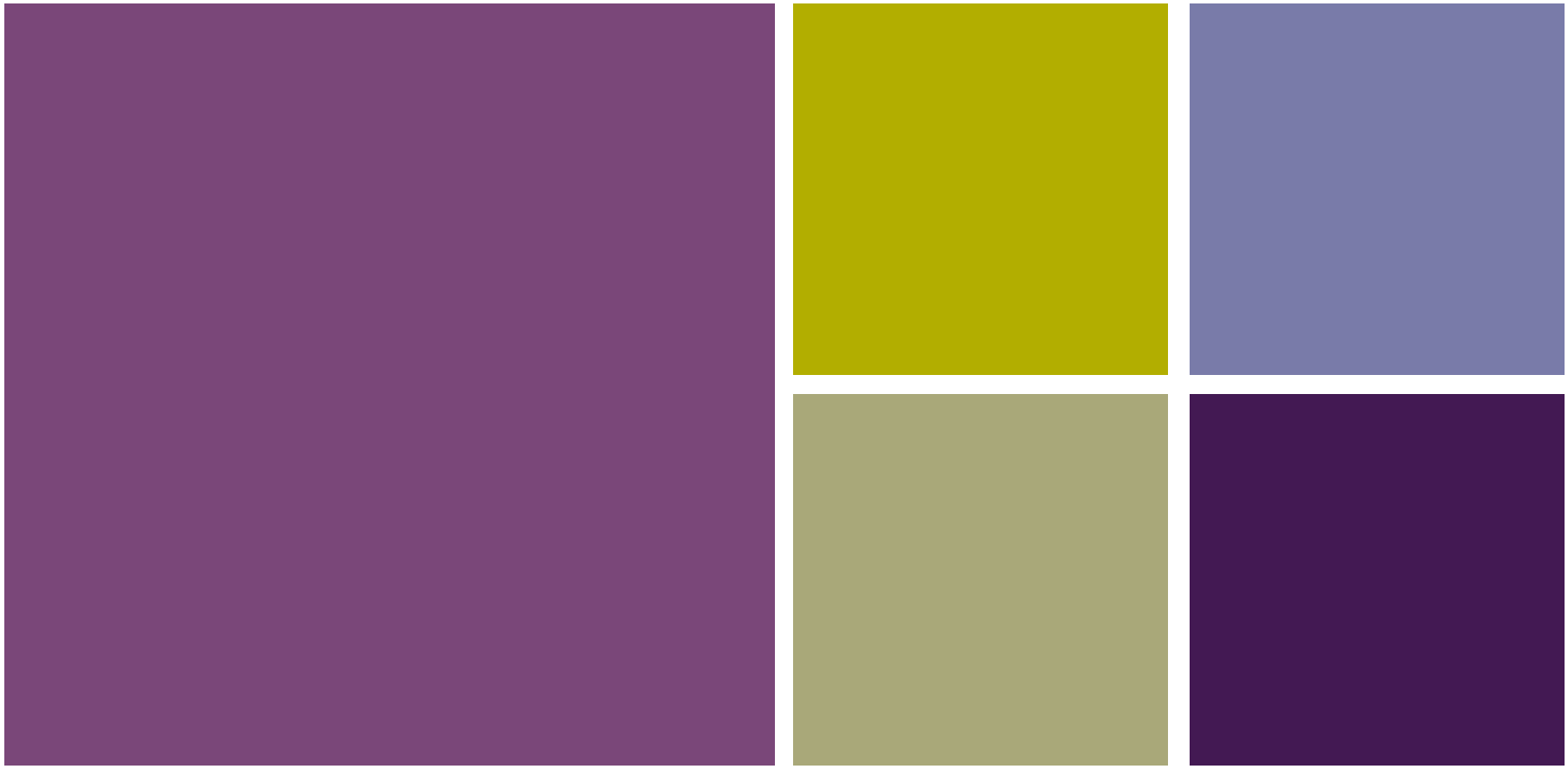


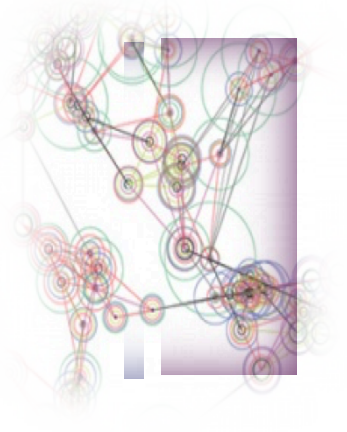
Non-Linear Classifiers 3: Neural Networks

Pattern Recognition and Image Analysis

Dr. Manal Helal – Fall 2014

Lecture 11

CONTENTS



INTRODUCTION

- "Neural" is an adjective for neuron, and "network" denotes a graph like structure.
- Artificial Neural Networks are also referred to as "neural nets" , "artificial neural systems", "parallel distributed processing systems", "connectionist systems".
- For a computing systems to be called by these pretty names, it is necessary for the system to have a labeled directed graph structure where nodes performs some simple computations.
- "Directed Graph" consists of set of "nodes"(vertices) and a set of "connections"(edges/links/arcs) connecting pair of nodes.
- A graph is said to be "labeled graph" if each connection is associated with a label to identify some property of the connection

CONTD...

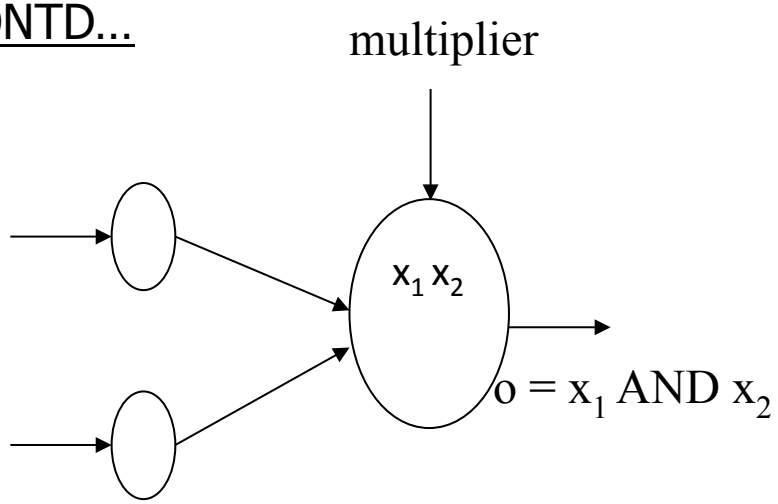


Fig 1: AND gate graph

This graph cannot be considered a neural network since the connections between the nodes are fixed and appear to play no other role than carrying the inputs to the node that computed their conjunction.

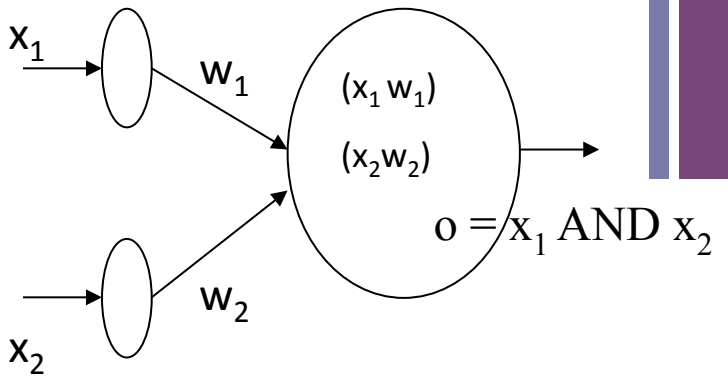


Fig 2: AND gate network

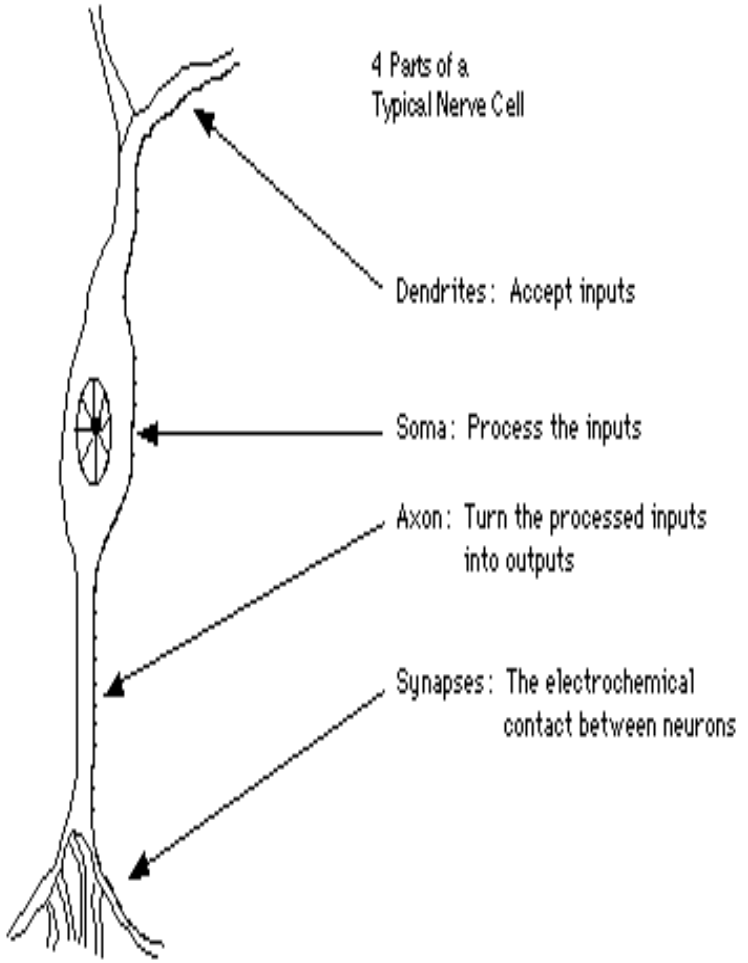
The graph structure which connects the weights modifiable using a learning algorithm, qualifies the computing system to be called an artificial neural networks.

The field of neural network was pioneered by BERNARD WIDROW of Stanford University in 1950's.

BIOLOGICAL NEURON MODEL

Four parts of a typical nerve cell :

- DENDRITES: Accepts the inputs
- SOMA : Process the inputs
- AXON : Turns the processed inputs into outputs.
- SYNAPSES : The electrochemical contact between the neurons.

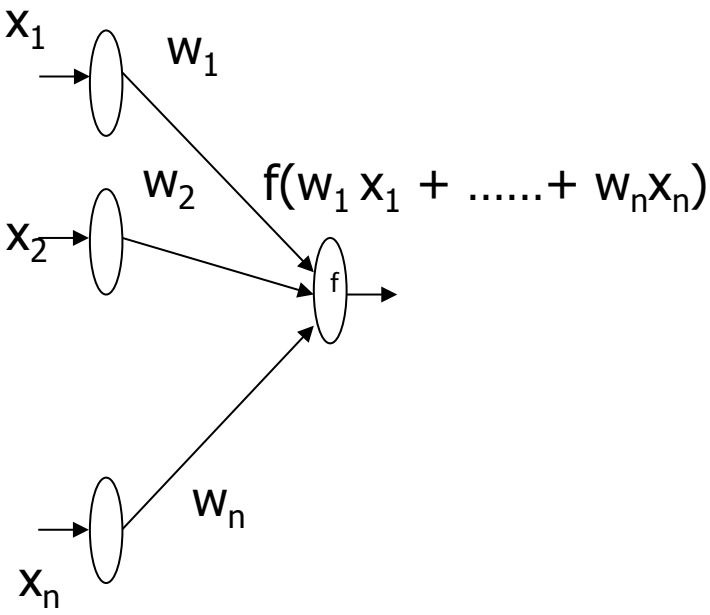


ARTIFICIAL NEURON MODEL

- Inputs to the network are represented by the mathematical symbol, x_n
- Each of these inputs are multiplied by a connection weight , w_n

$$\text{sum} = w_1 x_1 + \ldots\ldots + w_n x_n$$

- These products are simply summed, fed through the transfer function, f( ) to generate a result and then output.



TERMINOLOGY

Biological Terminology	**Artificial Neural Network Terminology**
Neuron	Node/Unit/Cell/Neurode
Synapse	Connection/Edge/Link
Synaptic Efficiency	Connection Strength/Weight
Firing frequency	Node output

ARTIFICIAL NEURAL NETWORK

- **Artificial Neural Network (ANNs)** are programs designed to solve any problem by trying to mimic the structure and the function of our nervous system.

- Neural networks are based on simulated neurons, Which are joined together in a variety of ways to form networks.

- Neural network resembles the human brain in the following two ways: -
  - A neural network acquires knowledge through learning.
  - A neural network's knowledge is stored within the interconnection strengths known as synaptic weight.

