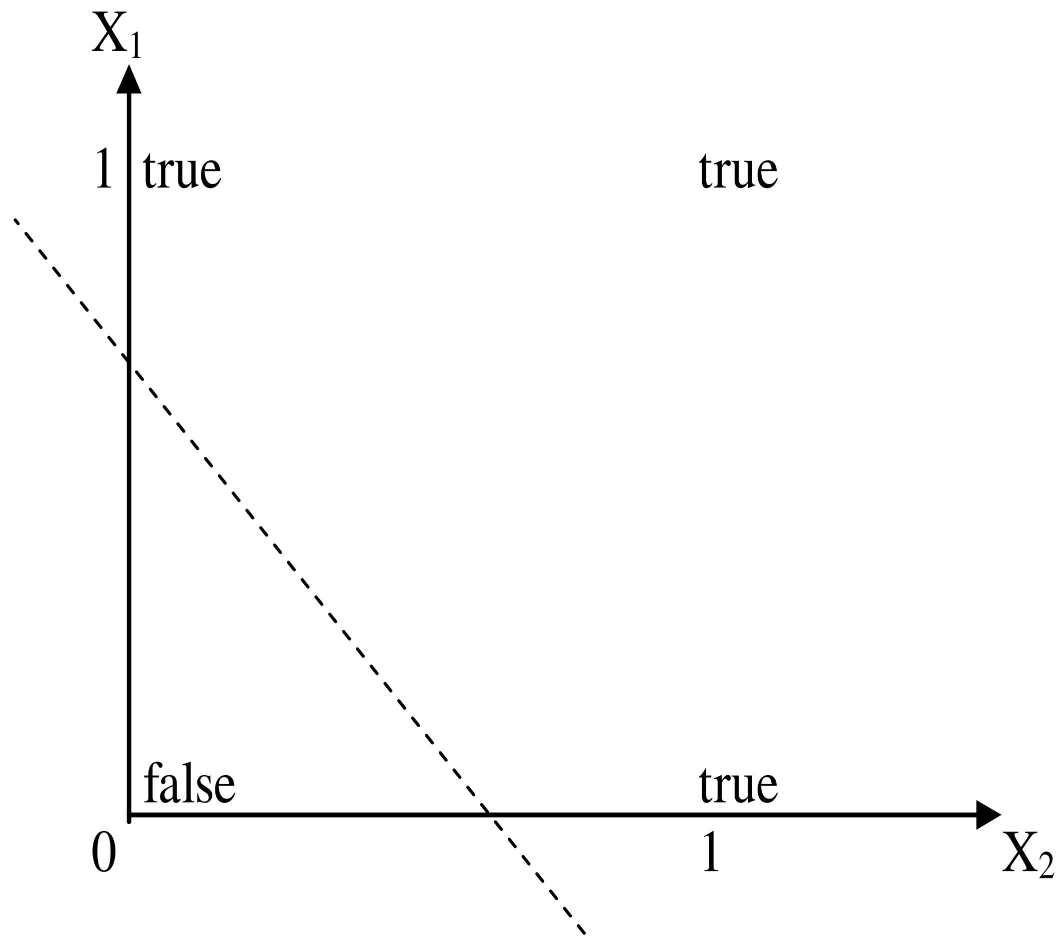
Example: Perceptron to learn OR function

- Initially consider w1 = -0.2 and w2 = 0.4
- Training data say, x1 = 0 and x2 = 0, output is 0.
- Compute y = Step(w1*x1 + w2*x2) = 0. Output is correct so weights are not changed.
- For training data x1=0 and x2 = 1, output is 1
- Compute y = Step(w1*x1 + w2*x2) = 0.4 = 1. Output is correct so weights are not changed.
- Next training data x1=1 and x2 = 0 and output is 1
- Compute y = Step(w1*x1 + w2*x2) = - 0.2 = 0. Output is incorrect, hence weights are to be changed.
- Assume a = 0.2 and error e=1

  wi = wi + (a * xi * e) gives w1 = 0 and w2 =0.4
- With these weights, test the remaining test data.
- Repeat the process till we get stable result.

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Boolean function OR – Linearly separable



Perceptron: Limitations

- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

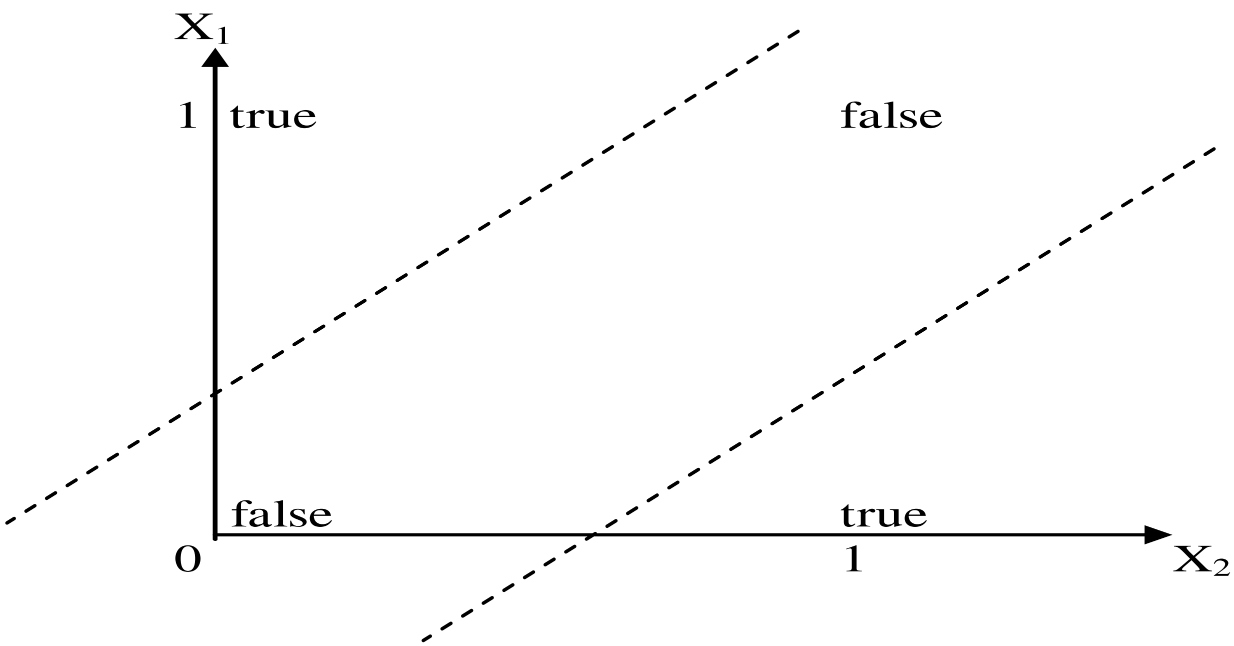
XOR – Non linearly separable function

- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or.**.
- This function takes two input arguments with values in {0,1} and returns one output in {0,1},
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

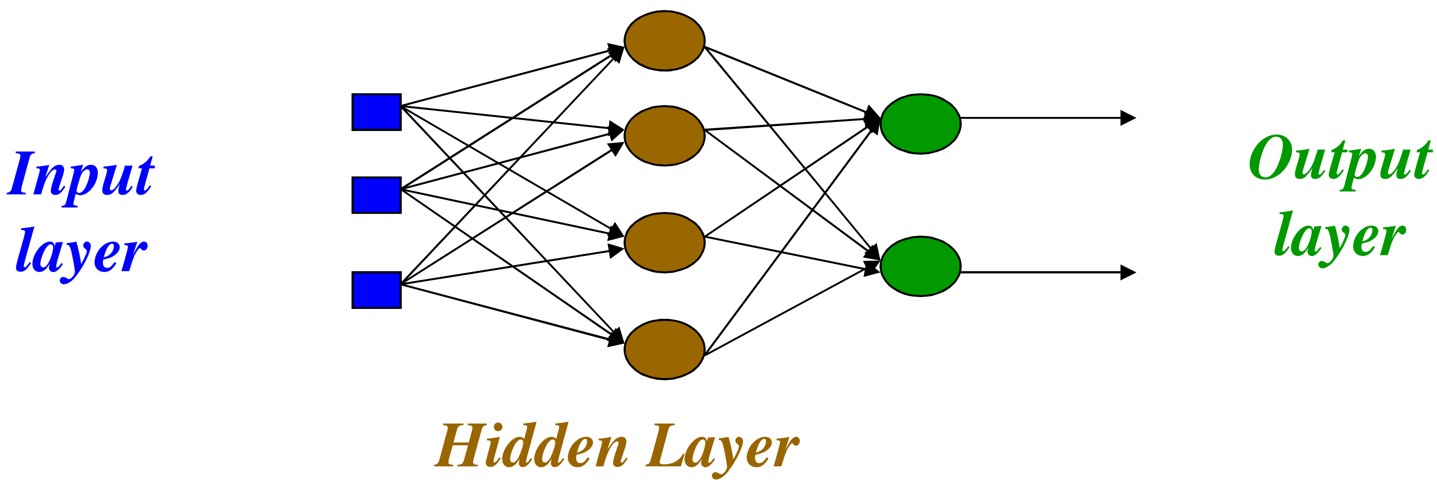
Input		Output
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.



Multi layer feed-forward NN (FFNN)

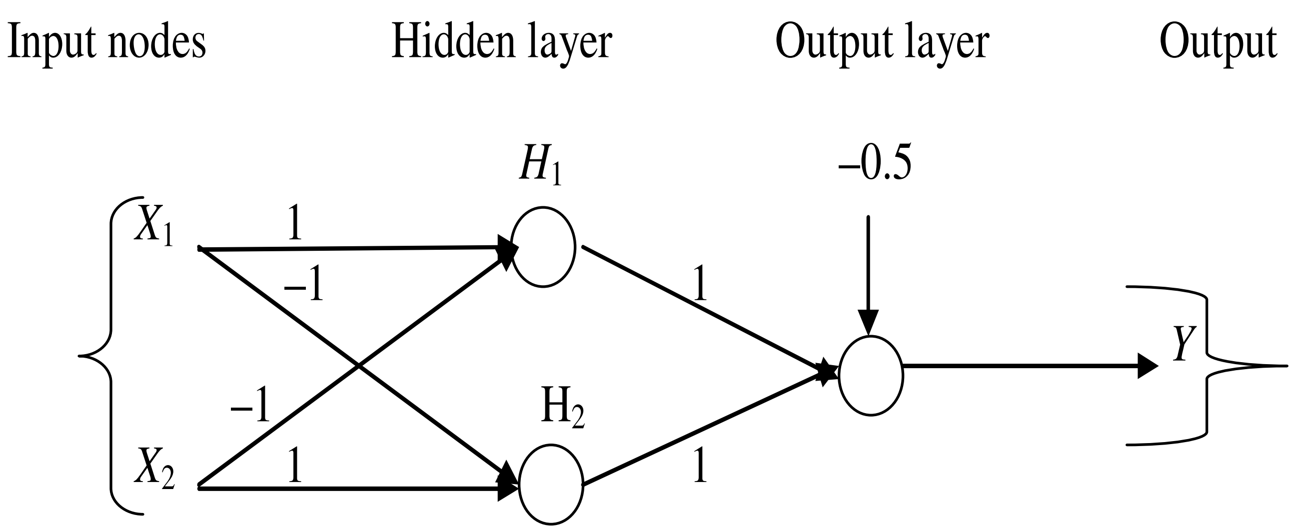
- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning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



3-4-2 Network

FFNN for XOR

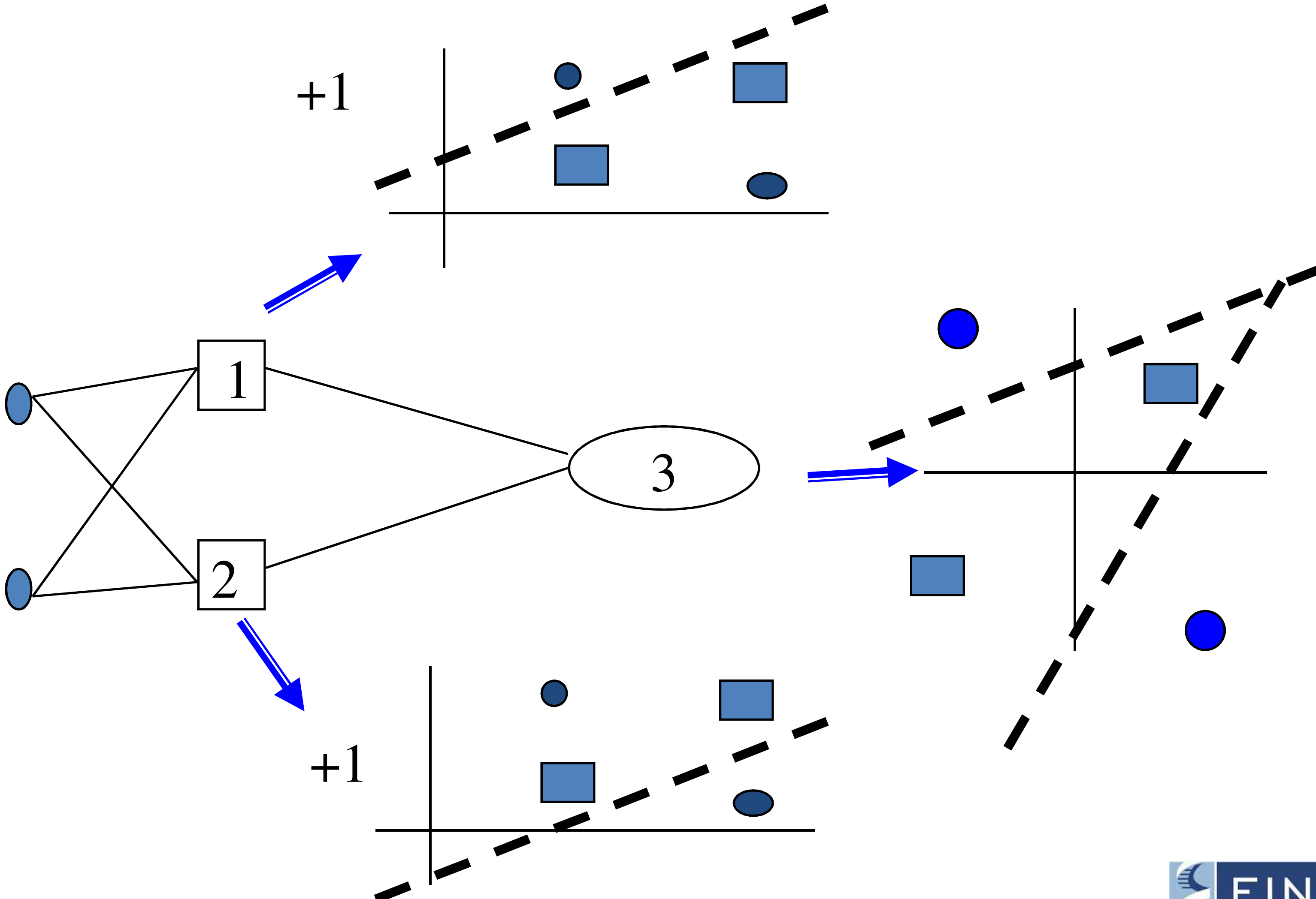
- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors (1,-1) and (-1,1).
- The output node is used to combine the outputs of the two hidden nodes.



Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

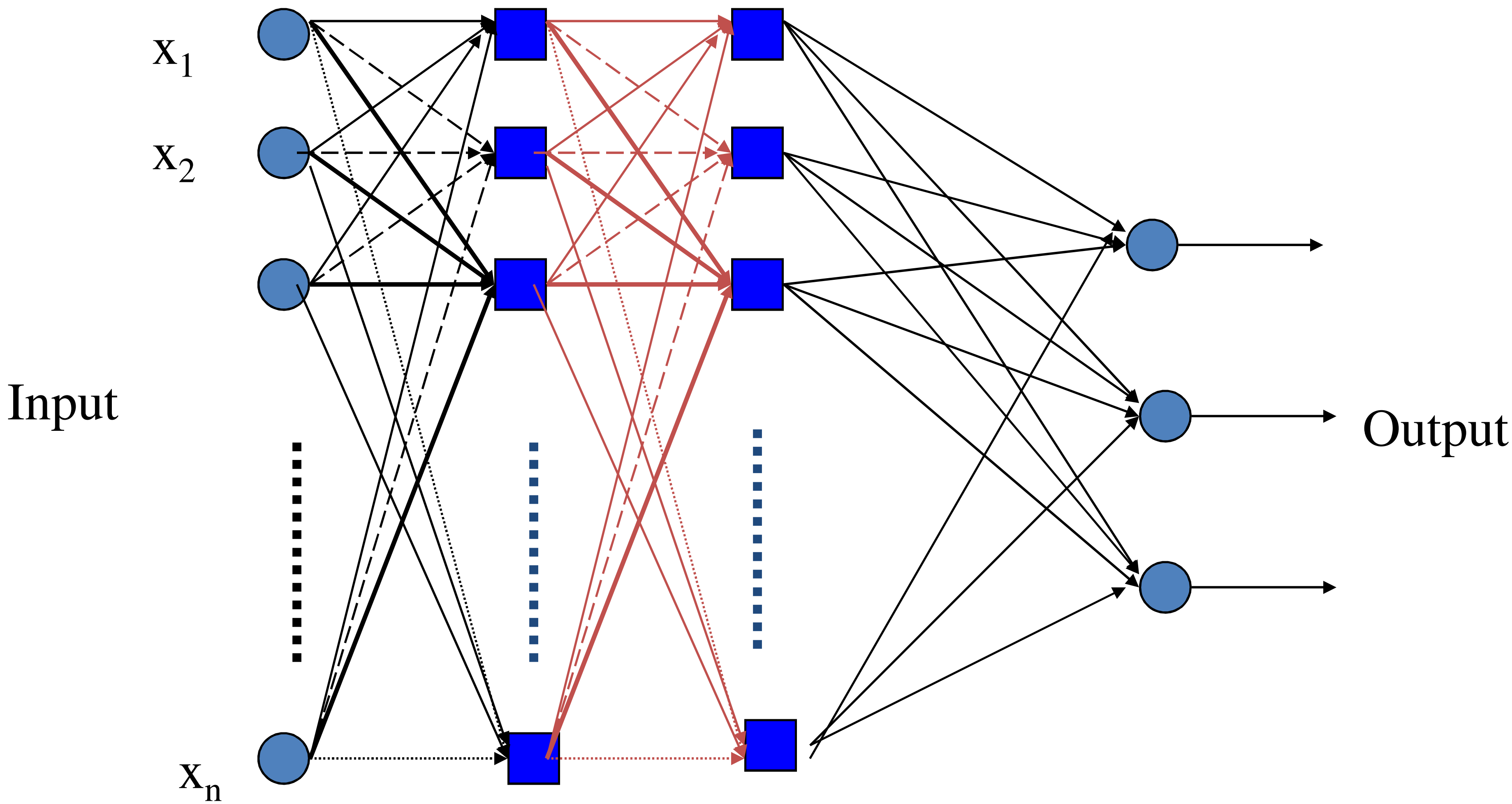
Inputs		Output of Hidden Nodes		Output Node	X_1 XOR X_2
X_1	X_2	H_1	H_2		
0	0	0	0	–0.5 → 0	0
0	1	-1 → 0	1	0.5 → 1	1
1	0	1	-1 → 0	0.5 → 1	1
1	1	0	0	–0.5 → 0	0

Since we are representing two states by 0 (false) and 1 (true), we will map negative outputs (–1, –0.5) of hidden and output layers to 0 and positive output (0.5) to 1.



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

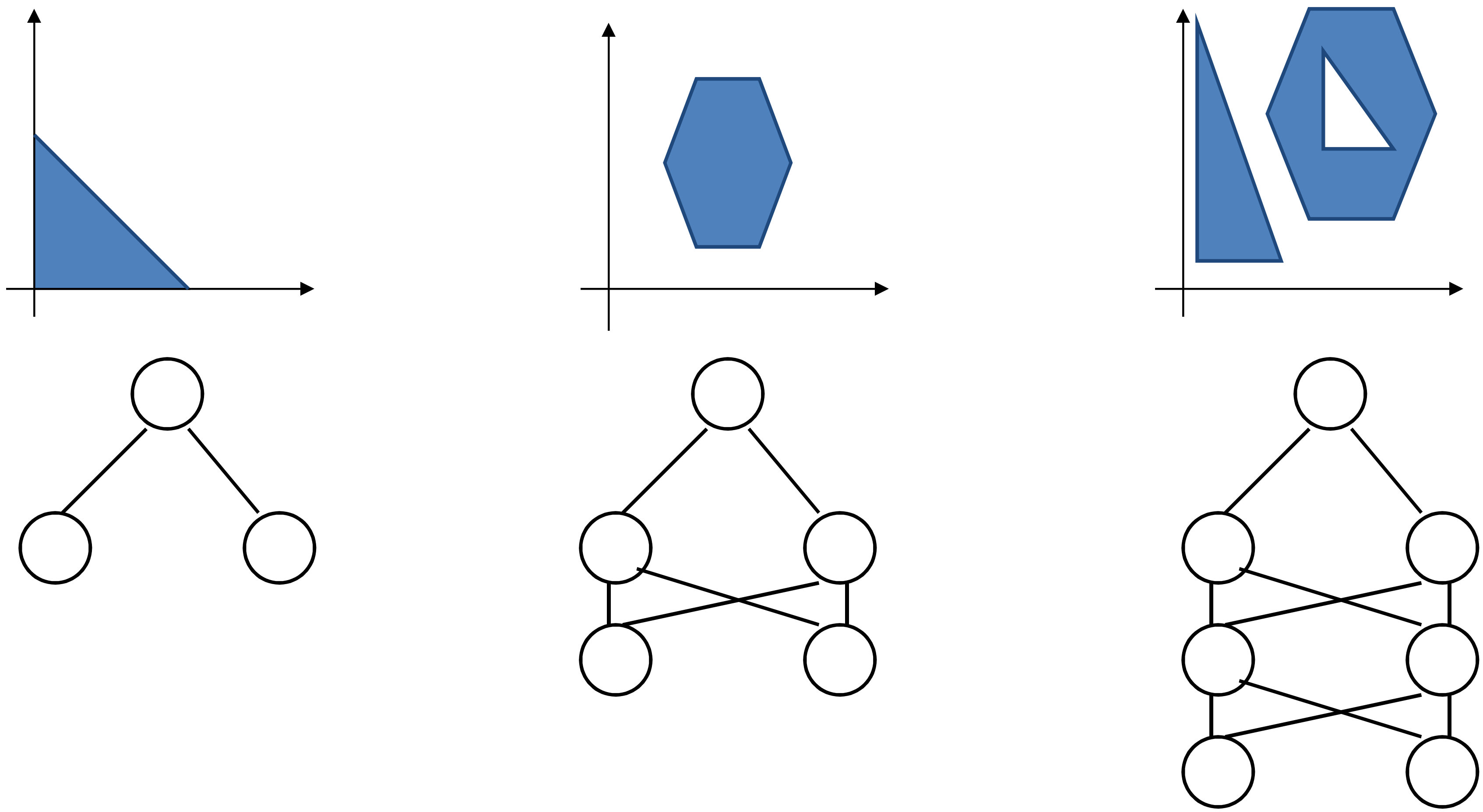
Three-layer networks



Hidden layers

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

What do each of the layers do?



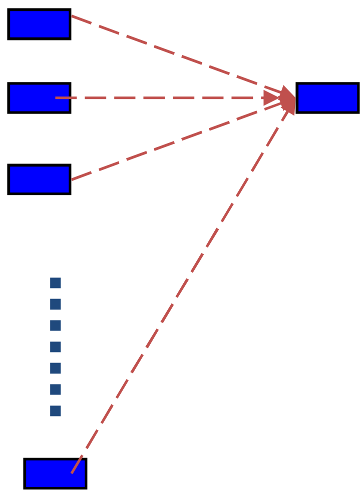
1st layer draws linear boundaries

2nd layer combines the boundaries

3rd layer can generate arbitrarily complex boundaries

Properties of architecture

- No connections within a layer

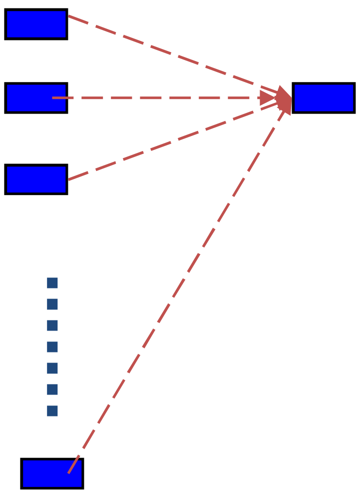


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
-

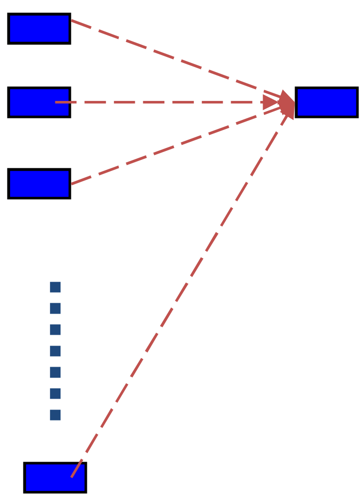


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
-



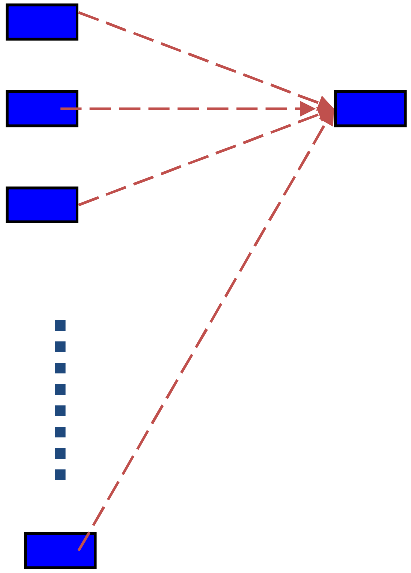
Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

Albert Einstein College of Medicine of Yeshiva University

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units



Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

Often include bias as an extra weight

Backpropagation learning algorithm 'BP'

Solution to credit assignment problem in MLP. *Rumelhart, Hinton and Williams (1986) (*though actually invented earlier in a PhD thesis relating to economics)