ARTIFICIAL NEURAL NETWORK MODEL

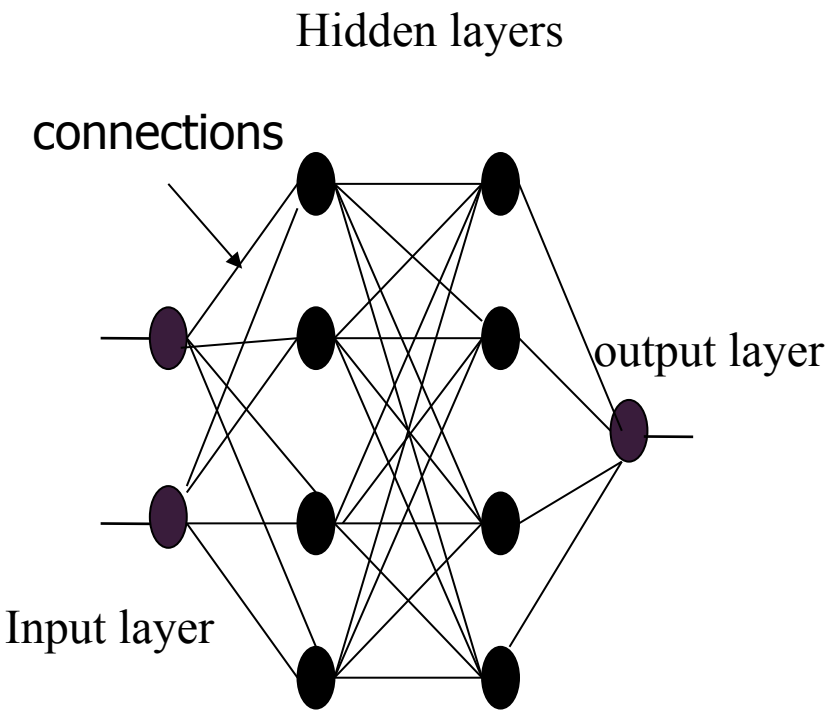


Fig 1 : artificial neural network model

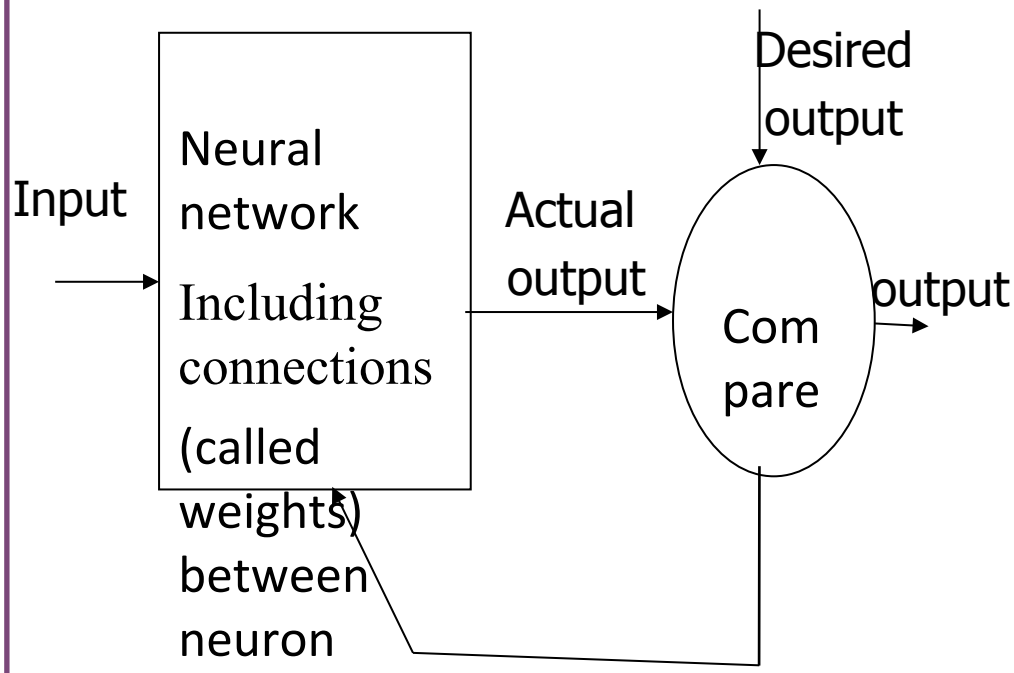


Figure showing adjust of neural network

NEURAL NETWORK ARCHITECTURES

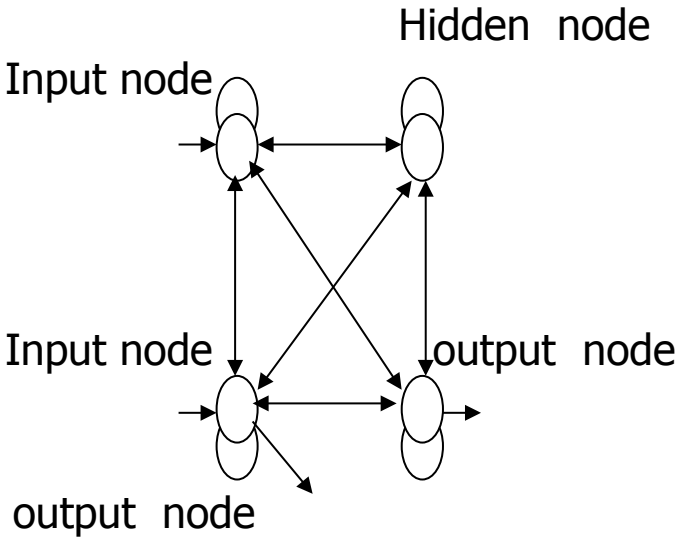


Fig: fully connected network

The neural network in which every node is connected to every other nodes, and these connections may be either excitatory (positive weights), inhibitory (negative weights), or irrelevant (almost zero weights).

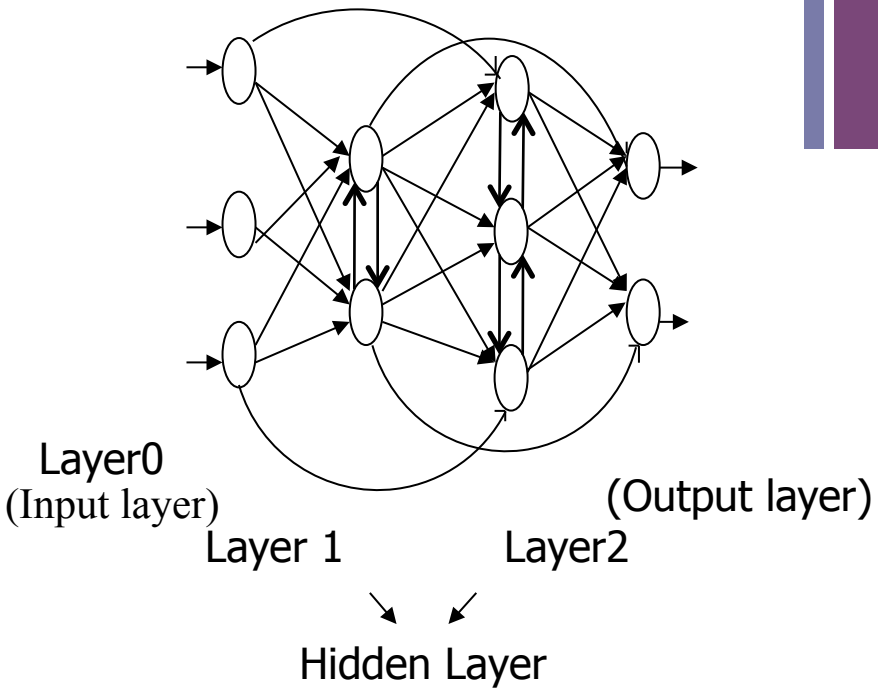


fig: layered network

These are networks in which nodes are partitioned into subsets called layers, with no connections from layer j to k if j > k.

CONTD...

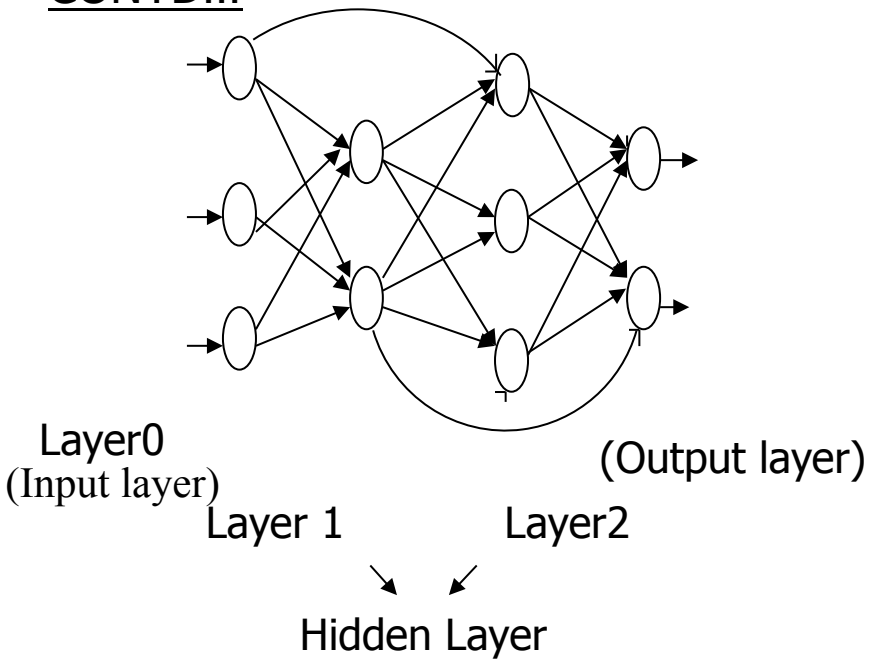


Fig : Acyclic network

This is the subclass of the layered networks in which there is no intra-layer connections. In other words, a connection may exist between any node in layer i and any node in layer j for i < j, but a connection is not allowed for i=j.

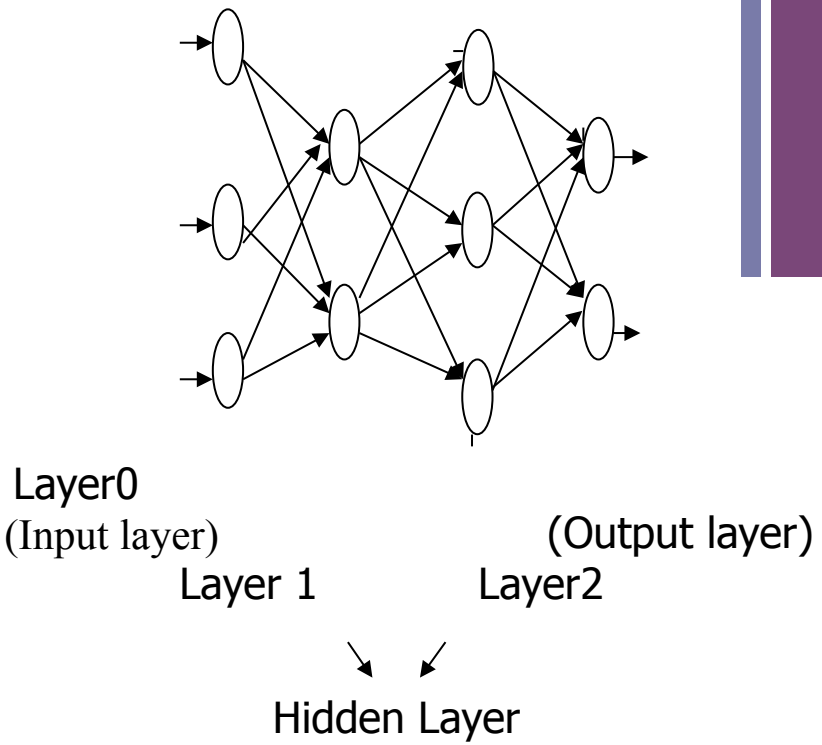


fig : Feedforward network

This is a subclass of acyclic networks in which a connection is allowed from a node in layer i only to nodes in layer i+1

CONTD…

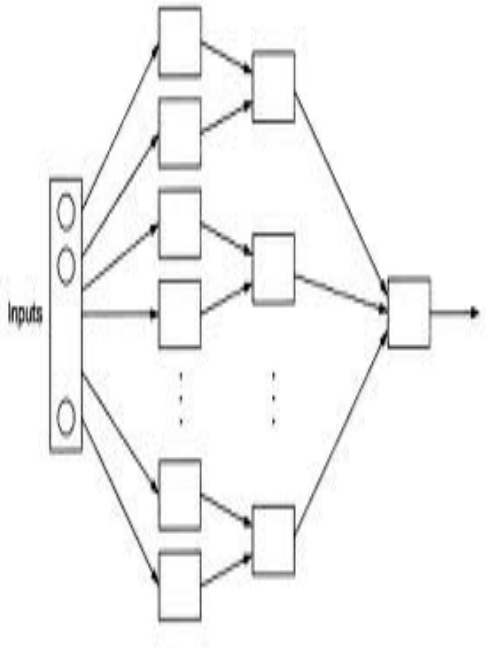
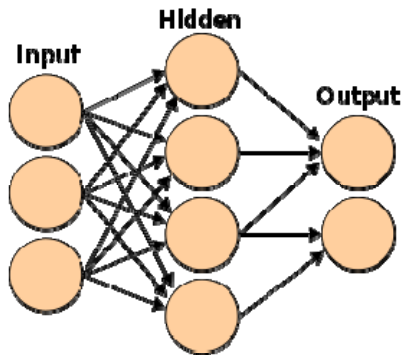
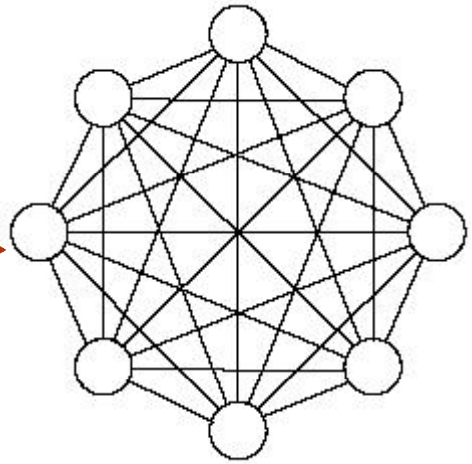


Fig : Modular neural network

Many problems are best solved using neural networks whose architecture consists of several modules, with sparse interconnections between them. Modules can be organized in several different ways as Hierarchial organization, Successive refinement, Input modularity

- Connection patterns
  - Feed-Forward
    - Single layer perceptron
    - Multilayer Perceptron
    - Radial Basis Functions
  - Recurrent
    - Competetive Networks
    - Kohonen's SOM
    - Hopefield Network
    - ART models





LEARNING

- Neurons in an animal's brain are "hard wired". It is equally obvious that animals, especially higher order animals, learn as they grow.
- How does this learning occur?
- What are possible mathematical models of learning?
- In artificial neural networks, learning refers to the method of modifying the weights of connections between the nodes of a specified network.
- The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.

CONTD...

SUPERVISED LEARNING

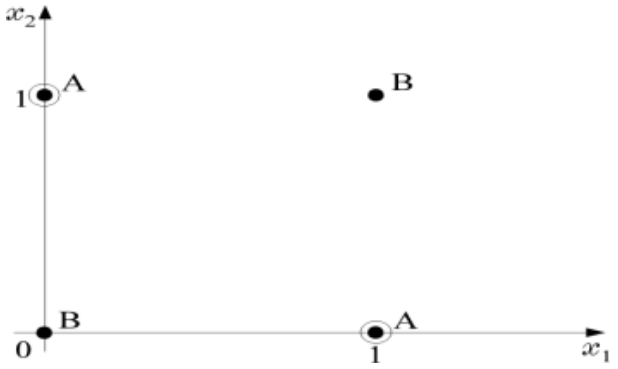
- A teacher is available to indicate whether a system is performing correctly, or to indicate the amount of error in system performance. Here a teacher is a set of training data.
- The training data consist of pairs of input and desired output values that are traditionally represented in data vectors.
- Supervised learning can also be referred as classification, where we have a wide range of classifiers, (Multilayer perceptron, k nearest neighbor..etc)

UNSUPERVISED LEARNING

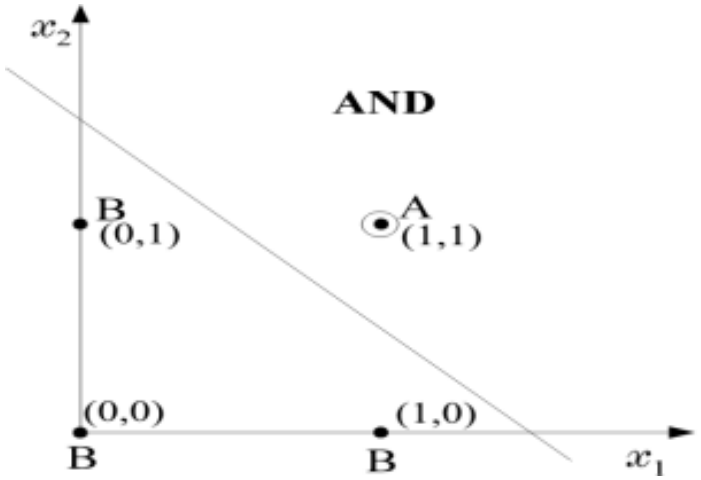
- This is learning by doing.
- In this approach no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs.
- One common form of unsupervised learning is clustering where we try to categorize data in different clusters by their similarity.

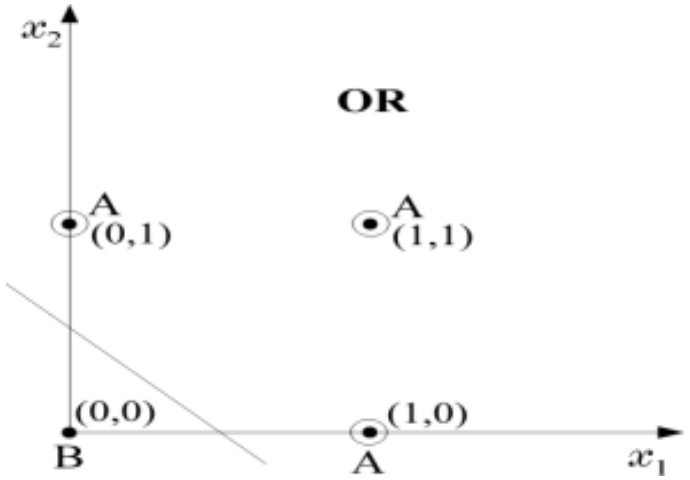
The XOR problem

- There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems.



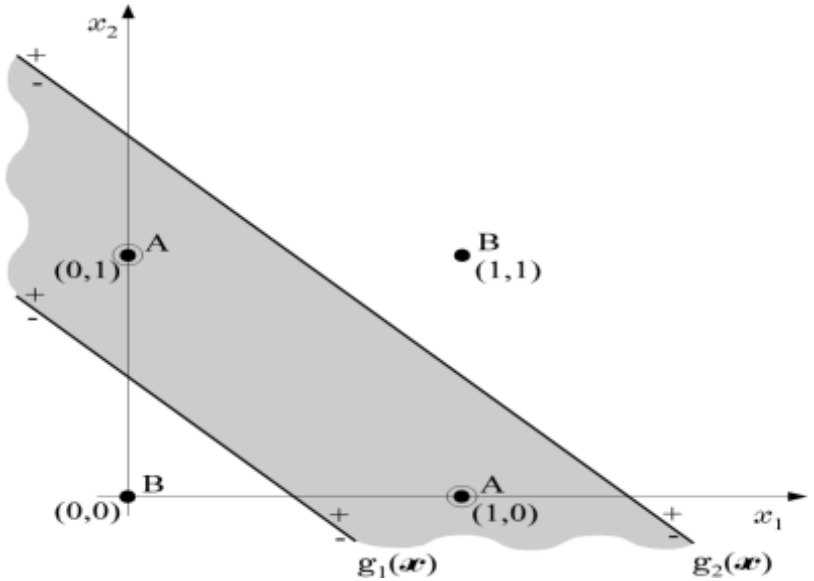
x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B





The Two-Layer Perceptron

- For the XOR problem, draw two lines, instead of one.
- Then class B is outside the shaded area and class is A inside.
- We call it a two-step design.

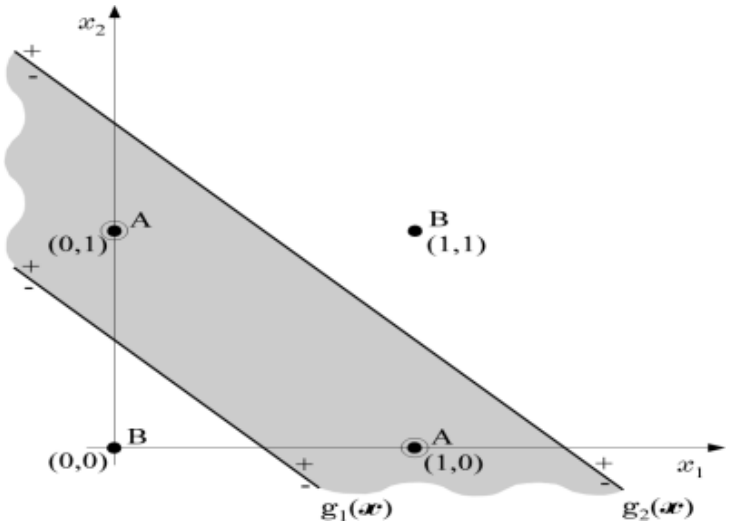


The Two-Layer Perceptron

- **Step 1:** Draw two lines (hyperplanes)

$g_1(x) = 0,$

$g_2(x) = 0$



Each of them is realized by a perceptron.

- The outputs of the perceptrons will be :

$$y_i = f(g_i(x)) = \begin{cases} 0 \\ 1 \end{cases} \quad i = 1,2$$

depending on the value of x (f is the activation function).

- **Step 2:** Find the 'position' of x w.r.t. both lines, based on the values of y_1, y_2.

The Two-Layer Perceptron

- Equivalently:
  1. The computations of the first step perform a mapping

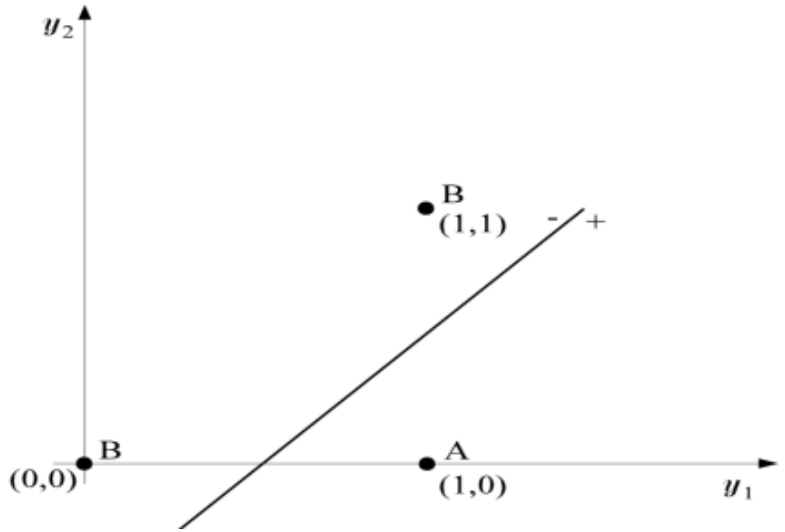
     $$x \rightarrow y = [y_1 , y_2]^T$$

  2. The decision is then performed on the transformed data y.

1st step				2nd step
x_1	x_2	y_1	y_2	
0	0	0(-)	0(-)	B(0)
0	1	1(+)	0(-)	A(1)
1	0	1(+)	0(-)	A(1)
1	1	1(+)	1(+)	B(0)

The Two-Layer Perceptron

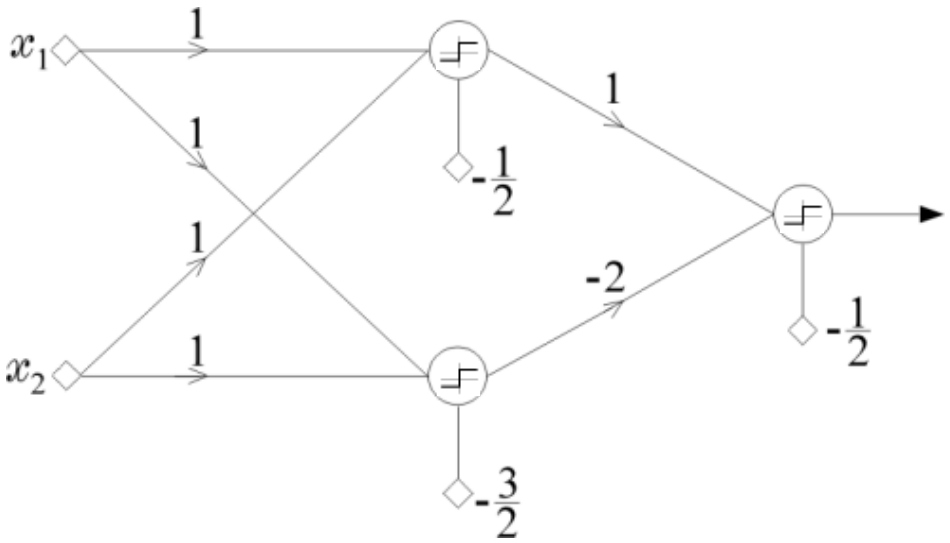
- This decision can be performed via a second line $g(y) = 0$, which can also be realized by a perceptron.