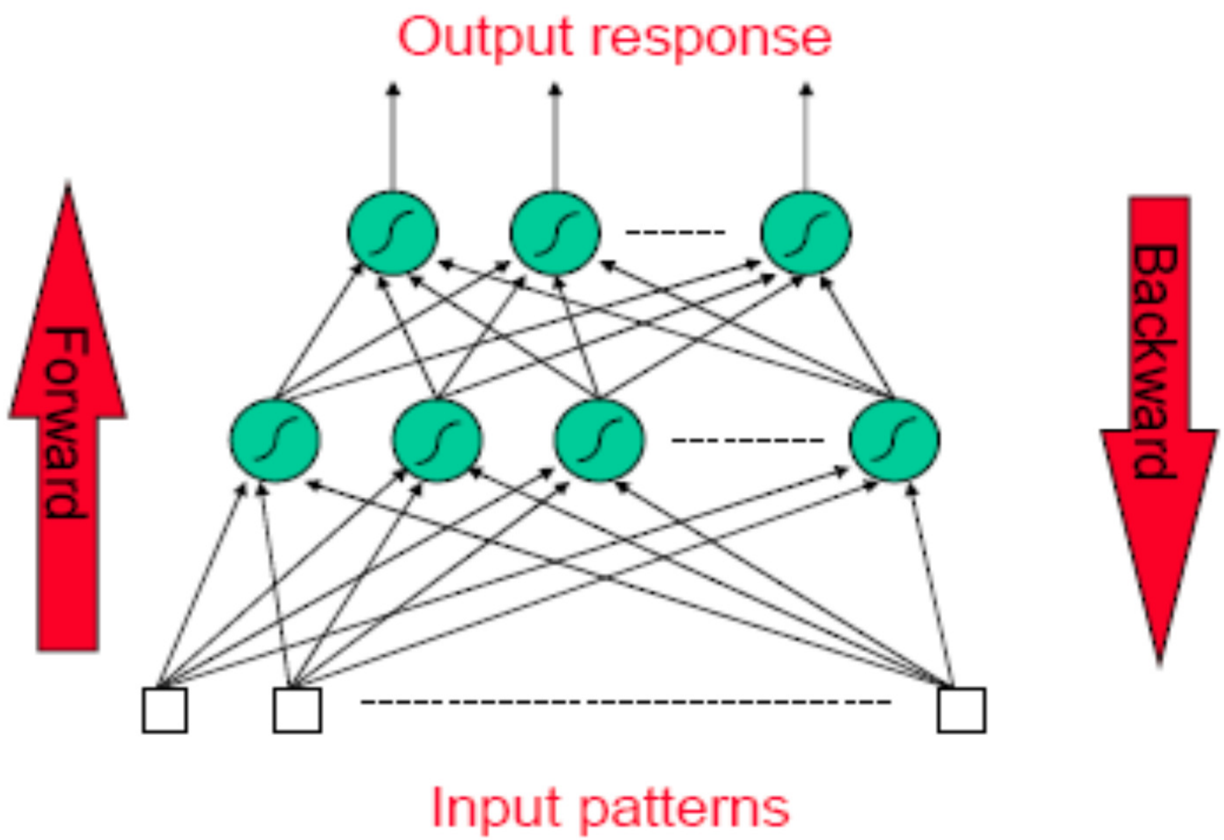
BP has two phases:

Forward pass phase: computes 'functional signal', feed forward propagation of input pattern signals through network

Backward pass phase: computes 'error signal', *propagates* the error *backwards* through network starting at output units (where the error is the difference between actual and desired output values)

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Conceptually: Forward Activity - Backward Error



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Forward Propagation of Activity

- Step 1: Initialize weights at random, choose a learning rate η
- Until network is trained:
- For each training example i.e. input pattern and target output(s):
- Step 2: Do forward pass through net (with fixed weights) to produce output(s)
  - i.e., in Forward Direction, layer by layer:
    - Inputs applied
    - Multiplied by weights
    - Summed
    - 'Squashed' by sigmoid activation function
    - Output passed to each neuron in next layer
  - Repeat above until network output(s) produced

Step 3. Back-propagation of error

• Compute error (delta or local gradient) for each output unit $\delta\ k$
• Layer-by-layer, compute error (delta or local gradient) for each hidden unit $\delta\ j$ by backpropagating errors

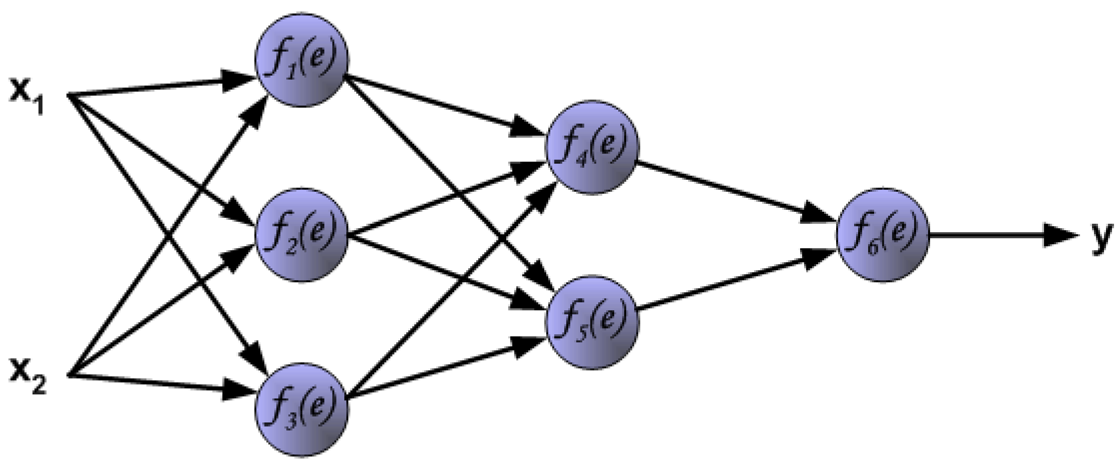
Step 4: Next, update all the weights Δwij
By gradient descent, and go back to Step 2

– The overall MLP learning algorithm, involving forward pass and backpropagation of error (until the network training completion), is known as the Generalised Delta Rule (GDR), or more commonly, the Back Propagation (BP) algorithm

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

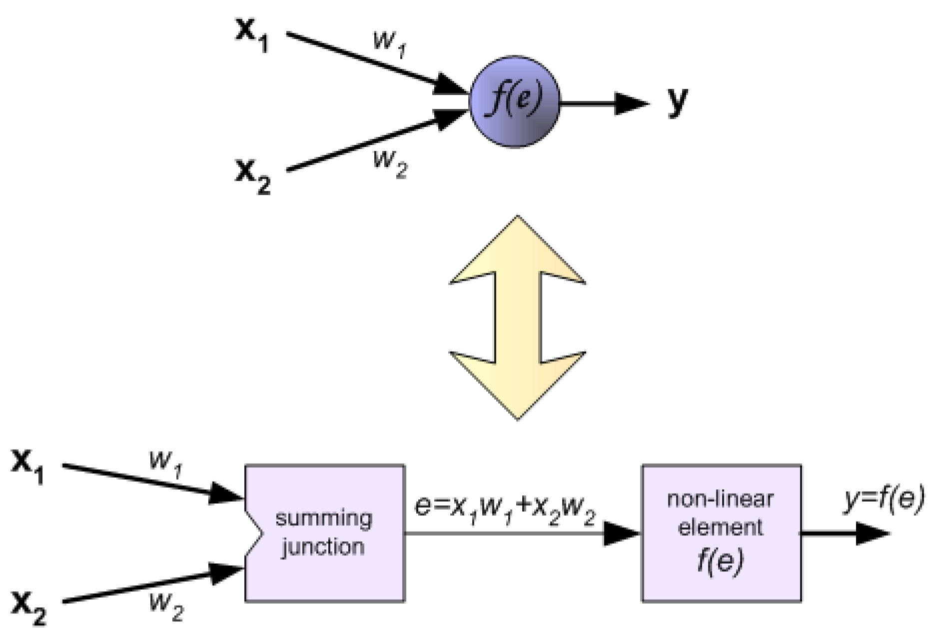
Learning Algorithm: Backpropagation

The following slides describes **teaching process** of multi-layer neural network employing **backpropagation** algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:



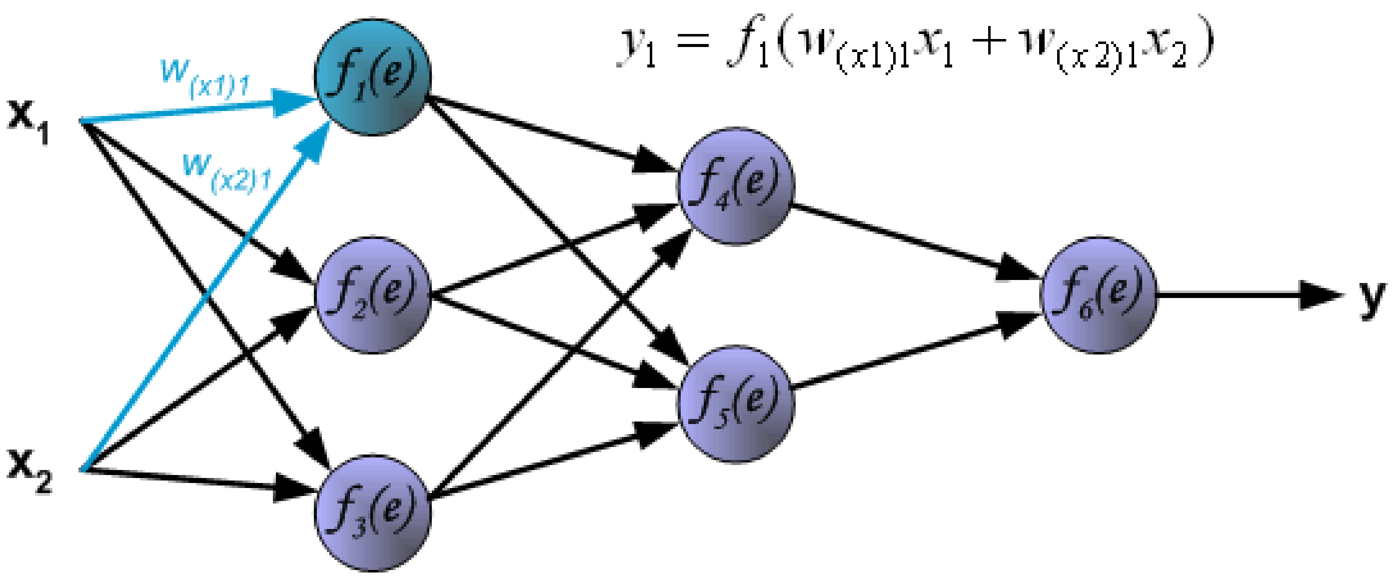
Learning Algorithm: Backpropagation

Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron transfer (activation) function. Signal *e* is adder output signal, and *y = f(e)* is output signal of nonlinear element. Signal *y* is also output signal of neuron.

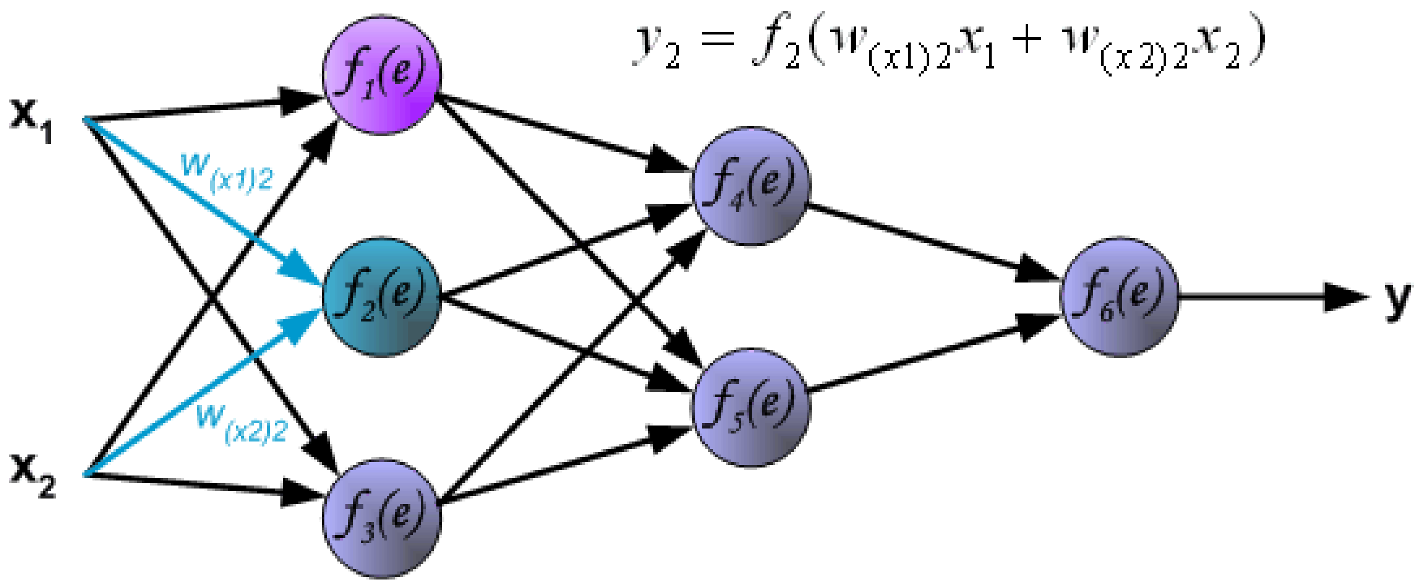


Learning Algorithm: Backpropagation

Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.

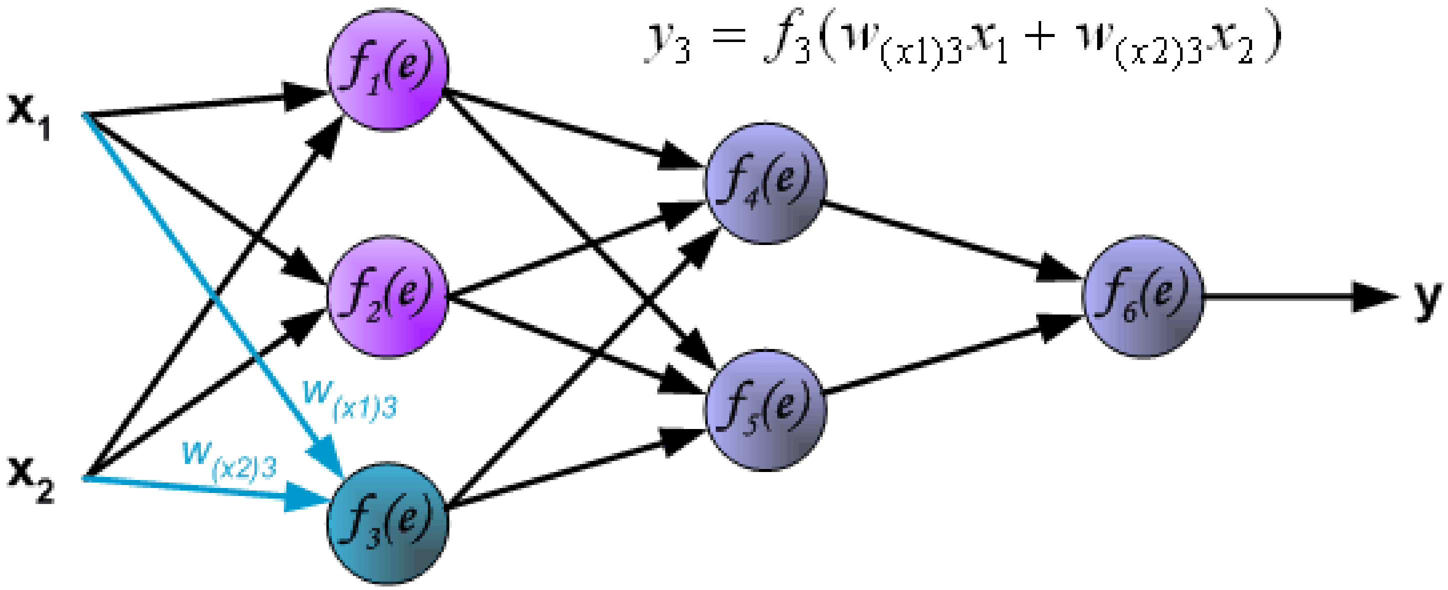


Learning Algorithm: Backpropagation



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

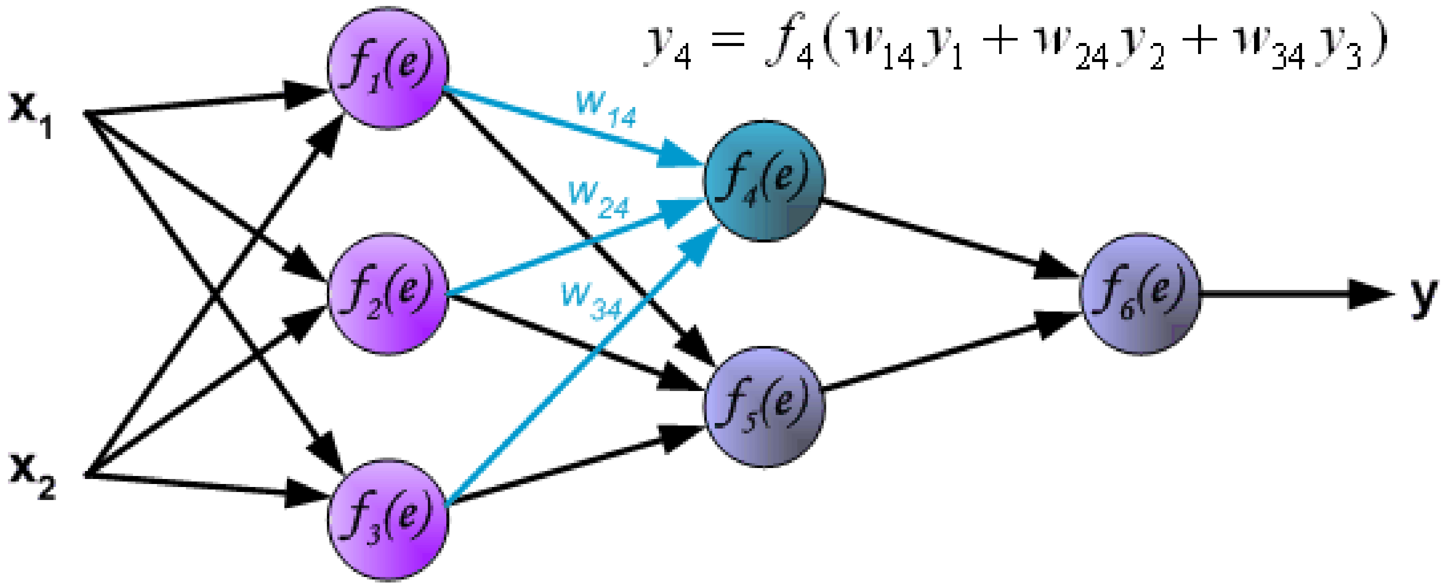
Learning Algorithm: Backpropagation



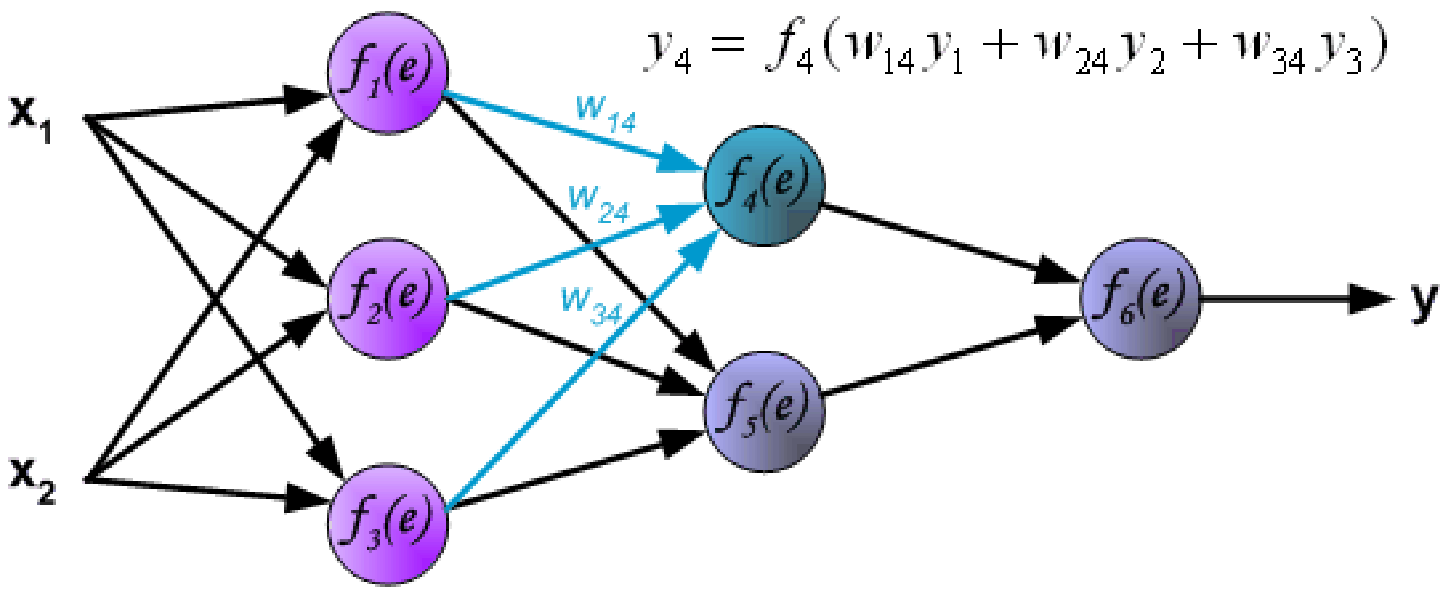
$$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$$

Learning Algorithm: Backpropagation

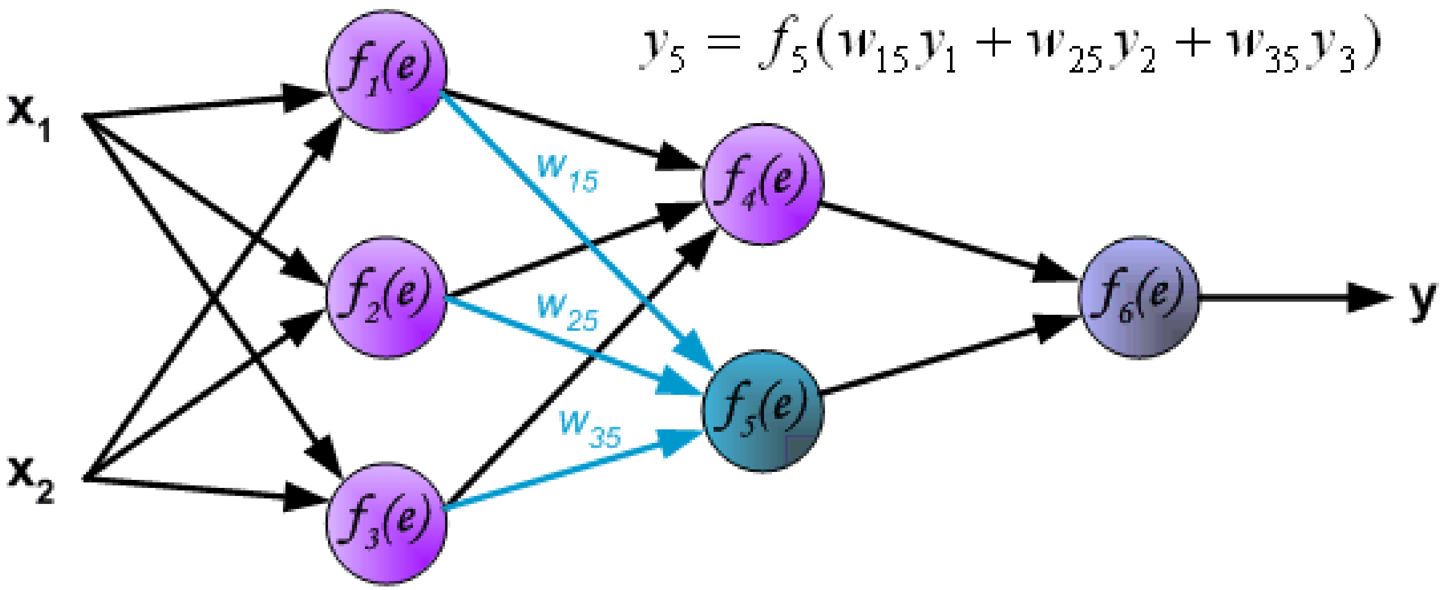
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron *m* and input of neuron *n* in the next layer.



Learning Algorithm: Backpropagation

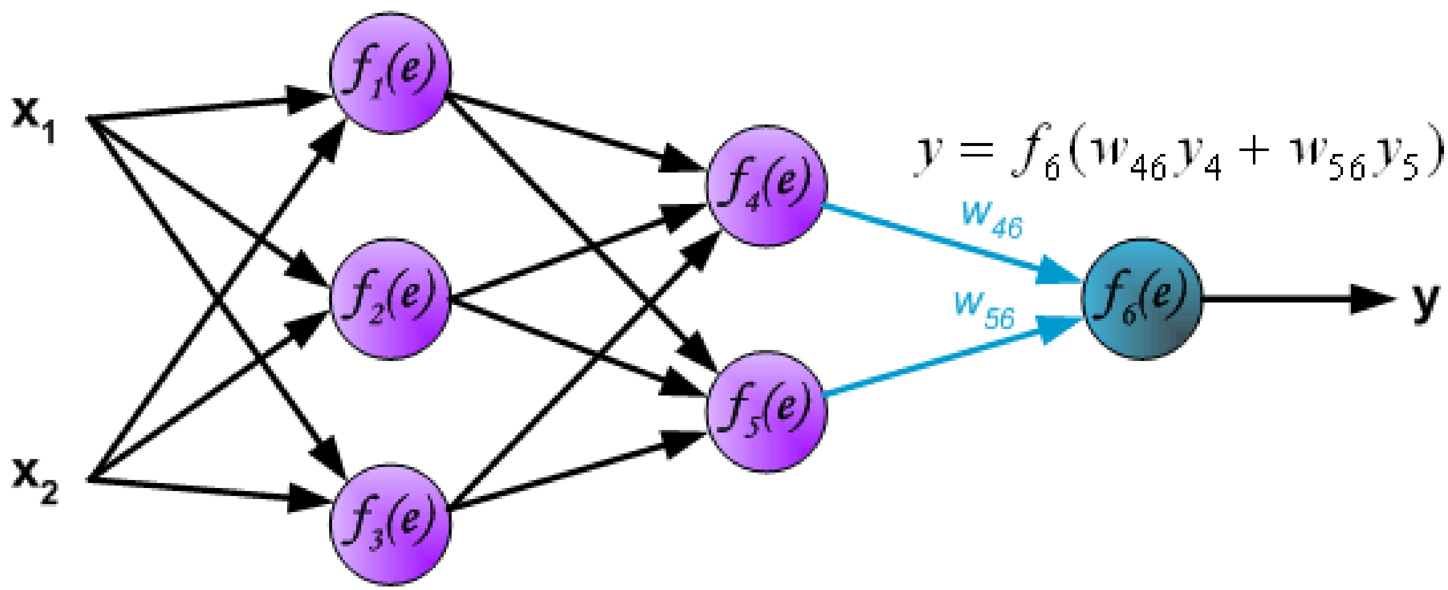


Learning Algorithm: Backpropagation



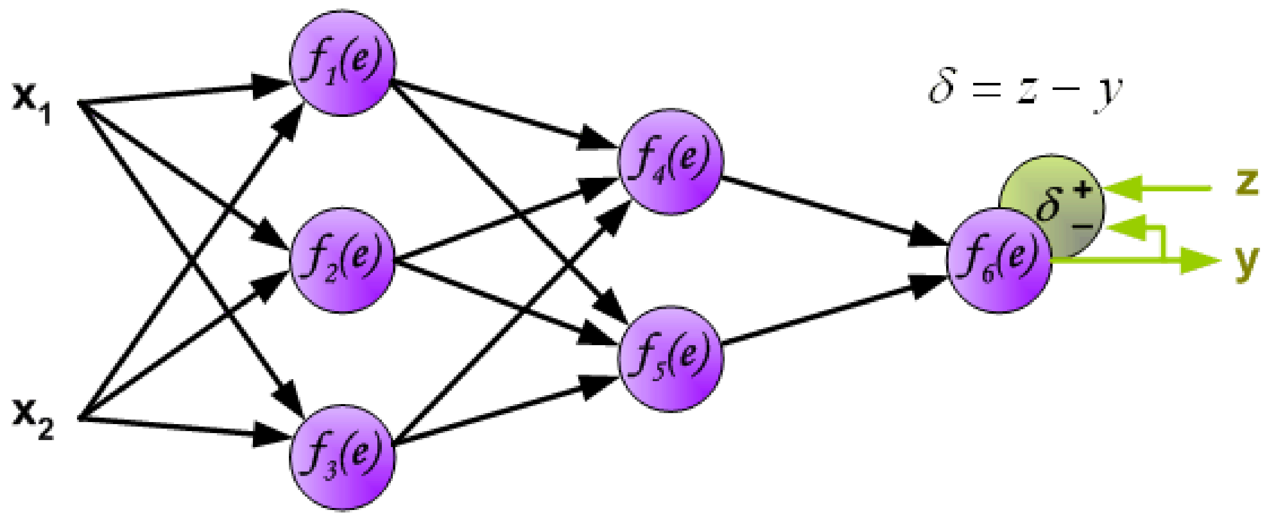
Learning Algorithm: Backpropagation

Propagation of signals through the output layer.