- Computations of the first step perform a mapping that transforms the nonlinearly separable problem to a linearly separable one.

The Two-Layer Perceptron

- The architecture



- This is known as the two layer perceptron with one hidden and one output layer.
  The activation functions are:

$$f(.) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The Two-Layer Perceptron

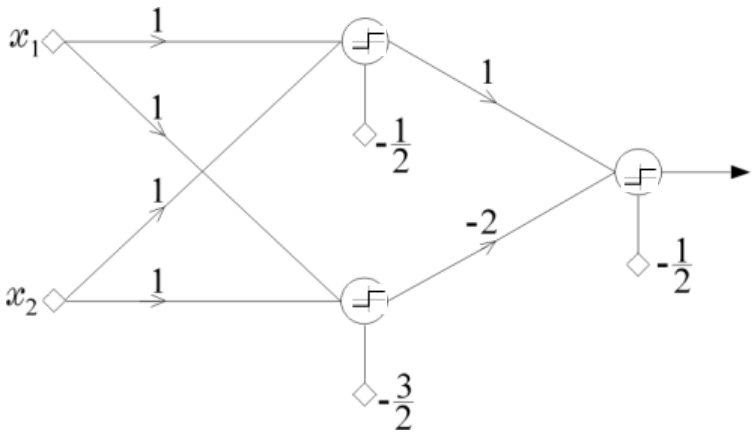
- The nodes (neurons) of the figure realize the following lines (hyper planes).

$$g_1(\underline{x}) = 1x_1 + 1x_2 - \frac{1}{2} = 0$$

$$g_2(\underline{x}) = 1x_1 + 1x_2 - \frac{3}{2} = 0$$

$$g_3(\underline{y}) = 1y_1 - 2y_2 - \frac{1}{2} = 0$$

$$f(.) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$



- Classification capabilities:

- All possible mappings performed by the first layer are onto the vertices of the unit side square,
  e.g., (0, 0), (1, 0), (1, 0), (1, 1).

Classification Capabilities

- The more general case

$$\underline{x} \in R^l,$$

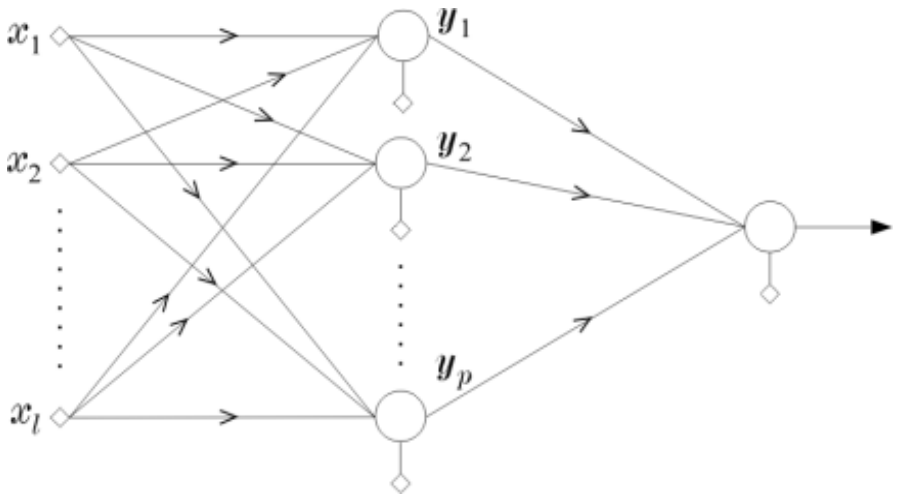
$$y_i \in \{0,\ 1\} \qquad i = 1,\ 2, \ldots p$$

$$\underline{x} \rightarrow \underline{y} = [y_1, \ldots y_p]^T, \quad \underline{y} \in R^p$$

$$y_i = f(g_i)$$

$$g_i(\underline{x}) = \sum_{k=1}^{l} w_{ik} x_k + w_{i0} = 0$$

$$g_j(\underline{y}) = \sum_{k=1}^{p} w_{jk} y_k + w_{j0} = 0$$



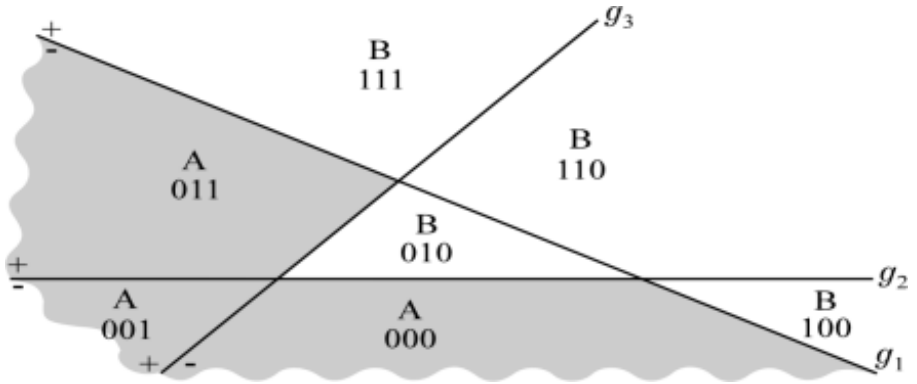
$$g_j(\underline{y}) = \underline{w}_j^T \underline{y} + w_{j0} = 0 \qquad \underline{w}_j,\ \underline{y} \in R^p$$

$$g_i(\underline{x}) = \underline{w}_i^T \underline{x} + w_{i0} = 0 \qquad \underline{w}_i,\ \underline{x} \in R^l$$

- Performs a mapping of a vector onto the vertices of the unit side H_p hypercube.

Classification Capabilities

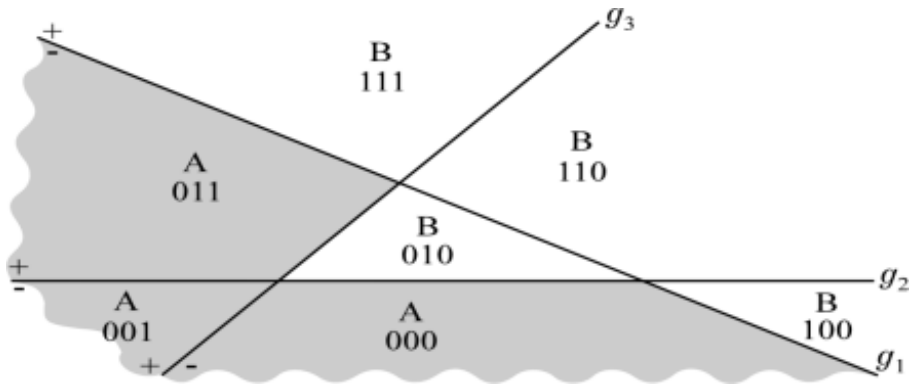
- The mapping is achieved with p nodes each realizing a hyperplane. The output of each of these nodes is 0 or 1 depending on the relative position of x w.r.t. the hyperplane.



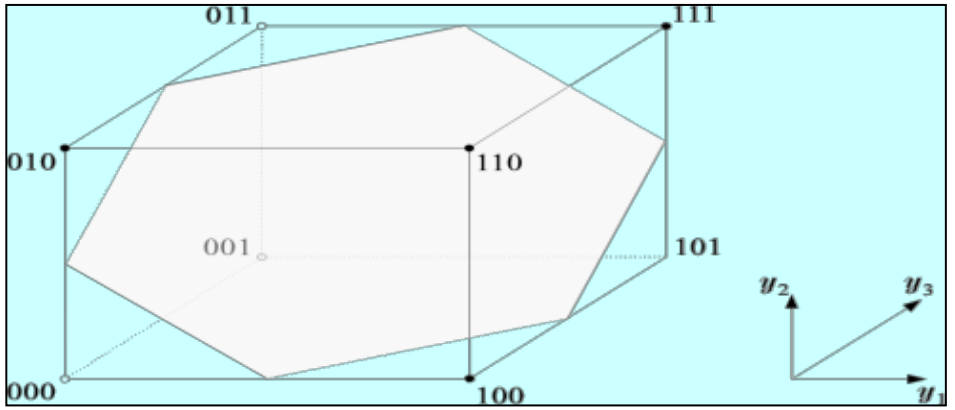
- Intersections of these hyperplanes form regions in the l-dimensional space. Each region corresponds to a vertex of the H_p unit hypercube.

Classification Capabilities

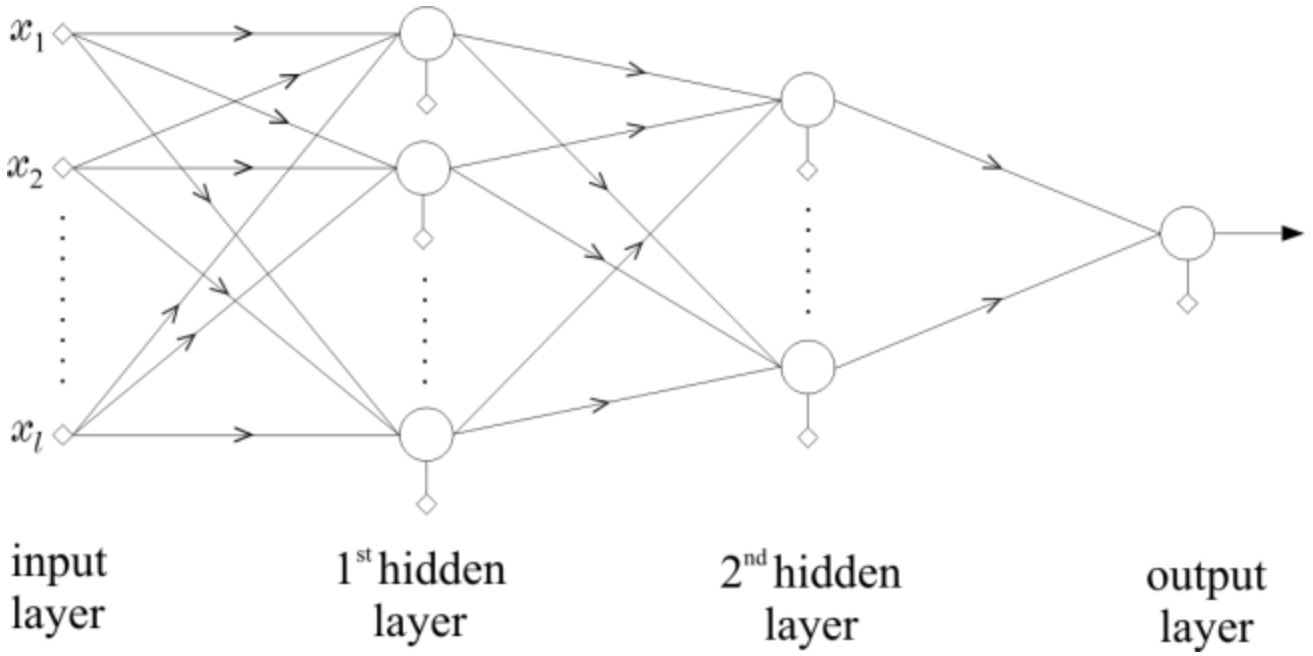
- For example, the 001 vertex corresponds to the region which is located
  - to the (-) side of $g_1(x) = 0$
  - to the (-) side of $g_2(x) = 0$
  - to the (+) side of $g_3(x) = 0$



- The output node realizes a hyperplane in the y space, that separates some of the vertices from the others. Thus, the two layer perceptron has the capability to classify vectors into classes that consist of unions of polyhedral regions. But not ANY union. It depends on the relative position of the corresponding vertices.