Learning Algorithm: Backpropagation

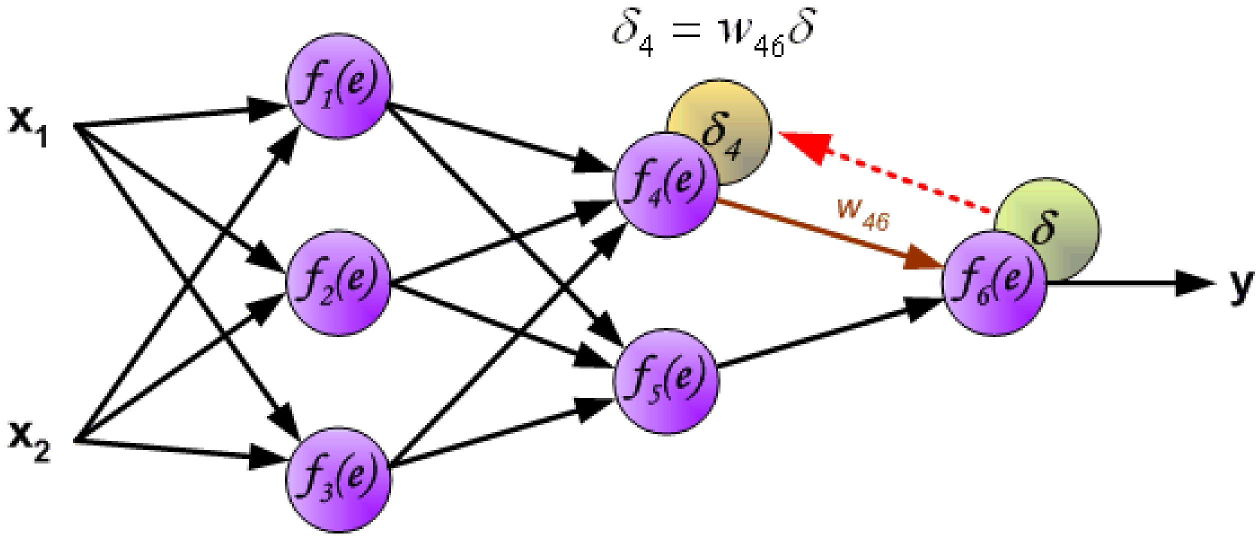
In the next algorithm step the output signal of the network *y* is compared with the desired output value (the target), which is found in training data set. The difference is called error signal *d* of output layer neuron



Albert Einstein College of Medicine of Yeshiva University

Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

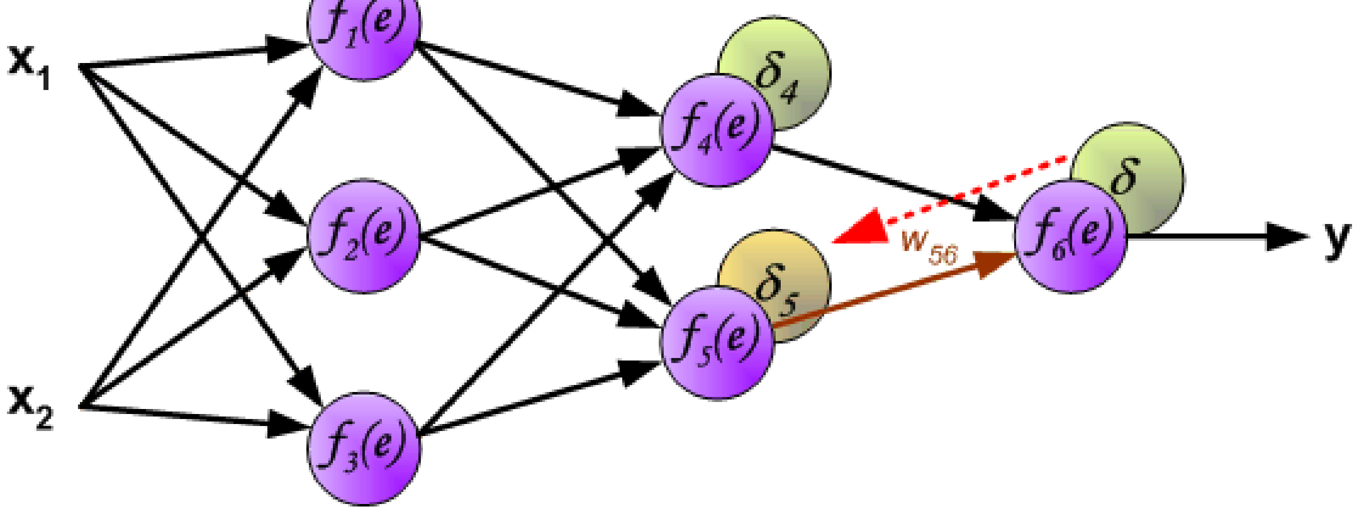


Albert Einstein College of Medicine OF YESHIVA UNIVERSITY

Learning Algorithm: Backpropagation

The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

$$\delta_5 = w_{56}\delta$$

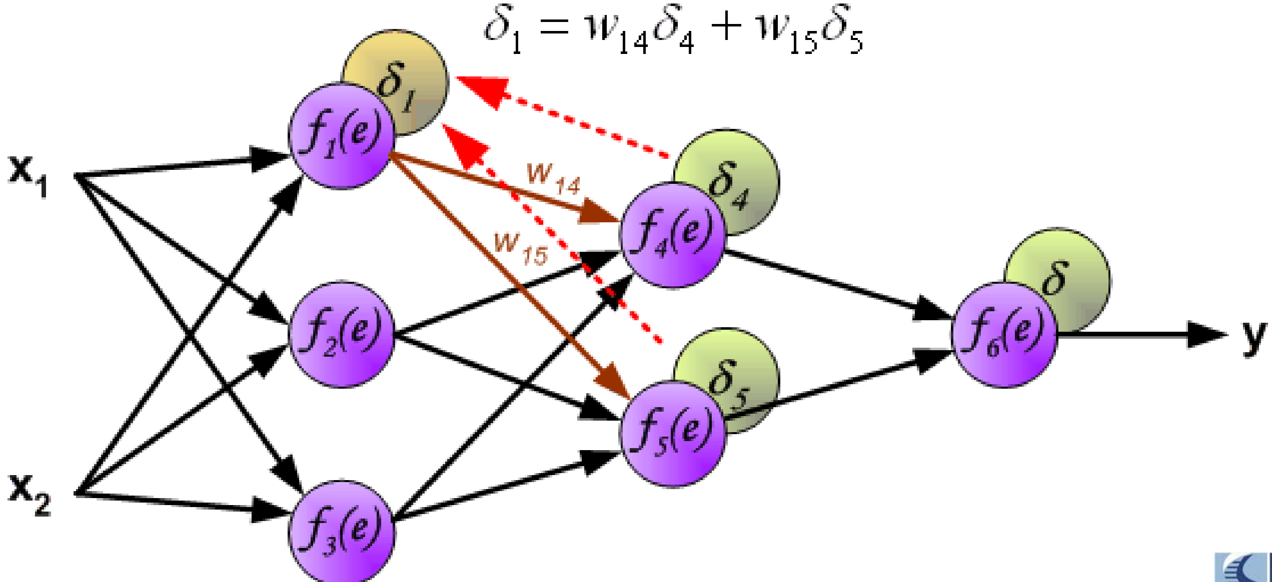


Albert Einstein College of Medicine OF YESHIVA UNIVERSITY

Learning Algorithm: Backpropagation

The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$

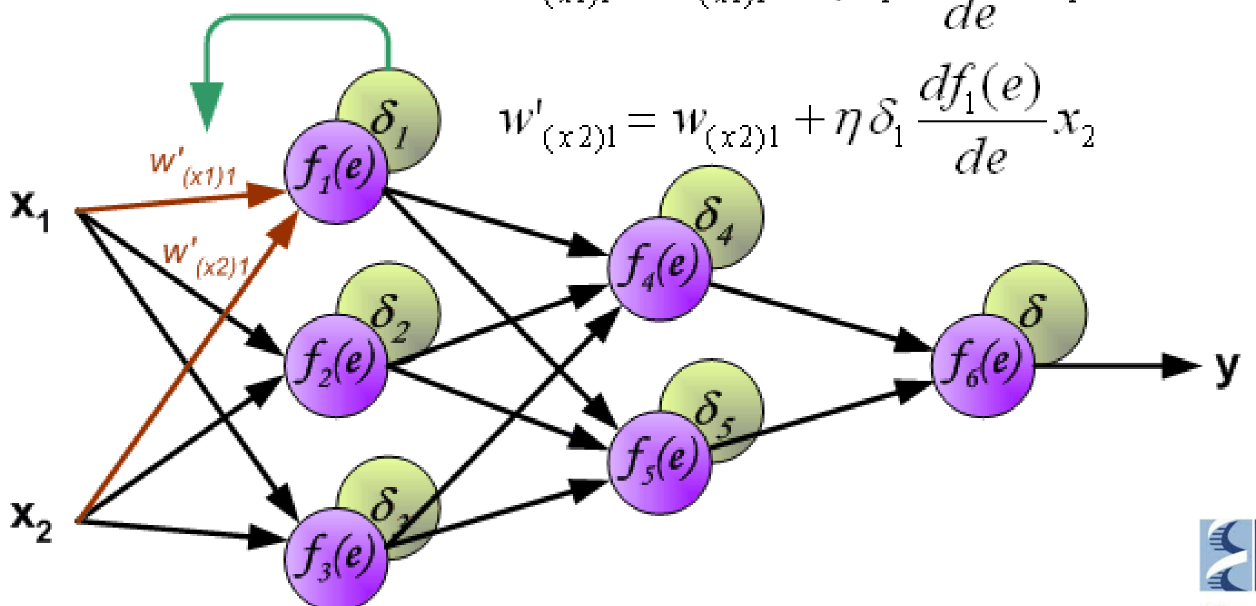


Learning Algorithm: Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

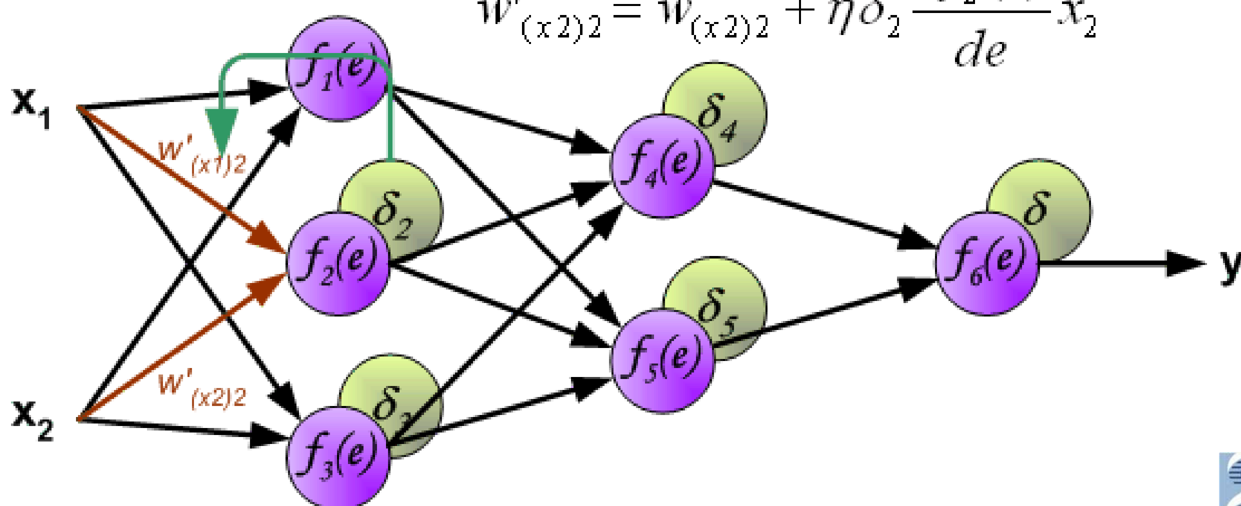


Learning Algorithm: Backpropagation

When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

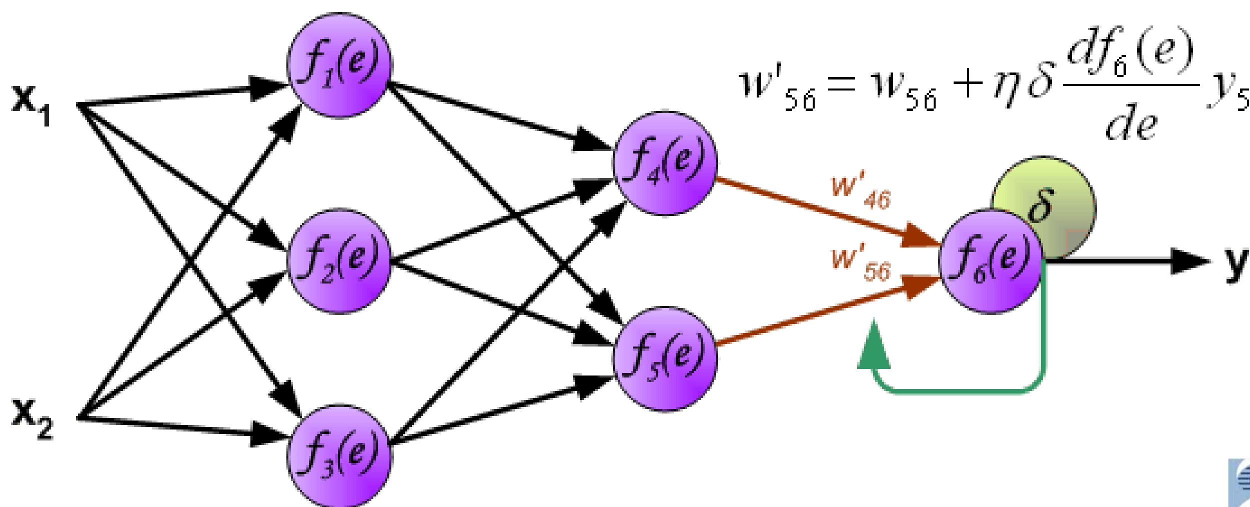


Learning Algorithm: Backpropagation

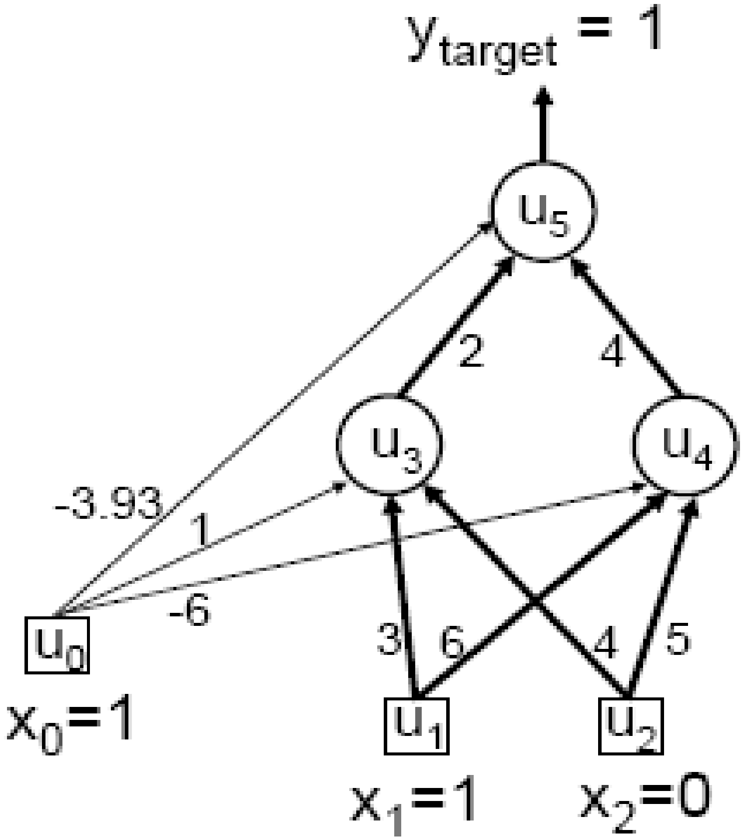
When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



MLP/BP: A worked example



Current state:

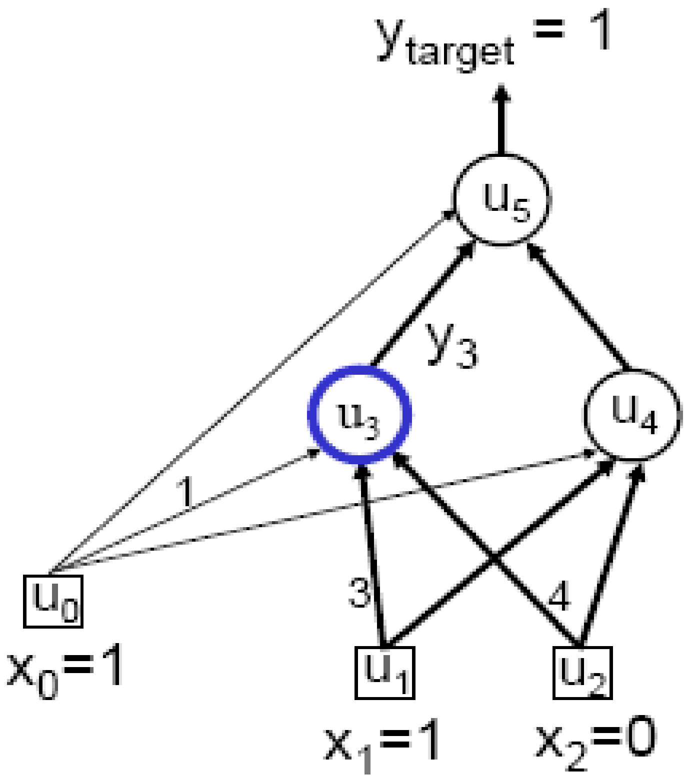
- Weights on arrows e.g. $w_{13} = 3$, $w_{35} = 2$, $w_{24} = 5$
- Bias weights, e.g.

bias for unit 4 (u_4) is $w_{04} = -6$

Training example (e.g. for logical OR problem):

- Input pattern is $x_1=1$, $x_2=0$
- Target output is $y_{target}=1$

Worked example: Forward Pass



Output for any neuron/unit j can be calculated from:

$$a_j = \sum_i w_{ij} x_i$$

$$y_j = f(a_j) = \frac{1}{1+e^{-a_j}}$$

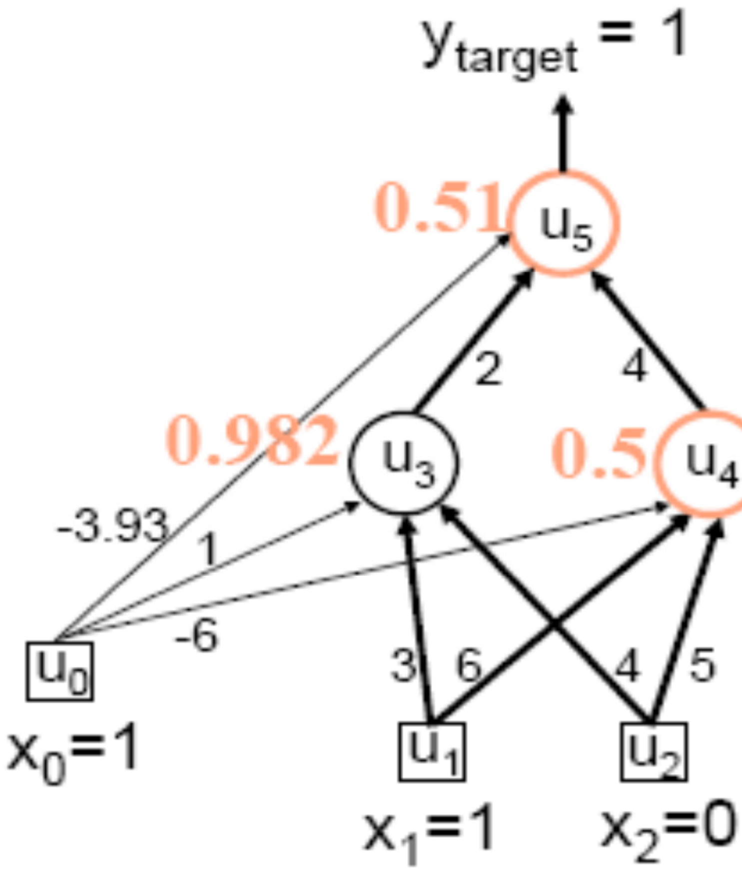
e.g Calculating output for Neuron/unit 3 in hidden layer:

$$a_3 = 1*1 + 3*1 + 4*0 = 4$$

$$y_3 = f(4) = \frac{1}{1+e^{-4}} = 0.982$$

Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

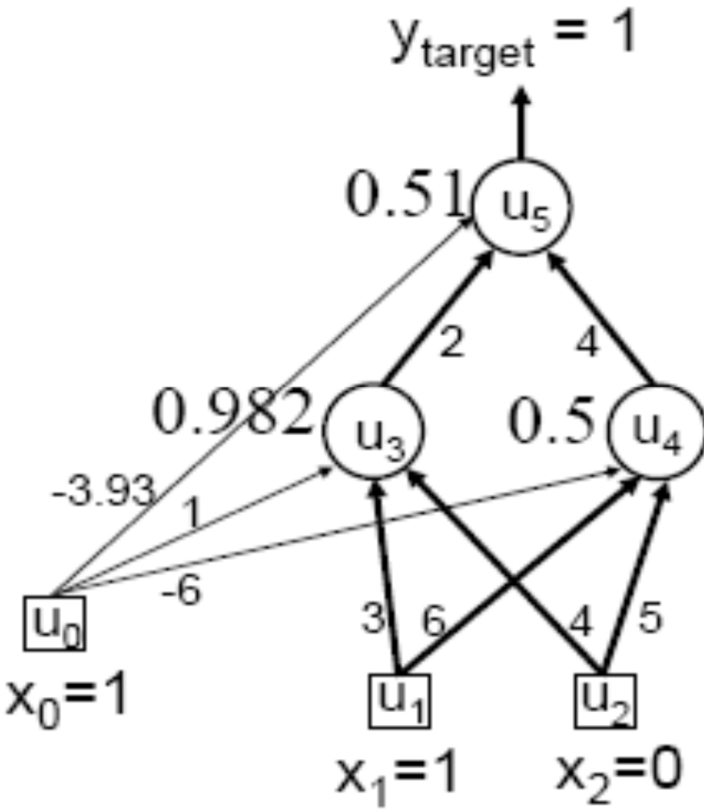
Worked example: Forward Pass



Unit	**activation** a_j	**output** y_j
u_3	4.00	0.982
u_4	0.00	0.500
u_5	0.04	**0.510** (network output)

So the error for this training example is: $(y_{target} - y_5) = (1 - 0.510) = 0.490$

Worked example: Backward Pass



Now compute delta values starting at the output:

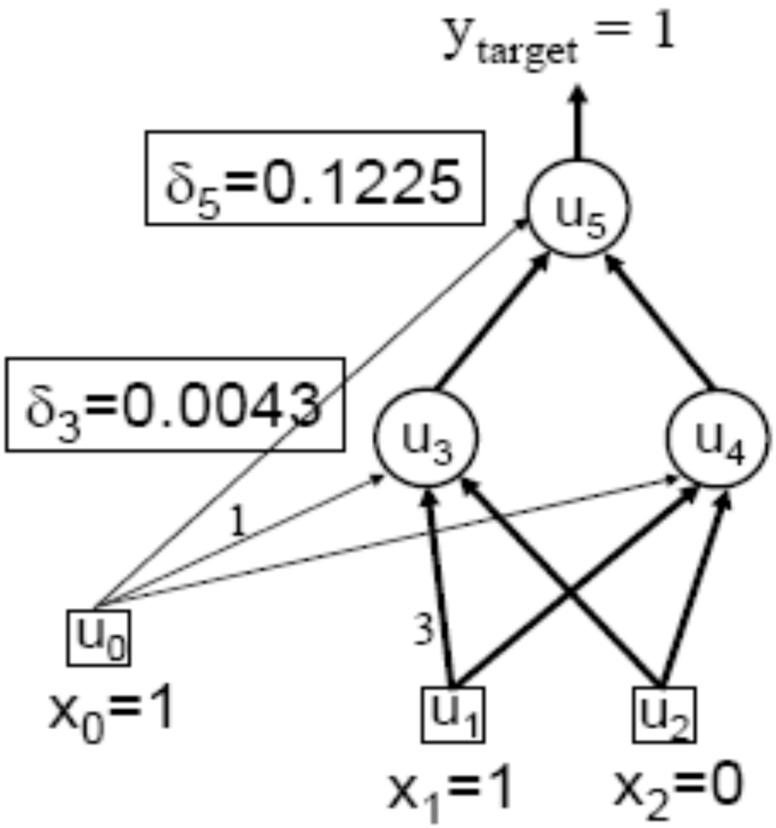
$$\delta_5 = y_5(1 - y_5)(y_{target} - y_5) = 0.51(1 - 0.51) \times 0.49 = \mathbf{0.1225}$$

Then for hidden units:

$$\delta_4 = y_4(1 - y_4)\, w_{45}\, \delta_5 = 0.5(1 - 0.5) \times 4 \times 0.1225 = \mathbf{0.1225}$$

$$\delta_3 = y_3(1 - y_3)\, w_{35}\, \delta_5 = 0.982(1-0.982) \times 2 \times 0.1225 = \mathbf{0.0043}$$

Worked example: Update Weights Using Generalized Delta Rule (BP)



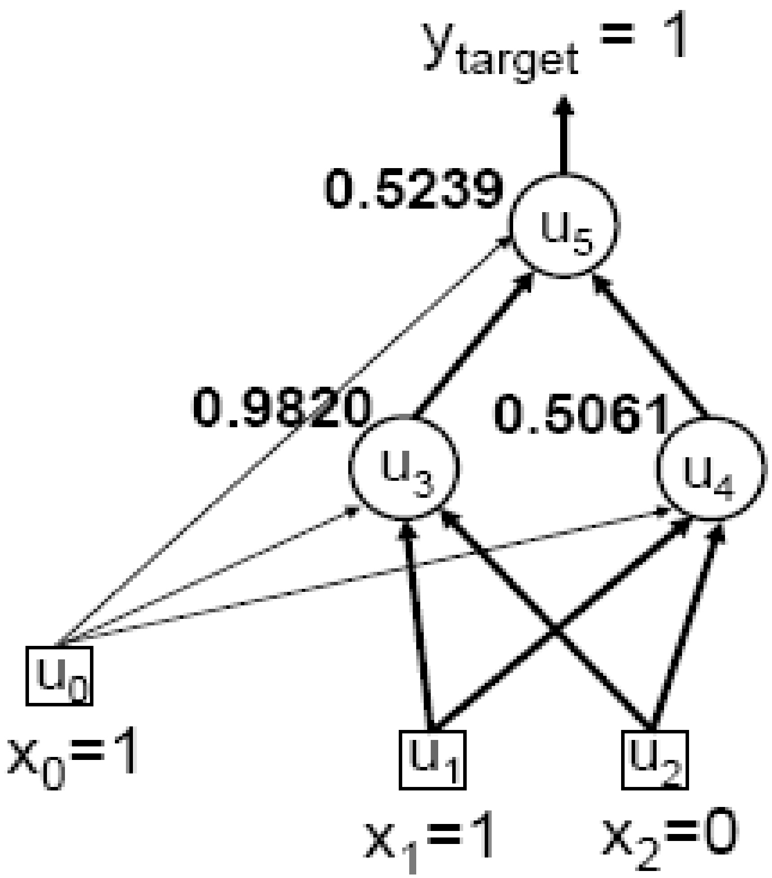
- Set learning rate η = 0.1
  Change weights by:
  $$\Delta w_{ij} = \eta \delta_j y_i$$
- e.g.bias weight on u_3:
  Δw_{03} = $\eta \delta_3 x_0$
  =0.1*0.0043*1
  = 0.0004
  So, new w_{03} ⌫
  w_{03}(old)+Δw_{03}
  =1+0.0004=1.0004
- and likewise:
  w_{13} ⌫ 3 + 0.0004
  = 3.0004