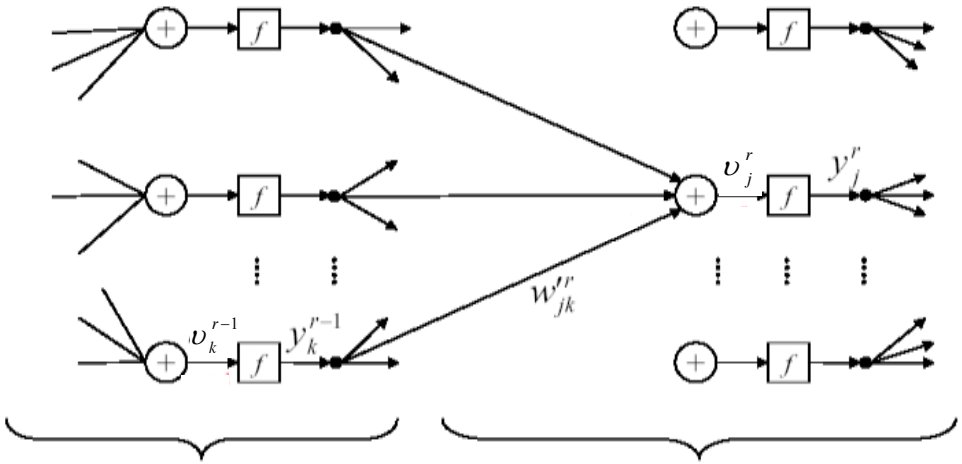
The Three-Layer Perceptron



- This is capable to classify vectors into classes consisting of ANY union of polyhedral regions.
- The idea is similar to the XOR problem. It realizes more than one plane in the space.

- The reasoning
  - For each vertex, corresponding to class A, construct a hyperplane which leaves THIS vertex on one side (+) and ALL the others to the other side (-).
  - The output neuron realizes an OR gate.
- Overall:
  - The first layer of the network forms the hyperplanes, the second layer forms the regions and the output nodes forms the classes.

The Multi-Layer Neural Network



- For the i-th trainings pair

$y_k^{r-1}(i)$ output of the k-th node at layer r-1

$\upsilon_j^r(i)$ argument for $f(.)$ for the i-th trainings pair

$$\upsilon_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{j0}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i), \quad with \quad y_0^r(i) \equiv +1$$

$$\upsilon_j^r(i) = \underline{w}_j^r \underline{y}^{r-1}(i)$$

$$y_j^r(i) = f\left(\upsilon_j^r(i)\right) = f\left(\underline{w}_j^r \underline{y}^{r-1}(i)\right)$$

Back propagation algorithm to train multilayer perceptrons

- Designing Multilayer Perceptrons
  - One direction is to adopt the above rationale and develop a structure that classifies correctly all the training patterns.
  - The other direction is to choose a structure and compute the *w*'s, often called 'synaptic weights', to optimize a cost function.
  - BP is an algorithmic procedure that computes the synaptic weights iteratively, so that an adopted cost function is minimized (optimized).

The Backpropagation Algorithm

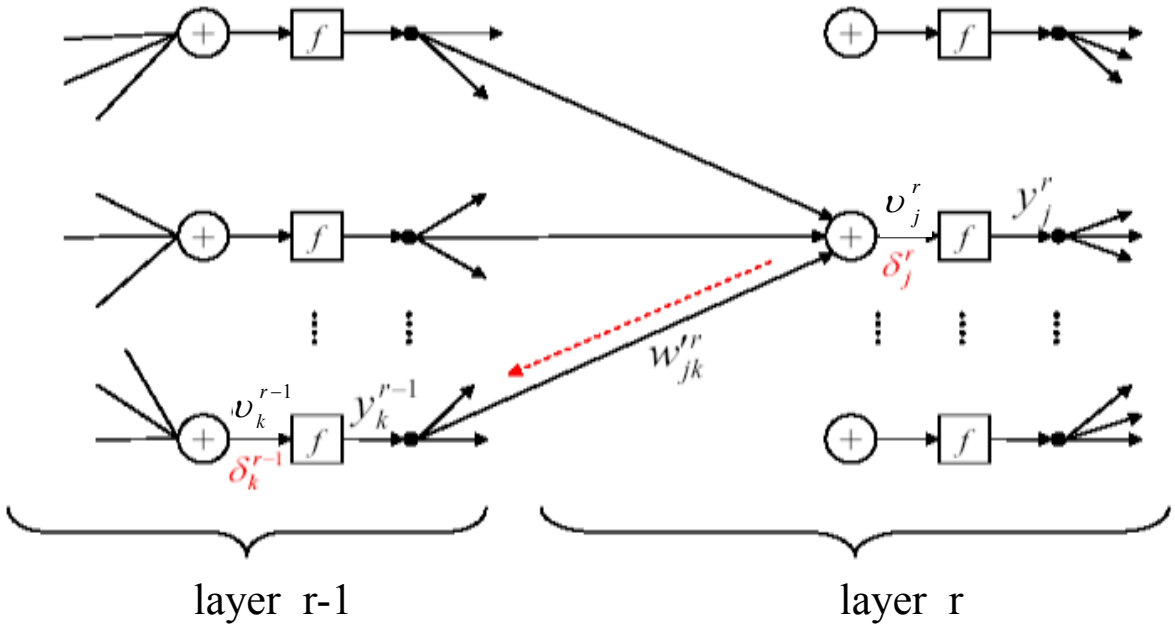
- The Steps:
  1. Adopt an optimizing cost function J(i), e.g.,
     - Least Squares Error
     - Relative Entropy

between desired responses and actual responses of the network for the available training patterns.

- That is, from now on we have to live with errors. We only try to minimize them, using certain criteria.

The Backpropagation Algorithm

- The Steps:
  2. Adopt an algorithmic procedure for the optimization of the cost function with respect to the weights w e.g.:
     - Gradient descent
     - Newton's algorithm
     - Conjugate gradient



The Backpropagation Algorithm

- The Steps:
  3. The task is a nonlinear optimization e.g. with gradient descent.

$$\underline{w}_j^r(\text{new}) = \underline{w}_j^r(\text{old}) + \Delta \underline{w}_j^r$$

$$\Delta \underline{w}_j^r = -\mu \frac{\partial J}{\partial \underline{w}_j^r}$$

$$J = \sum_{i=1}^{N} E(i)$$

The Backpropagation Algorithm Example

- The activation function of the artificial neurons in ANNs implementing the backpropagation algorithm is a weighted sum (the sum of the inputs x_i multiplied by their respective weights w_{ji})

$$A_j(\bar{x}, \bar{w}) = \sum_{i=0}^{n} x_i w_{ji}$$

- The most common output function is the sigmoidal function:

$$O_j(\bar{x}, \bar{w}) = \frac{1}{1 + e^{A(\bar{x}, \bar{w})}}$$

- Since the error is the difference between the actual and the desired output, the error depends on the weights, and we need to adjust the weights in order to minimize the error. We can define the error function for the output of each neuron:

$$E_j(\bar{x}, \bar{w}, d) = \left(O_j(\bar{x}, \bar{w}) - d_j\right)^2$$

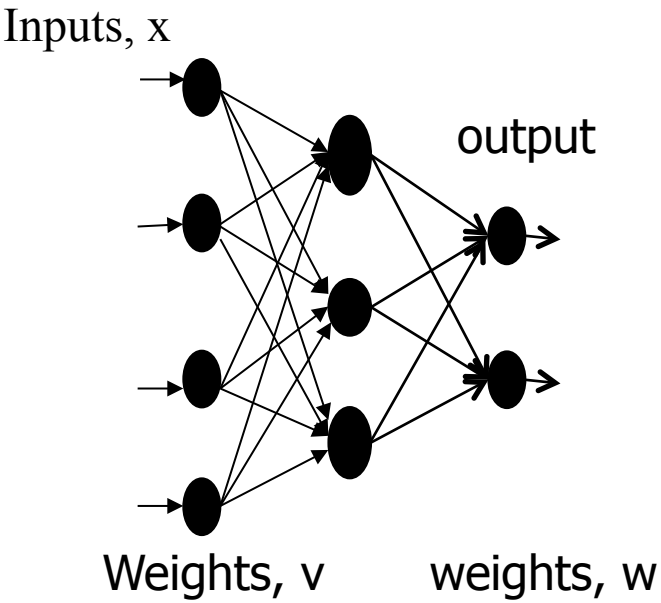


Fig: Basic Block of Back propagation neural network

CONTD...

- The backpropagation algorithm now calculates how the error depends on the output, inputs, and weights.

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

the adjustment of each weight (Δw_{ji}) will be the negative of a constant eta (η) multiplied by the dependance of the "w_{ji}" previous weight on the error of the network.

- First, we need to calculate how much the error depends on the output

$$\frac{\partial E}{\partial O_j} = 2\left(O_j - d_j\right)$$

- Next, how much the output depends on the activation, which in turn depends on the weights

$$\frac{\partial O_j}{\partial w_{ji}} = \frac{\partial O_j}{\partial A_j}\frac{\partial A_j}{\partial w_{ji}} = O_j(1-O_j)x_i$$

- And so, the adjustment to each weight will be

$$\Delta w_{ji} = -2\eta\left(O_j - d_j\right)O_j(1-O_j)x_i$$

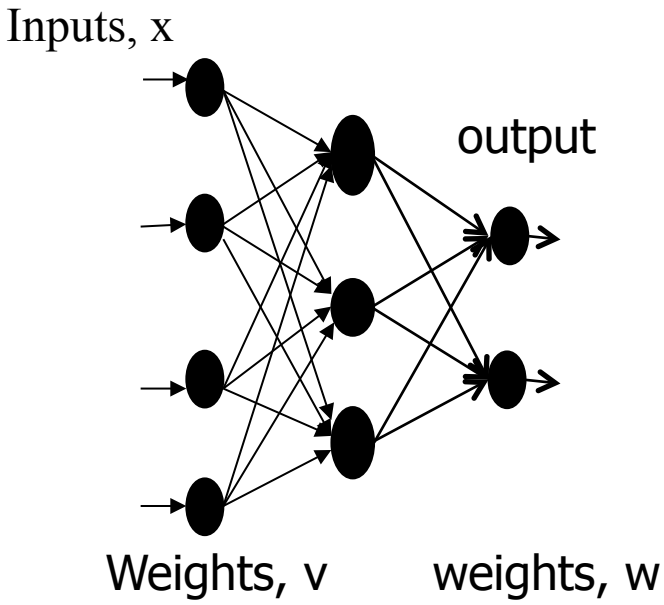
CONTD…

- If we want to adjust v_{ik}, the weights (let's call them v_{ik}) of a previous layer, we need first to calculate how the error depends not on the weight, but in the input from the previous layer i.e. replacing w by x as shown in the equation:

$$\Delta v_{ik} = -\eta \frac{\partial E}{\partial v_{ik}} = -\eta \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial v_{ik}}$$

where $$\frac{\partial E}{\partial w_{ji}} = 2\left(O_j - d_j\right) O_j (1 - O_j) w_{ji}$$

and $$\frac{\partial x_i}{\partial v_{ik}} = x_i (1 - x_i) v_{ik}$$



The Backpropagation Algorithm

- The Procedure:

1. **Initialization:**

Initialize unknown weights randomly with small values.

2. **Forward computations:**

For each of the training examples compute the output of all neurons of all layers. Compute the cost function for the current estimate of weights.

3. **Backward computations:**

Compute the gradient terms backwards, starting with the weights of the last (e.g. 3rd) layer and then moving towards the first.

4. **Update:** Update the weights.

5. **Termination:**

Repeat until a termination procedure is met

The Backpropagation Algorithm

- In a large number of optimizing procedures, computation of derivatives are involved. Hence, discontinuous activation functions pose a problem, i.e.,

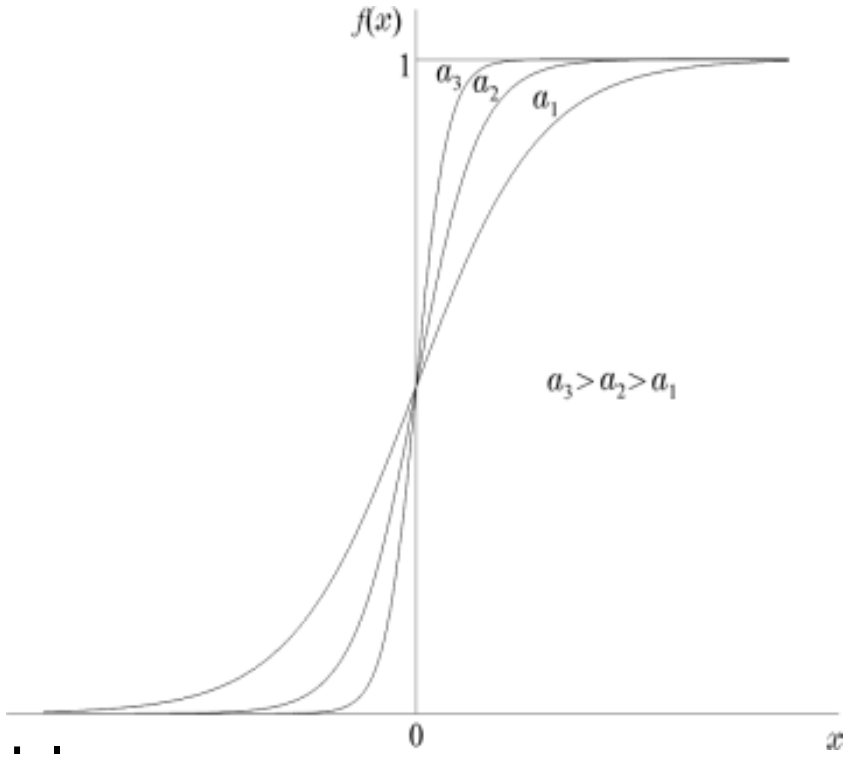
$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