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Similarly for the all weights wij:

i	j	w_{ij}	δ_j	y_i	Updated w_{ij}
0	3	**1**	0.0043	1.0	**1.0004**
1	3	**3**	0.0043	1.0	**3.0004**
2	3	**4**	0.0043	0.0	**4.0000**
0	4	**-6**	0.1225	1.0	**-5.9878**
1	4	**6**	0.1225	1.0	**6.0123**
2	4	**5**	0.1225	0.0	**5.0000**
0	5	**-3.92**	0.1225	1.0	**-3.9078**
3	5	**2**	0.1225	0.9820	**2.0120**
4	5	**4**	0.1225	0.5	**4.0061**

Verification that it works



On <u>next forward pass</u>:

The new activations are:

$y_3 = f\ (4.0008) = 0.9820$

$y_4 = f\ (0.0245) = 0.5061$

$y_5 = f\ (0.0955) = \mathbf{0.5239}$

Thus the <u>new error</u>

$(y_{target} - y_5) = (1 - 0.5239) = 0.476$

<u>has been reduced</u> by **0.014**

(from **0.490** to **0.476**)

Ref: "Neural Network Learning & Expert Systems" by Stephen Gallant

Training

- This was a single iteration of back-prop
- Training requires many iterations with many training examples or *epochs* (one epoch is entire presentation of complete training set)
- It can be slow !
- Note that computation in MLP is local (with respect to each neuron)
- Parallel computation implementation is also possible

Training and testing data

- How many examples ?
  - The more the merrier !
- Disjoint training and testing data sets
  - learn from training data but evaluate performance (generalization ability) on unseen test data
- **Aim**: minimize error on *test* data

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab

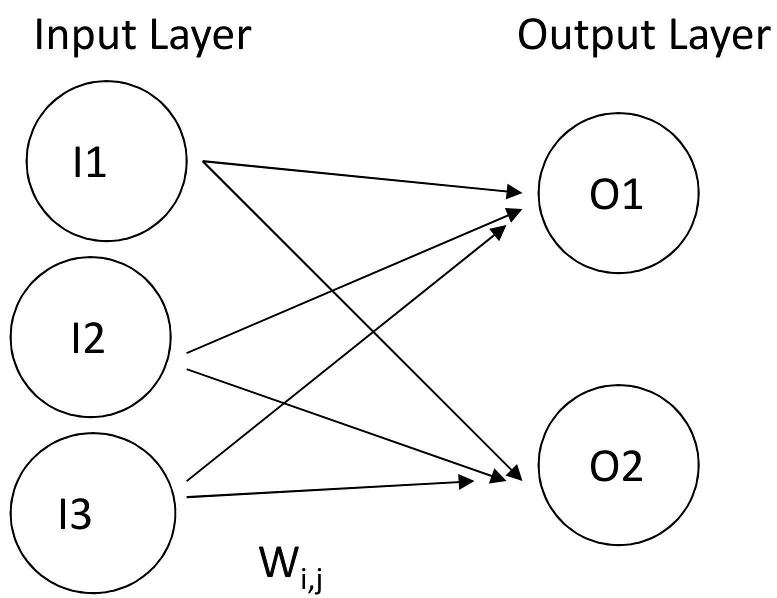
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Unsupervised Learning – Self Organizing Maps

- Self-organizing maps (SOMs) are a data visualization technique invented by Professor Teuvo Kohonen
  - Also called Kohonen Networks, Competitive Learning, Winner-Take-All Learning
  - Generally reduces the dimensions of data through the use of self-organizing neural networks
  - Useful for data visualization; humans cannot visualize high dimensional data so this is often a useful technique to make sense of large data sets

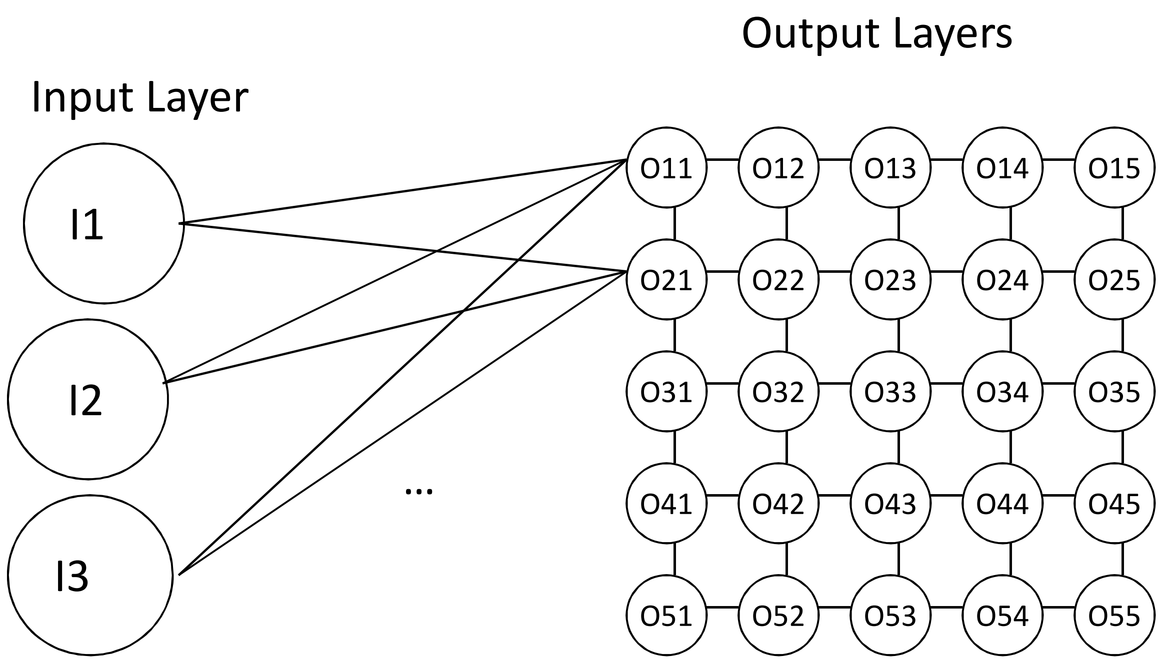
Basic "Winner Take All" Network

- Two layer network
  - Input units, output units, each input unit is connected to each output unit



Typical Usage: 2D Feature Map

- In typical usage the output nodes form a 2D “map” organized in a grid-like fashion and we update weights in a neighborhood around the winner



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Basic Algorithm

- Initialize Map (randomly assign weights)
- Loop over training examples
  - Assign input unit values according to the values in the current example
  - Find the "winner", i.e. the output unit that most closely matches the input units, using some distance metric, e.g.

    For all output units j=1 to m
    and input units i=1 to n
    Find the one that minimizes:

$$\sqrt{\sum_{i=1}^{n}\left(W_{ij}-I_i\right)^2}$$

  - Modify weights on the winner to more closely match the input

$$\Delta W^{t+1} = c(X_i^t - W^t)$$

where c is a small positive learning constant
that usually decreases as the learning proceeds

Result of Algorithm

- Initially, some output nodes will randomly be a little closer to some particular type of input
- These nodes become “winners” and the weights move them even closer to the inputs
- Over time nodes in the output become representative prototypes for examples in the input
- Note there is no supervised training here
- Classification:
  - Given new input, the class is the output node that is the winner

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

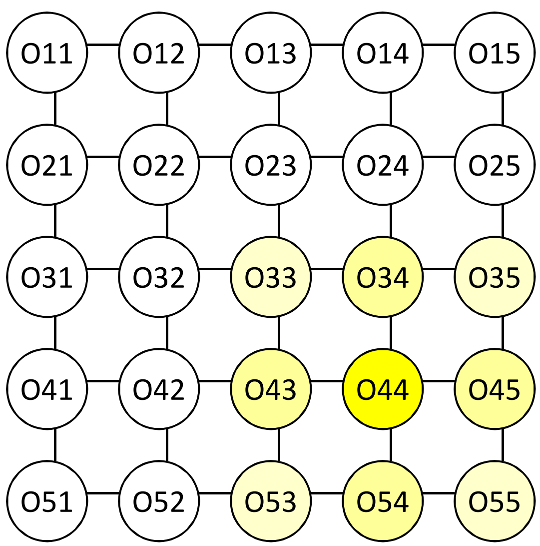
Modified Algorithm

- Initialize Map (randomly assign weights)
- Loop over training examples
  - Assign input unit values according to the values in the current example
  - Find the “winner”, i.e. the output unit that most closely matches the input units, using some distance metric, e.g.
  - Modify weights on the winner to more closely match the input
  - **Modify weights in a neighborhood around the winner so the neighbors on the 2D map also become closer to the input**
    - Over time this will tend to cluster similar items closer on the map

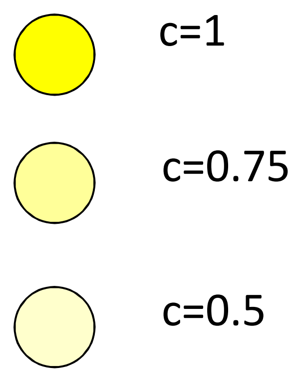
Updating the Neighborhood

- Node O_{44} is the winner
  - Color indicates scaling to update neighbors

Output Layers



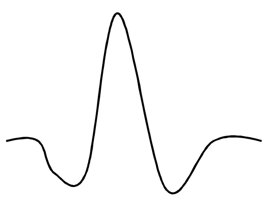
$$\Delta W^{t+1} = c(X_i^t - W^t)$$

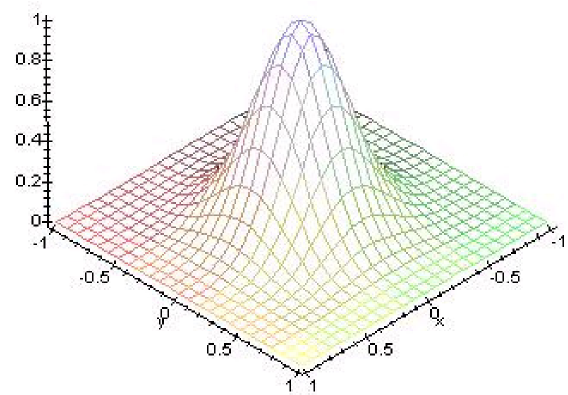


Consider if O_{42} is winner for some other input; "fight" over claiming O_{43}, O_{33}, O_{53}

Selecting the Neighborhood

- Typically, a "Sombrero Function" or Gaussian function is used





- Neighborhood size usually decreases over time to allow initial "jockeying for position" and then "fine-tuning" as algorithm proceeds

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab

Implementation

1. Loading data source.
2. Selecting attributes required.
3. Decide training, validation, and testing data.
4. Data manipulations and Target generation.
   (for supervised learning)
5. Neural Network creation (selection of network architecture) and initialisation.
6. Network Training and Testing.
7. Performance evaluation.

Loading and Saving data

- load: retrieve data from disk.
  - In ascii or .mat format.

```
>> data = load('nndata.txt');

>> whos data;

Name        Size         Bytes  Class

data     826x7          46256  double array
```

- Save: saves variables in matlab environment in .mat format.

```
>> save nnoutput.txt x, y ,z;
```

Matrix manipulation

- region = data(:,1);
- training = data([1:500],:)
- w=[1;2]; w*w' => [1,2;2,4];

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

- w=[1,2;2,4]; w.*w => [1,4;4,16];

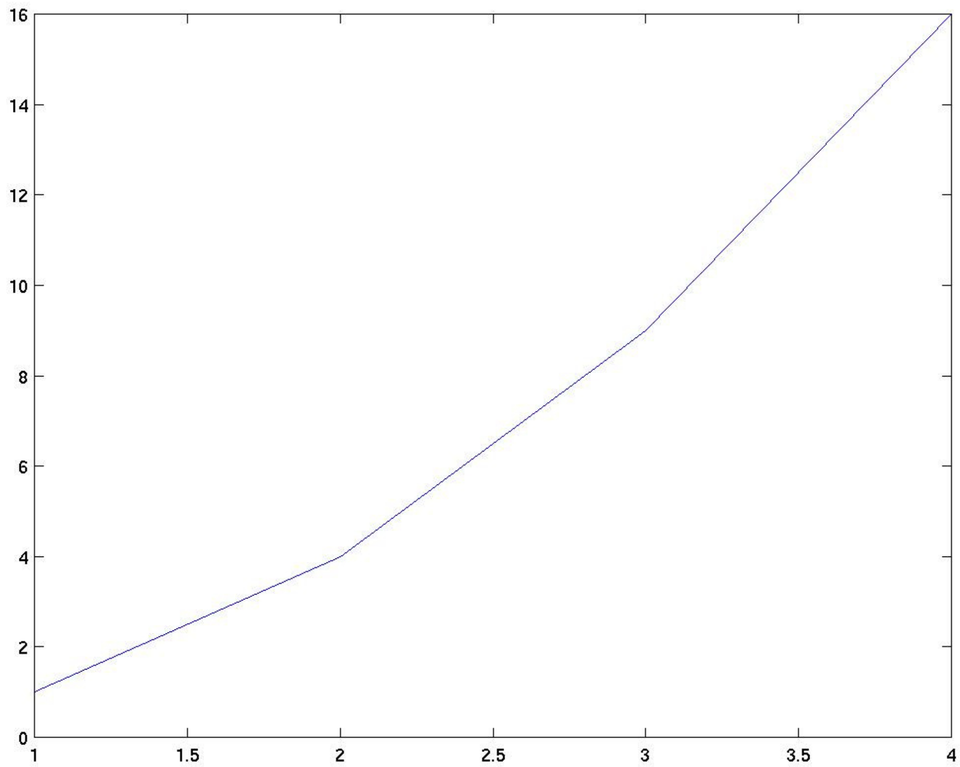
$$\begin{pmatrix} 1 & 4 \\ 4 & 16 \end{pmatrix}$$