- There is always an escape path!!! e.g. the logistic function:

$$f(x) = \frac{1}{1 + \exp(-ax)}$$

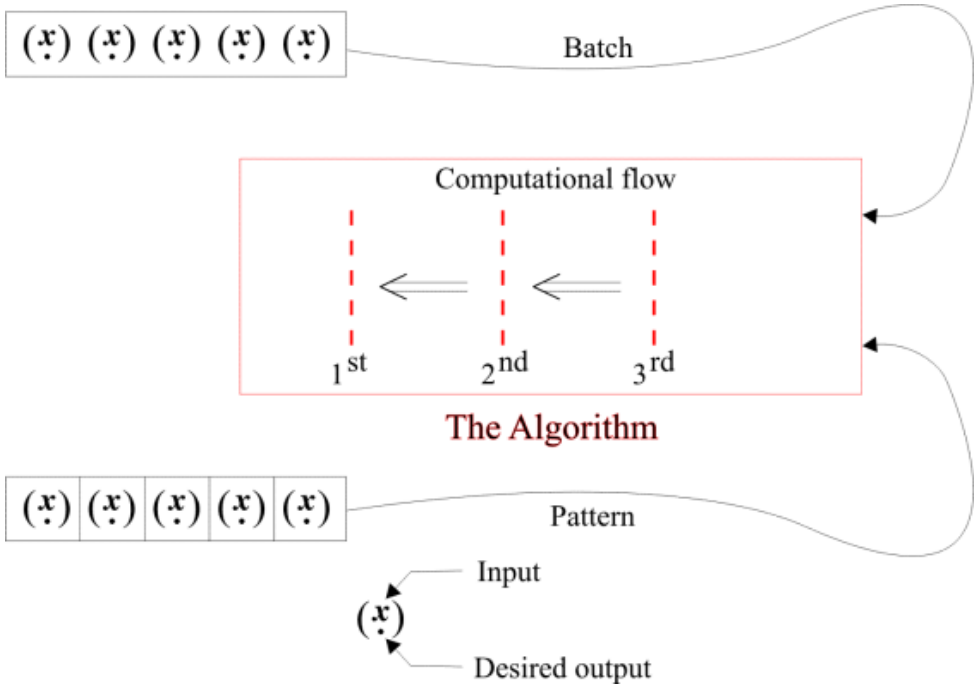
$$f'(x) = \alpha f(x)\left(1 - f(x)\right)$$

- Other differentiable functions are also possible and in some cases more desirable.



The Backpropagation Algorithm

- Two major philosophies:
  - **Batch mode:** The gradients of the last layer are computed once ALL training data have appeared to the algorithm, i.e., by summing up all error terms.
  - **Pattern mode:** The gradients are computed every time a new training data pair appears. Thus gradients are based on successive individual errors.

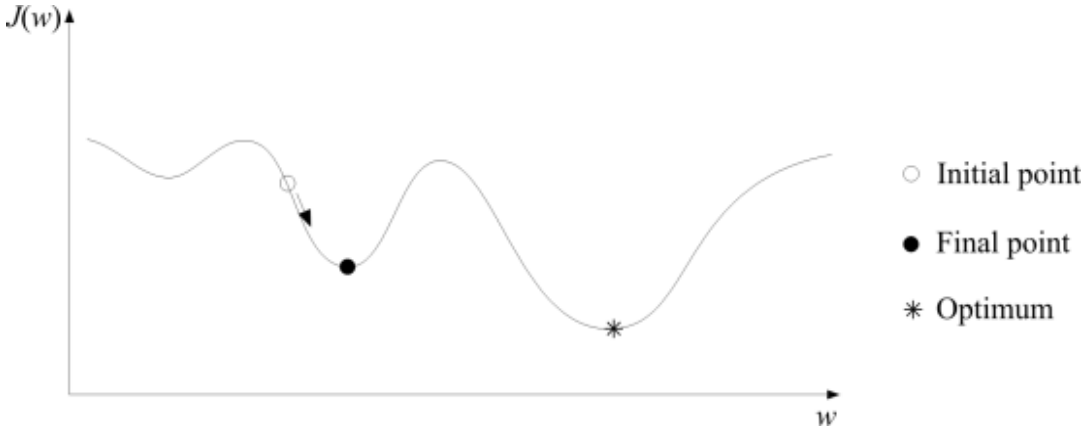


The Backpropagation Algorithm

- A major problem: The algorithm may converge to a local minimum.

The cost function choice Examples:

- The Least Squares



$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \frac{1}{2}\sum_{m=1}^{k} e_m^2(i) = \frac{1}{2}\sum_{m=1}^{k} (y_m(i) - \hat{y}_m(i))^2 \qquad i = 1, 2, \ldots, N$$

$\hat{y}_m(i)$: Desired response of m-th output node (1 or 0) for input $x(i)$.

$y_m(i)$: Actual response of m-th output node, in the interval [0, 1], for input $x(i)$.

The Backpropagation Algorithm

- The cost function choice Examples:
  - The cross-entropy

$$J = \sum_{i=1}^{N} E(i)$$

$$E(i) = \sum_{m=1}^{k} \left\{ \; y_m(i) \ln \hat{y}_m(i) \; + \; (1 - y_m(i)) \ln(1 - \hat{y}_m(i)) \; \right\}$$

  - This presupposes an interpretation of y and $\hat{y}$ as probabilities!
  - Classification error rate:
    - Also known as discriminative learning.
    - Most of these techniques use a smoothed version of the classification error.

The Backpropagation Algorithm

- "Well formed" cost functions :
  - Danger of local minimum convergence.
  - "Well formed" cost functions guarantee convergence to a "good" solution.
  - That is one that classifies correctly ALL training patterns, provided such a solution exists.
  - The cross-entropy cost function is a well formed one. The Least Squares is not.

The Backpropagation Algorithm

- optimally class a-posteriori probabilities:

Both, the Least Squares and the cross entropy lead to output values $\hat{y}_m$ (i) that approximate optimally class a- posteriori probabilities!

$\hat{y}_m(i) \cong P(\omega_m\ x(i))$

- That is, the probability of class ω_m given $x(i)$.

- It does not depend on the underlying distributions!!!

It is a characteristic of certain cost functions and the chosen architecture of the network. It depends on the model how good or bad the approximation is.

- It is valid at the global minimum.

Choice of the network size

- How big a network can be. How many layers and how many neurons per layer?
- There are two major techniques:
  - Pruning Techniques:
    - These techniques start from a large network and then weights and/or neurons are removed iteratively, according to a criterion.
  - Constructive techniques:
    - They start with a small network and keep increasing it, according to a predetermined procedure and criterion.

Choice of the network size

- Pruning Techniques:
  - Methods based on parameter sensitivity

$$\delta J = \sum_i g_i \delta w_i + \frac{1}{2}\sum_i h_{ii}\delta w_i^2 + \frac{1}{2}\sum_i \sum_j h_{ij}\delta w_i \delta w_j$$

  - + higher order terms where

$$g_i = \frac{\partial J}{\partial w_i}, \quad h_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

  - Near a minimum and assuming

$$\delta J \cong \frac{1}{2}\sum_i h_{ii}\delta w_i^2$$

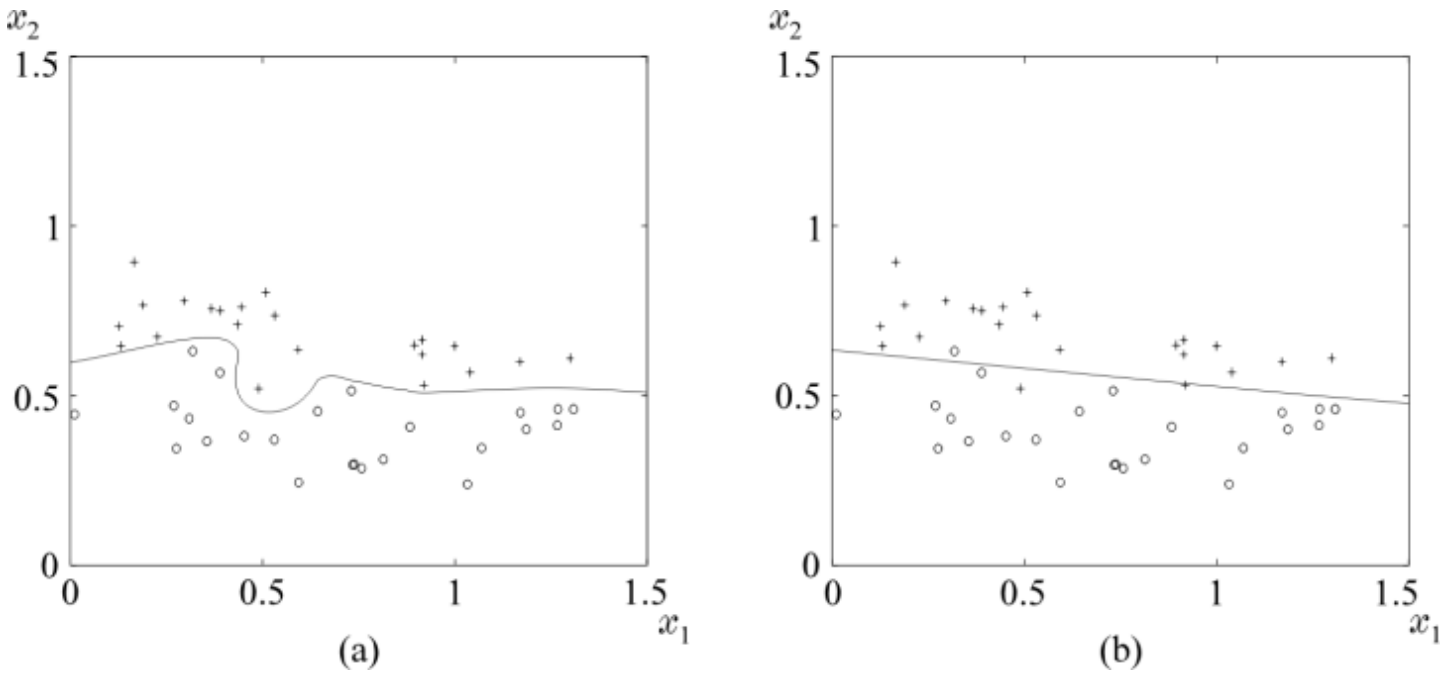
  - Pruning is now achieved as:
    1. Train the network using Backpropagation for a number of steps
    2. Compute the saliencies
    3. Remove weights w_i with small S_i.
    4. Repeat the process

Choice of the network size

- Idea: Start with a large network and leave the algorithm to decide which weights are small.
- Generalization properties:
  - Large network learn the particular details of the training set.
  - Not be able to perform well when presented with data unknown to it.
- The size of the network must be:
  - **Large enough** to learn what makes data of the same class similar and data from different classes dissimilar.
  - **Small enough** not to be able to learn underlying differences between data of the same class. This leads to the so called overfitting.

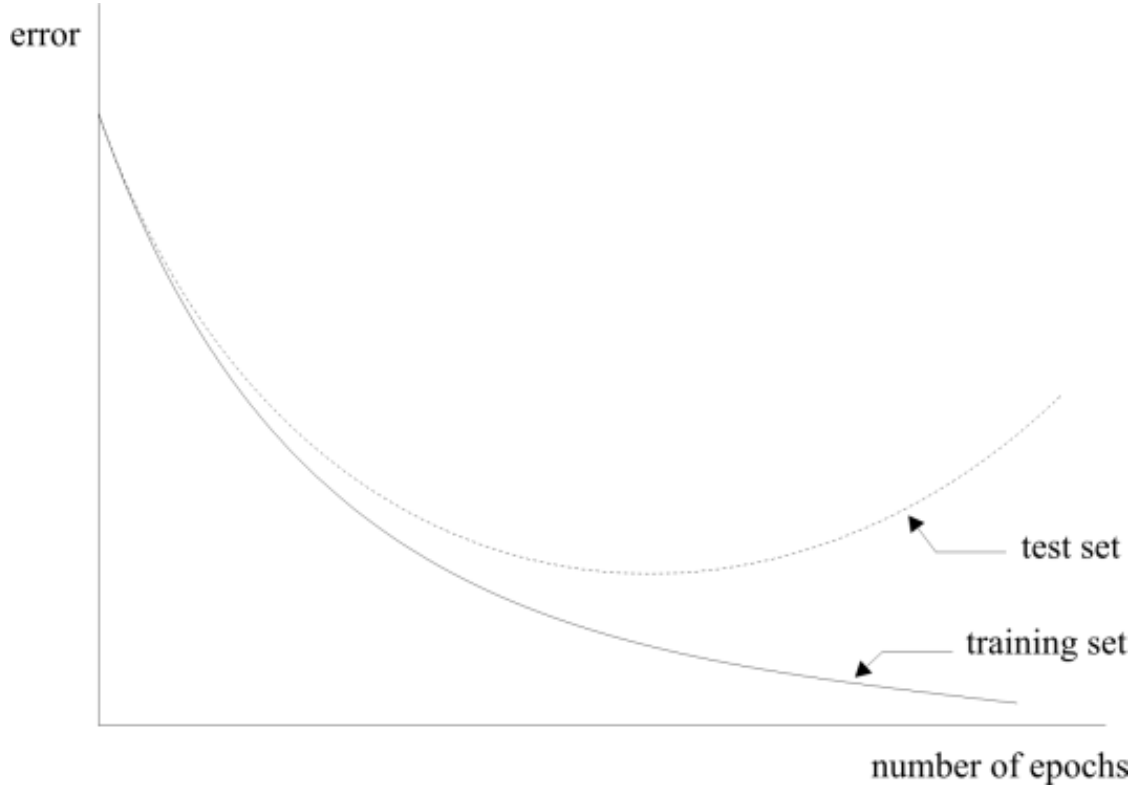
Choice of the network size

- Example:
- Decision curve (a) before and (b) after pruning.

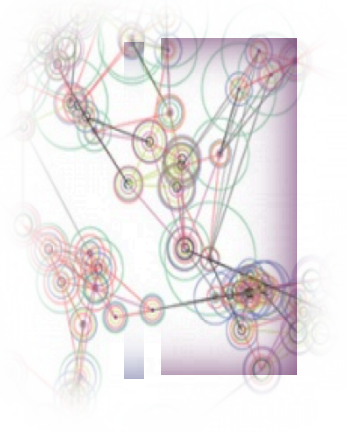


Choice of the network size

- Overtraining is another side of the same coin, i.e., the network adapts to the peculiarities of the training set.



ADVANTAGES

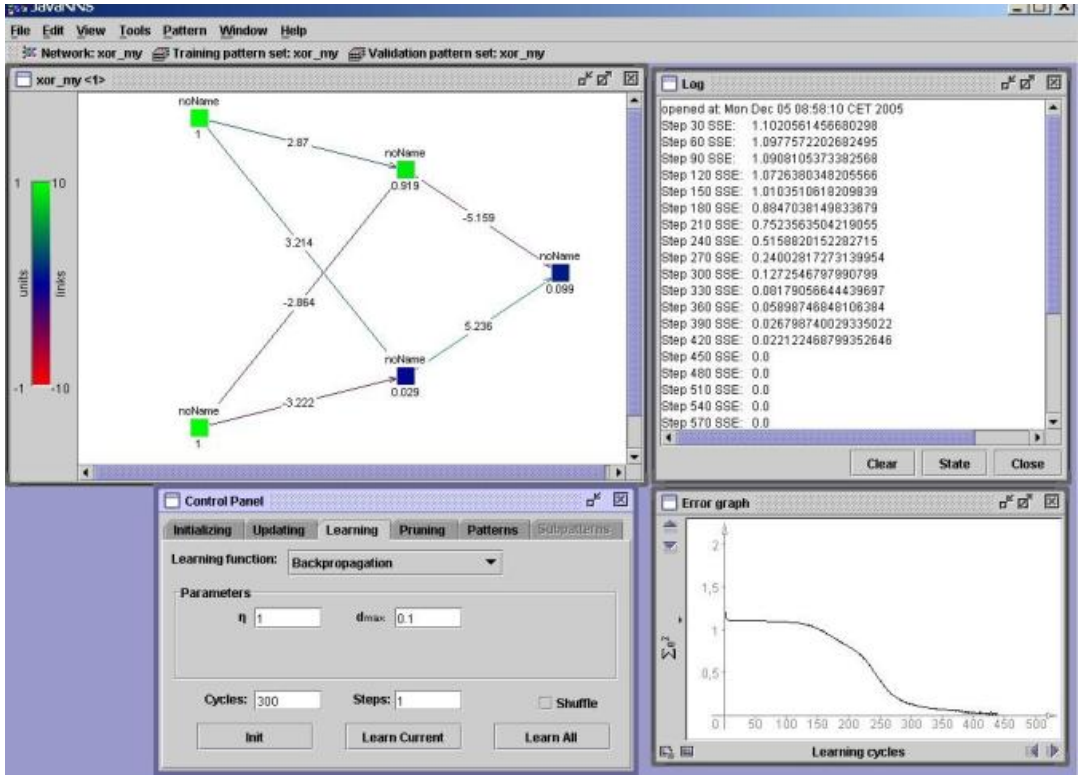


- It involves human like thinking.
- They handle noisy or missing data.
- They can work with large number of variables or parameters.
- They provide general solutions with good predictive accuracy.
- System has got property of continuous learning.
- They deal with the non-linearity in the world in which we live.

Non-Linear Classifiers: Conclusion

- Applications: XOR, ZIP Code, OCR problem
- Demo: Java-NNS, BPN

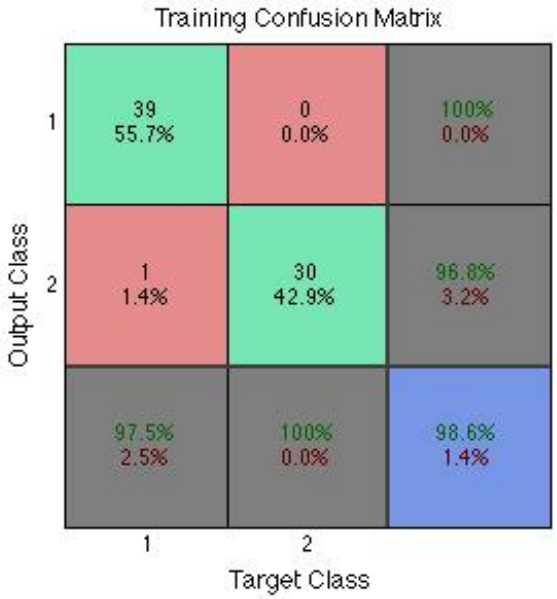
http://www-ra.informatik.uni-tuebingen.de/downloads/JavaNNS/

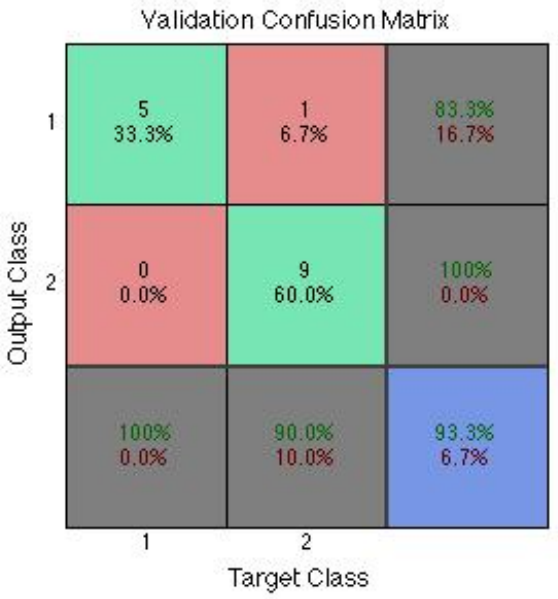


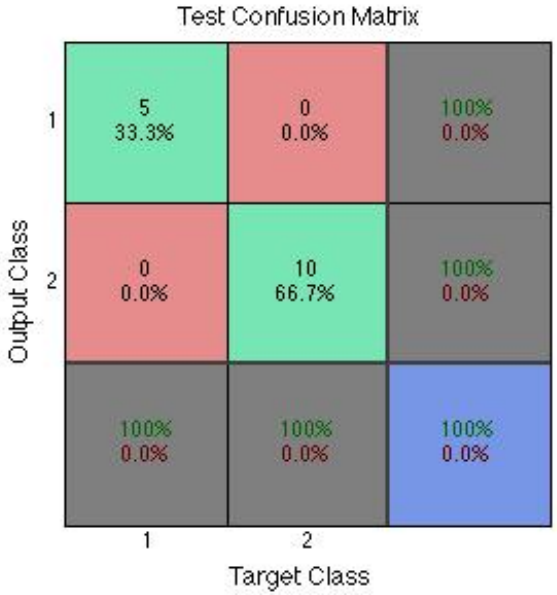
NN Matlab code

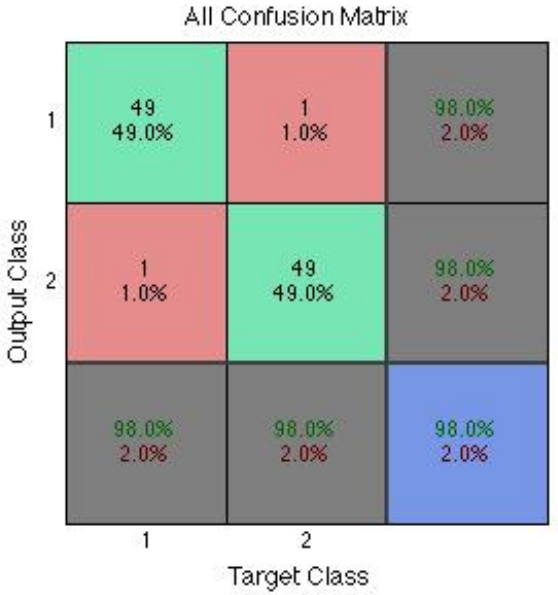
- For the Exam dataset:

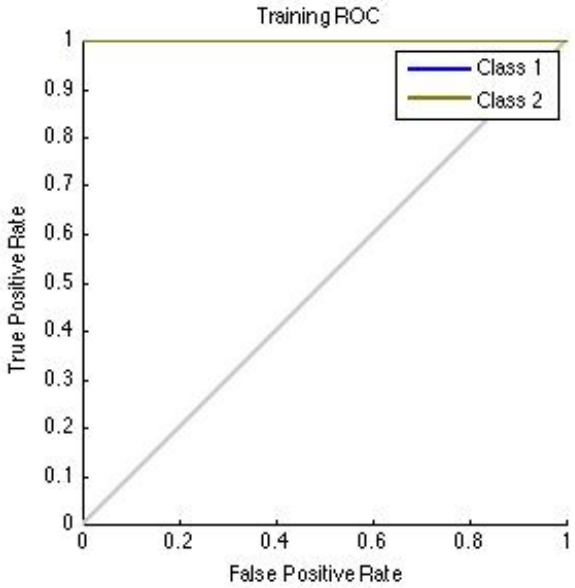
```
xtrain = Dataset(:,1:2)

ytrain = [Dataset(:, 3) ==1, Dataset(:, 3)==2]

plot(xtrain(1,:),xtrain(2,:),'+')

net = patternnet(10);

net = train(net,xtrain,ytrain);

view(net)

y = net(xtrain)

plotconfusion(ytrain,y)
```

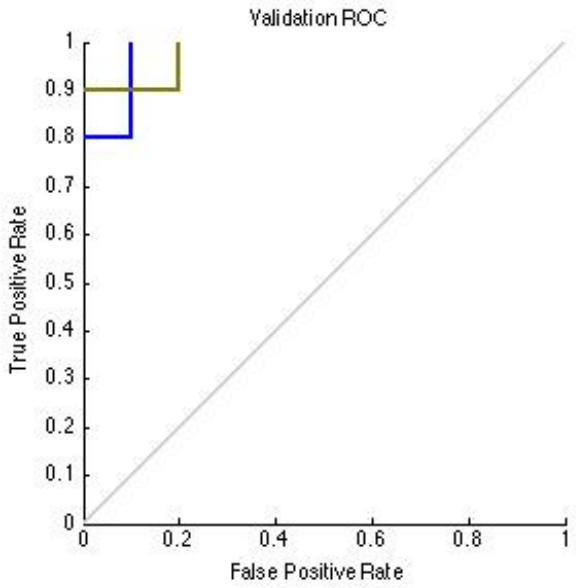

Training Confusion Matrix
39
55.7%
0
0.0%
100%
0.0%
1
1.4%
30
42.9%
96.8%
3.2%
97.5%
2.5%
100%
0.0%
98.6%
1.4%
1
2
Output Class
1
2
Target Class

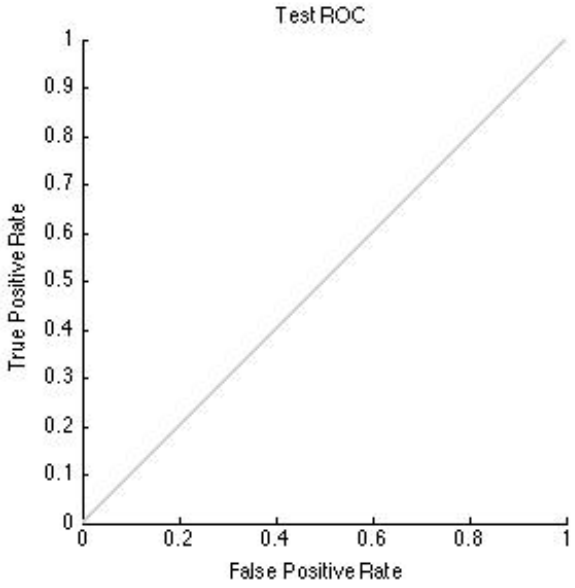

Validation Confusion Matrix
5
33.3%
1
6.7%
83.3%
16.7%
0
0.0%
9
60.0%
100%
0.0%
100%
0.0%
90.0%
10.0%
93.3%
6.7%
1
2
Output Class
1
2
Target Class

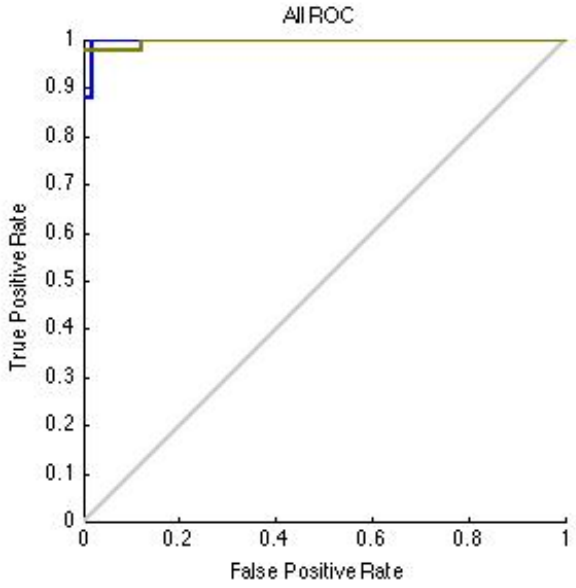

Test Confusion Matrix
5
33.3%
0
0.0%
100%
0.0%
0
0.0%
10
66.7%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Output Class
1
2
Target Class


All Confusion Matrix
49
49.0%
1
1.0%
98.0%
2.0%
1
1.0%
49
49.0%
98.0%
2.0%
98.0%
2.0%
98.0%
2.0%
98.0%
2.0%
1
2
Output Class
1
2
Target Class

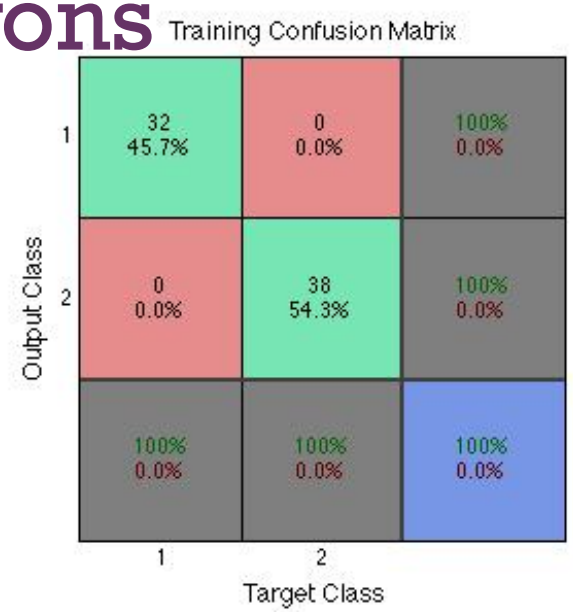

Training ROC
Class 1
Class 2
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

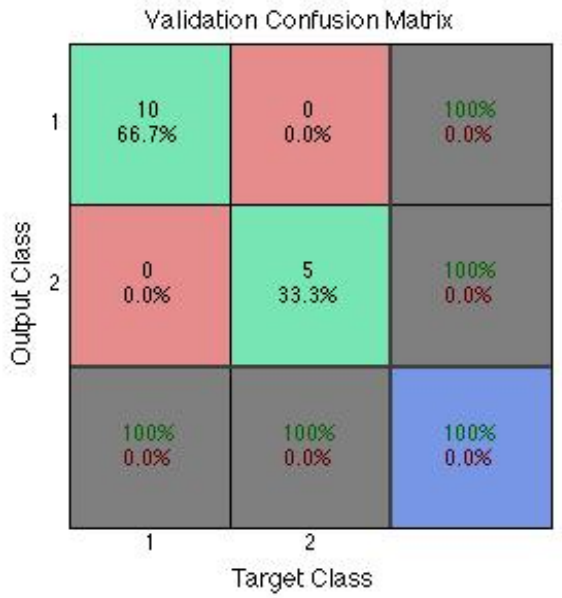

Validation ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

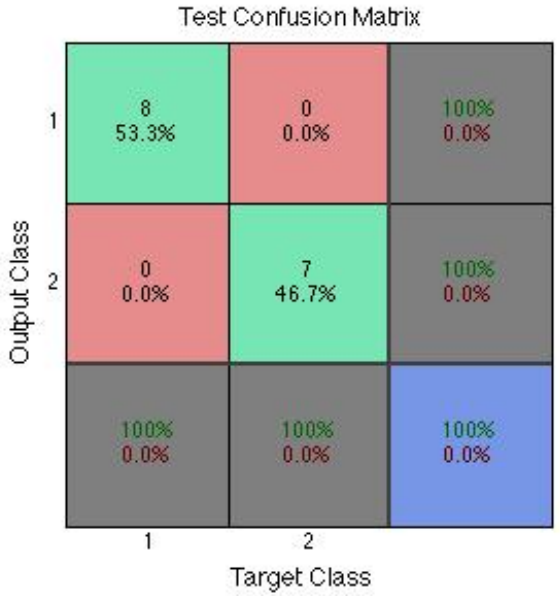

Test ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

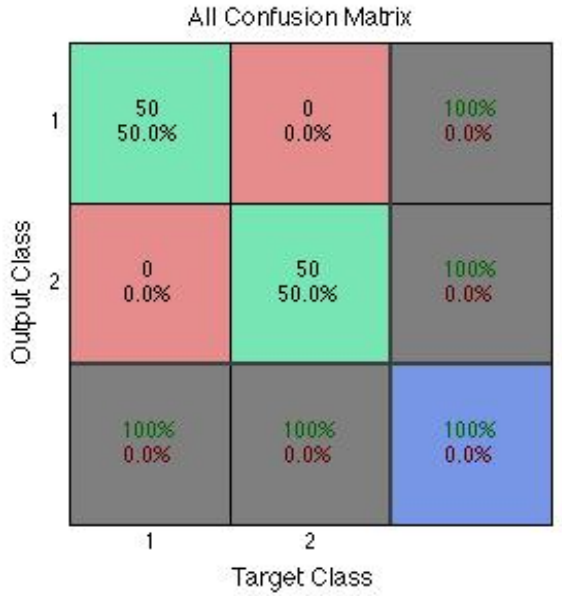

All ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

15 Neurons

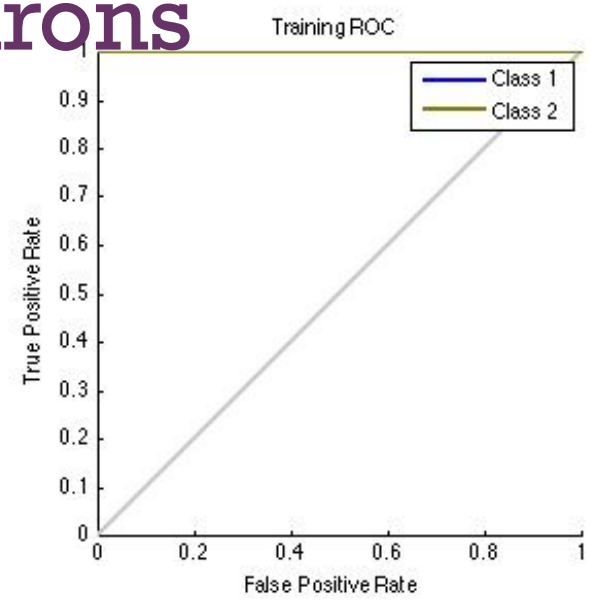

Training Confusion Matrix
32
45.7%
0
0.0%
100%
0.0%
0
0.0%
38
54.3%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Output Class
1
2
Target Class

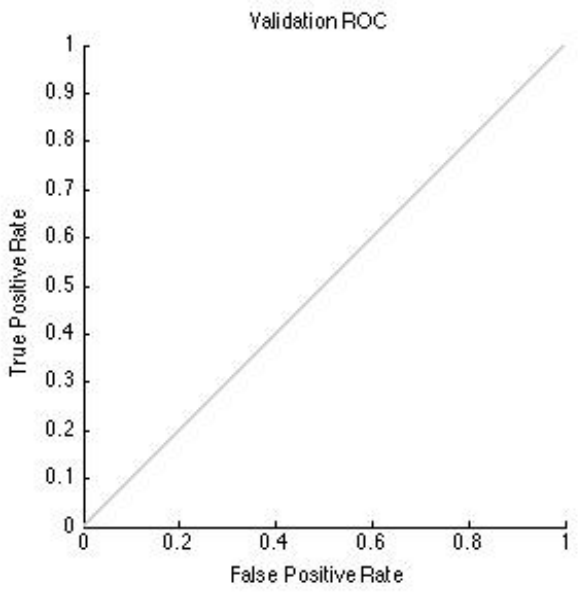

Validation Confusion Matrix
10
66.7%
0
0.0%
100%
0.0%
0
0.0%
5
33.3%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Output Class
1
2
Target Class

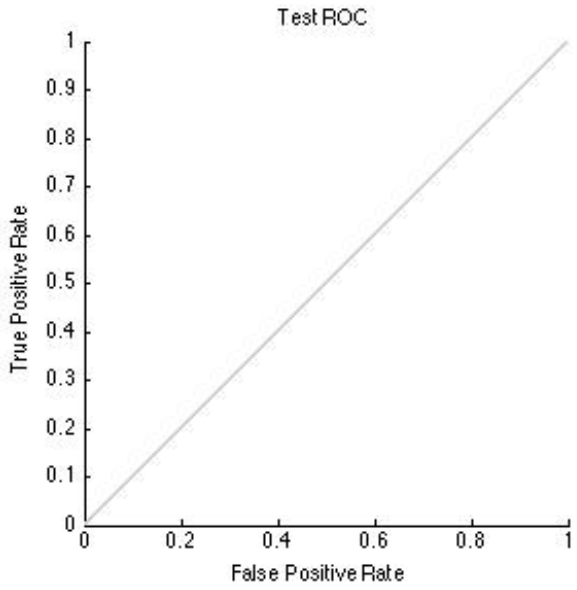

Test Confusion Matrix
8
53.3%
0
0.0%
100%
0.0%
0
0.0%
7
46.7%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Output Class
1
2
Target Class

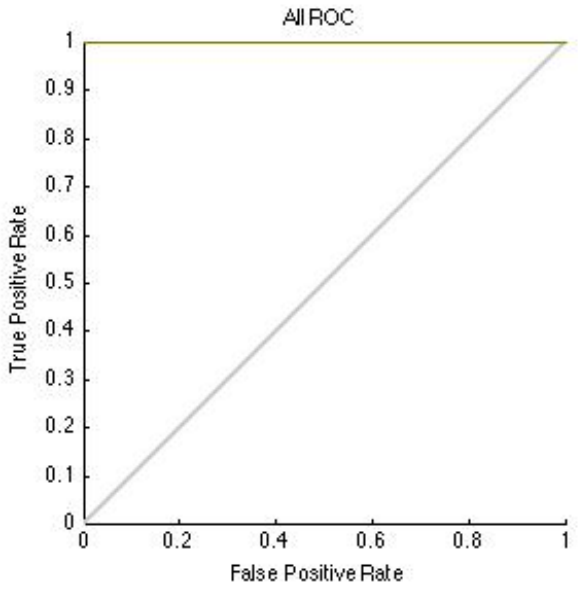

All Confusion Matrix
50
50.0%
0
0.0%
100%
0.0%
0
0.0%
50
50.0%
100%
0.0%
100%
0.0%
100%
0.0%
100%
0.0%
1
2
Output Class
1
2
Target Class

15 Neurons


Training ROC
Class 1
Class 2
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8


Validation ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8


Test ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8


All ROC
True Positive Rate
False Positive Rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8