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Plotting Data

- plot : plot the vector in 2D or 3D

>> y = [1 2 3 4]; figure(1); plot(power(y,2));



Redefine x axis:
>> x = [2 4 6 8];
>> plot(x,power(y,2));

Network creation

>>net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

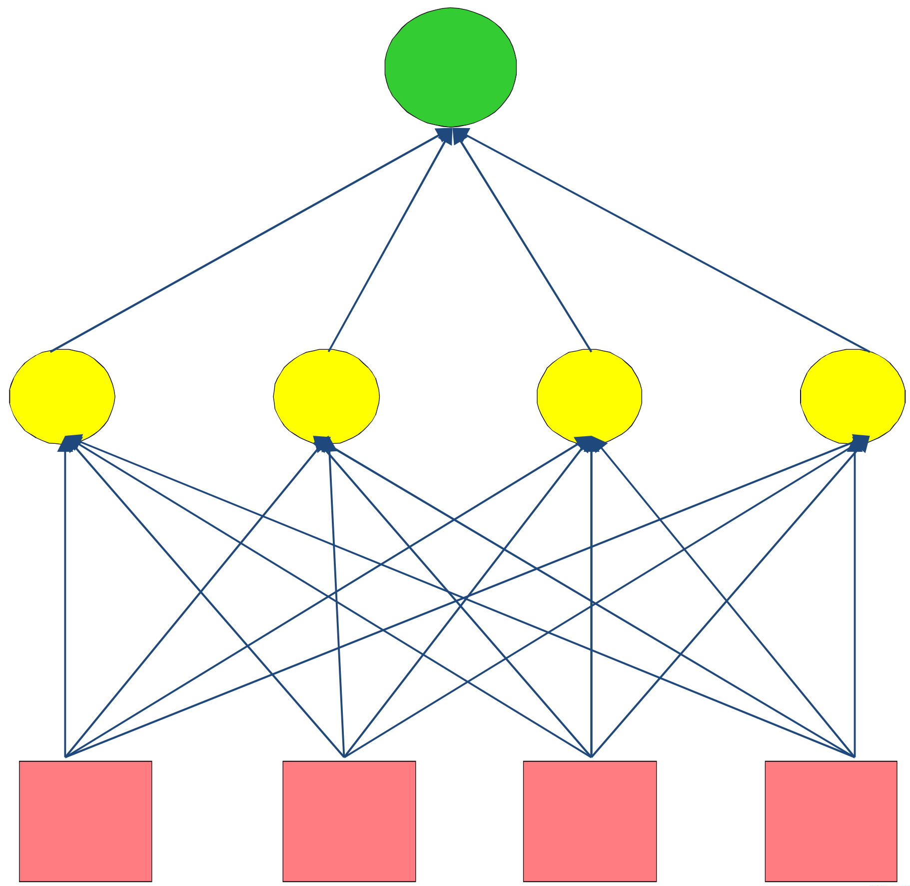
- PR - Rx2 matrix of min and max values for R input elements.
- Si - Size of ith layer, for Nl layers.
- TFi - Transfer function of ith layer, default = 'tansig'.
- BTF - Backprop network training function,
  - default = 'trainlm'.
- BLF - Backprop weight/bias learning function,
  - default = 'learngdm'.
- PF - Performance function,
  - default = 'mse'
- newff : create and returns "net" = a feed-forward backpropagatio

Network creation (cont.)

S2: number of ouput neuron

S1: number hidden neurons

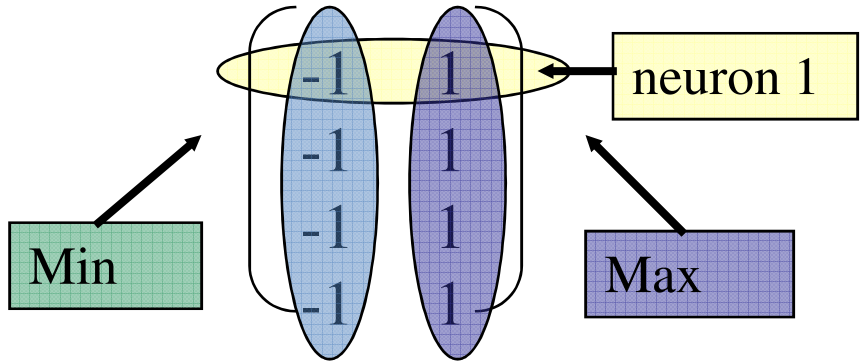
Number of inputs decided by PR



Albert Einstein College of Medicine OF YESHIVA UNIVERSITY

Network Initialisation

>> PR = [-1 1; -1 1; -1 1; -1 1];



- Initialise the net's weighting and biases
- >> net = init(net);   % init is called after newff

- re-initialise with other function:
  - net.layers{1}.initFcn = 'initwb';
  - net.inputWeights{1,1}.initFcn = 'rands';
  - net.biases{1,1}.initFcn = 'rands';
  - net.biases{2,1}.initFcn = 'rands';

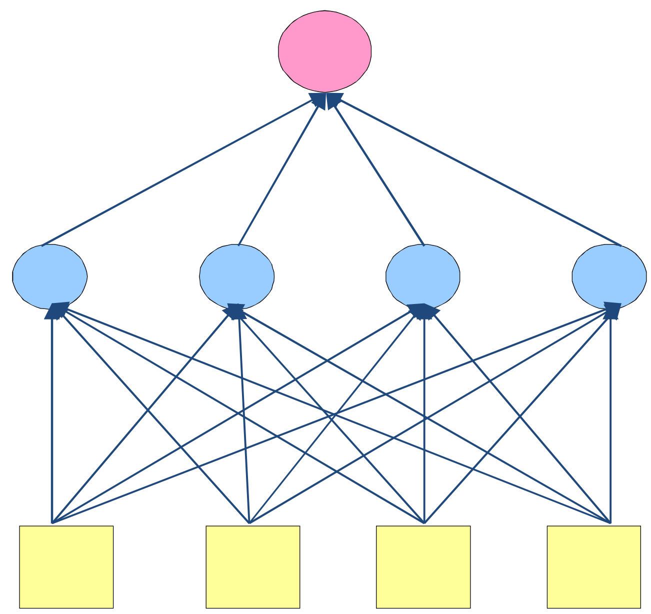
Albert Einstein College of Medicine OF YESHIVA UNIVERSITY

Neurons activation

>> net = newff([-1 1; -1 1; -1 1; -1 1], [4,1], {'logsig' 'logsig'});

TF2: logsig

TF1: logsig



Network Training

- The overall architecture of your neural network is store in the variable **net**;
- variable can be reset.

net.trainParam.epochs =1000;	(Max no. of epochs to train) [100]
net.trainParam.goal =0.01;	(stop training if the error goal hit) [0]
net.trainParam.lr =0.001;	(learning rate, not default trainlm) [0.01]
net.trainParam.show =1;	(no. epochs between showing error) [25]
net.trainParam.time =1000;	(Max time to train in sec) [inf]

net.trainParam parameters:

- epochs: 100
- goal: 0
- max_fail: 5
- mem_reduc: 1
- min_grad: 1.0000e-010
- mu: 0.0010
- mu_dec: 0.1000
- mu_inc: 10
- mu_max: 1.0000e+010
- show: 25
- time: Inf

net.trainFcn options

- net.trainFcn=trainlm ; a variant of BP based on second order algorithm (***Levenberg-Marquardt)***

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

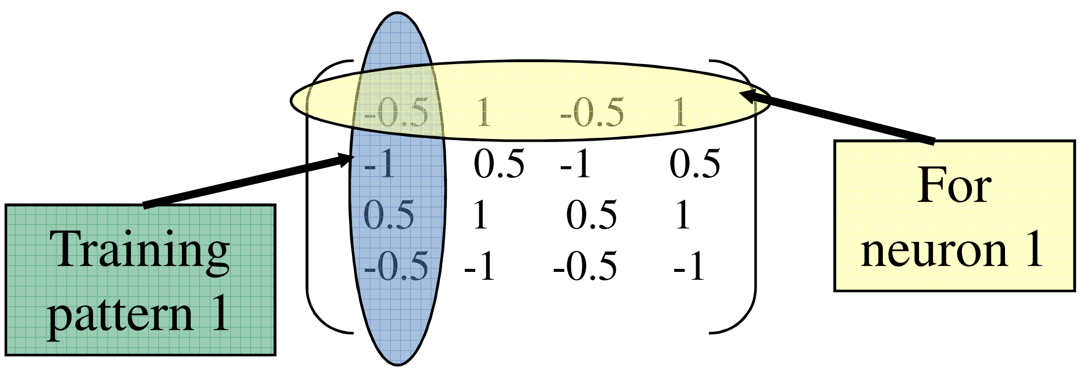
Network Training(cont.)

TRAIN trains a network NET according to NET.trainFcn and NET.trainParam.

>> TRAIN(NET,P,T,Pi,Ai)

- NET - Network.
- P   - Network inputs.
- T   - Network targets, default = zeros.
- Pi - Initial input delay conditions, default = zeros.
- Ai - Initial layer delay conditions, default = zeros.

>> p = [-0.5 1 -0.5 1; -1 0.5 -1 0.5; 0.5 1 0.5 1; -0.5 -1 -0.5 -1];



Network Training(cont.)

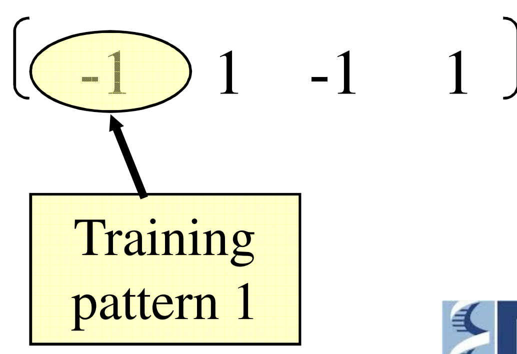
>>TRAIN(NET,P,T,Pi,Ai)

- NET - Network.
- P - Network inputs.
- T - Network targets, default = zeros. (optional only for NN with targets)
- Pi - Initial input delay conditions, default = zeros.
- Ai - Initial layer delay conditions, default = zeros.

```
>> p = [-0.5 1 -0.5 1; -1 0.5 -1 0.5; 0.5 1 0.5 1; -0.5 -1 -0.5 -1];

>> t = [-1 1 -1 1];

>> net = train(net, p, t);
```



EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

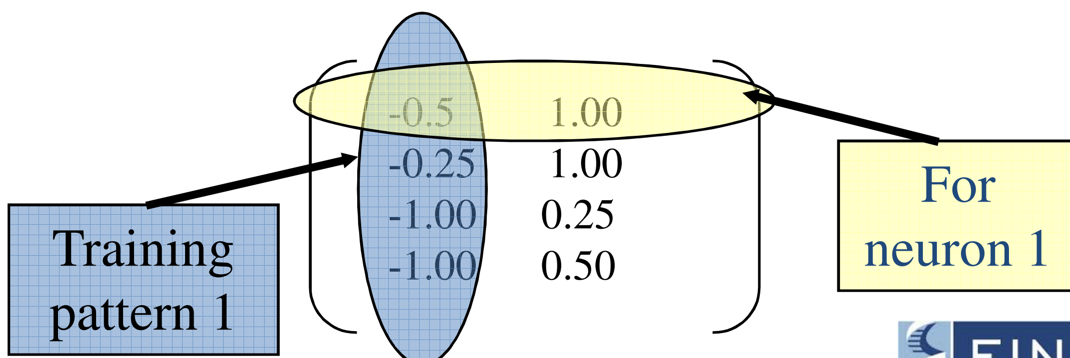
Simulation of the network

>> [Y] = SIM(model, UT)

- Y : Returned output in matrix or structure format.
- model : Name of a block diagram model.
- UT : For table inputs, the input to the model is interpolated.

>> UT = [-0.5 1 ; -0.25 1; -1 0.25 ; -1 0.5];

>> Y = sim(net,UT);



Albert Einstein College of Medicine of Yeshiva University

Performance Evaluation

- Comparison between target and network's output in testing set.
- Comparison between target and network's output in training set.
- Design a metric to measure the distance/similarity of the target and output, or simply use *mse*.

NEWSOM

- Create a self-organizing map.

>> net = newsom(PR,[d1,d2,...],tfcn,dfcn,olr,osteps,tlr,tns)

- PR - Rx2 matrix of min and max values for R input elements.
- Di - Size of ith layer dimension, defaults = [5 8].
- TFCN - Topology function, default = 'hextop'.
- DFCN - Distance function, default = 'linkdist'.
- OLR - Ordering phase learning rate, default = 0.9.
- OSTEPS - Ordering phase steps, default = 1000.
- TLR - Tuning phase learning rate, default = 0.02;
- TND - Tuning phase neighborhood distance, default = 1.

NewSom parameters

- The topology function TFCN can be HEXTOP, GRIDTOP, or RANDTOP.
- The distance function can be LINKDIST, DIST, or MANDIST.

- Exmple:

```
>> P = [rand(1,400)*2; rand(1,400)];
>> net = newsom([0 2; 0 1],[3 5]);
>> plotsom(net.layers{1}.positions)
```

TRAINWB1 By-weight-&-bias 1-vector-at-a-time training function

```
>> [net,tr] = trainwb1(net,Pd,Tl,Ai,Q,TS,VV,TV)
```

EINSTEIN

Albert Einstein College of Medicine OF YESHIVA UNIVERSITY

Outline

- Background
- Supervised learning (BPNN)
- Unsupervised learning (SOM)
- Implementation in Matlab
- Applications

EINSTEIN
Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY