4.3: MULTILAYER NEURAL NETWORKS

Some Figures in these slides were taken from
Pattern Classification (2nd ed) by R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, 2000
with the permission of the authors

Febrero-Mayo 2006

M. Cabrera, J. Vidal

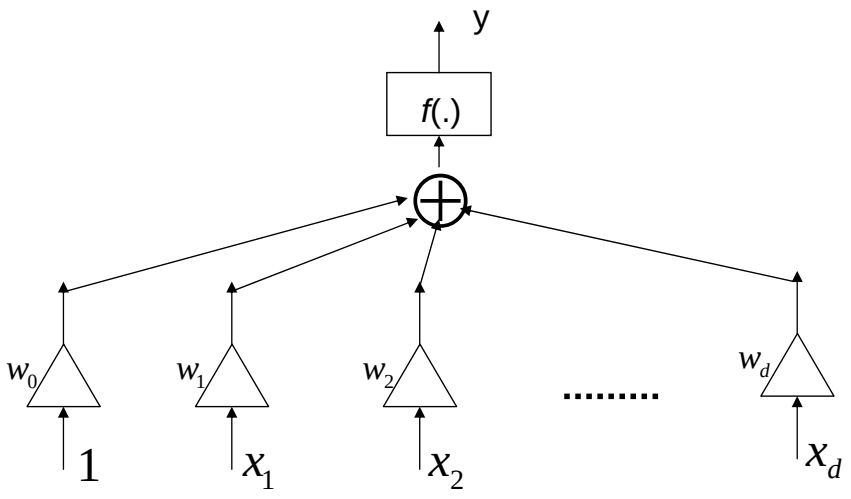
INDEX

1 INTRODUCTION

- Artificial NN originated from the idea to model mathematically human intellectual abilities by biologically plausible engineering designs.
- They are meant to be parallel computational schemes resembling a real brain.
- PROPERTY: Small changes in the training data might lead to a large change in the classifier, both in its structure and parameters.
- Modelling of the human brain, at either morphological or functional level, and trying to understand NN's cognitive capacity are also important research topics.
- A Linear classifier e.g. the perceptron, is a simplified NN (two layers).
- Non linear functions can be used to obtain arbitrary decision regions
- A multilayer neural network or a multilayer perceptron is a non linear function where the parameters are learned from the training database.
- Optimal Network topology depends upon the problem at hand. A complexity adjustment is necessary to decide how many free parameters we need.
- Backpropagation algorithm as a natural extension of the LMS algorithm.

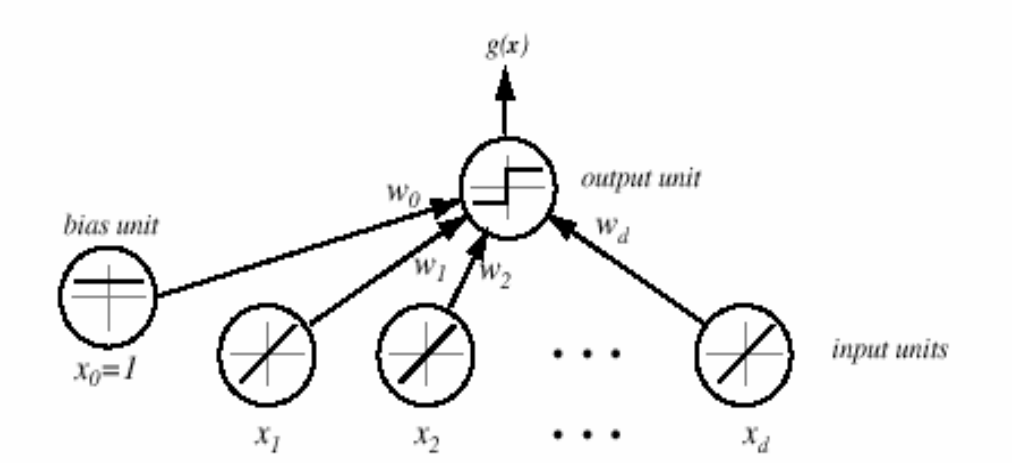
1 INTRODUCTION

- A single Perceptron:Linear decision boundaries and simple logic functions.



1 INTRODUCTION

- Perceptron: Basic block to build a neural net (MLP)



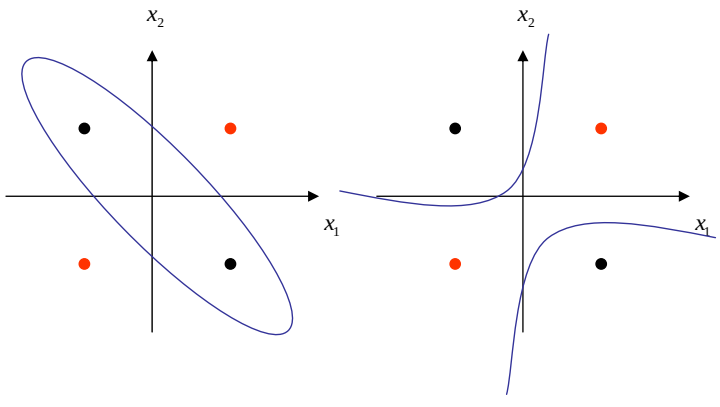
1 THE OR-EXCLUSIVE PROBLEM

AND and OR can be implemented by the perceptron

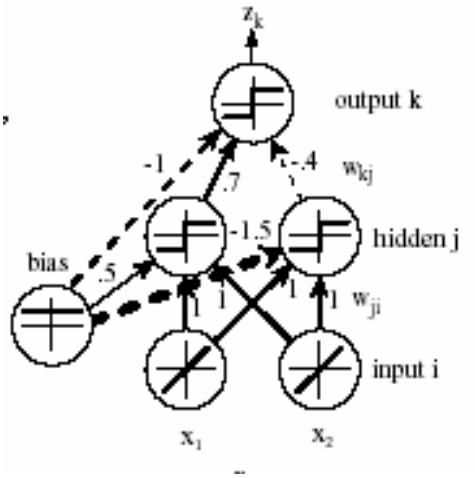
The OR-exclusive: a NON LINEAR problem.

$(x_1 \quad x_2)^T$

Input Vector	Class
(-1,-1)	2
(-1,+1)	1
(+1,-1)	1
(+1,+1)	2



1 THE OR-EXCLUSIVE PROBLEM



$$y_1 = f(Net_1) = sign\left(\begin{pmatrix} +0,5 & +1 & +1 \end{pmatrix} \begin{pmatrix} bias \\ x_1 \\ x_2 \end{pmatrix} \right)$$

$$y_2 = f(Net_2) = sign\left(\begin{pmatrix} -1,5 & +1 & +1 \end{pmatrix} \begin{pmatrix} bias \\ x_1 \\ x_2 \end{pmatrix} \right)$$

$$z = f(Net_{OUTPUT}) = sign\left(\begin{pmatrix} -1 & +0,7 & -0,4 \end{pmatrix} \begin{pmatrix} bias \\ y_1 \\ y_2 \end{pmatrix} \right)$$

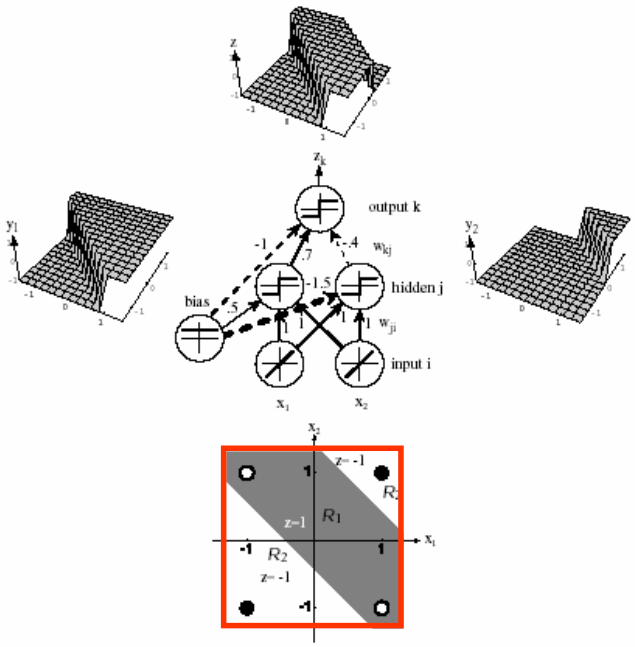
1 THE OR-EXCLUSIVE PROBLEM

The OR-exclusive

As a neural network THREE LAYER:

- An input Layer
- A hidden layer
- An output layer

•2-2-1 Fully Connected Topology



1 INTRODUCTION

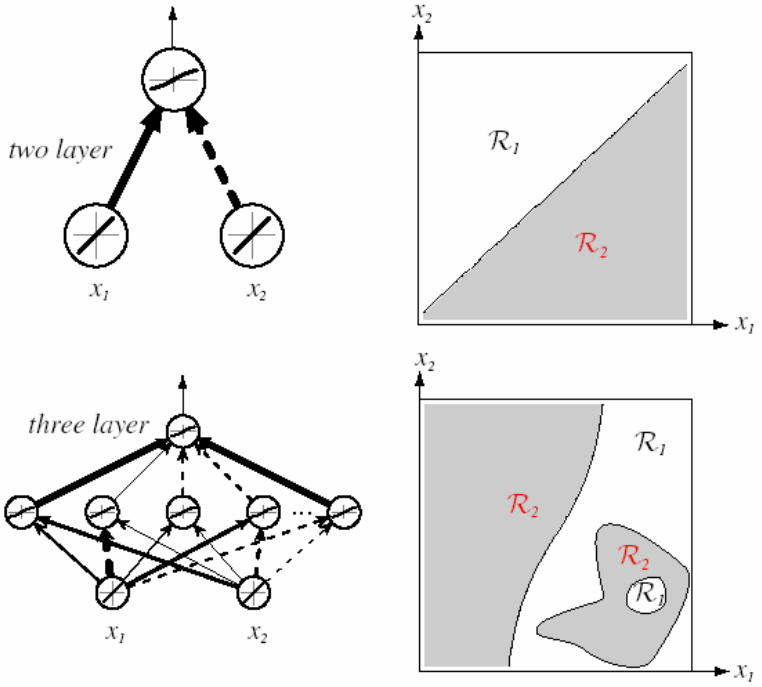
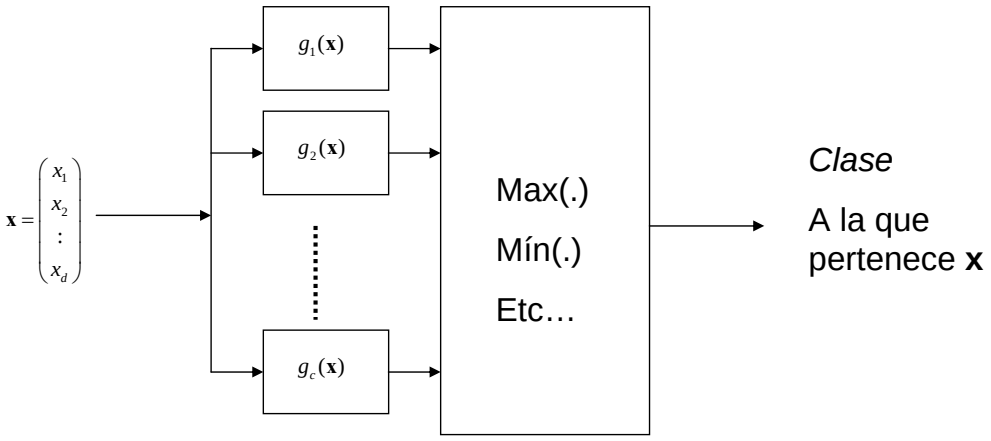


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

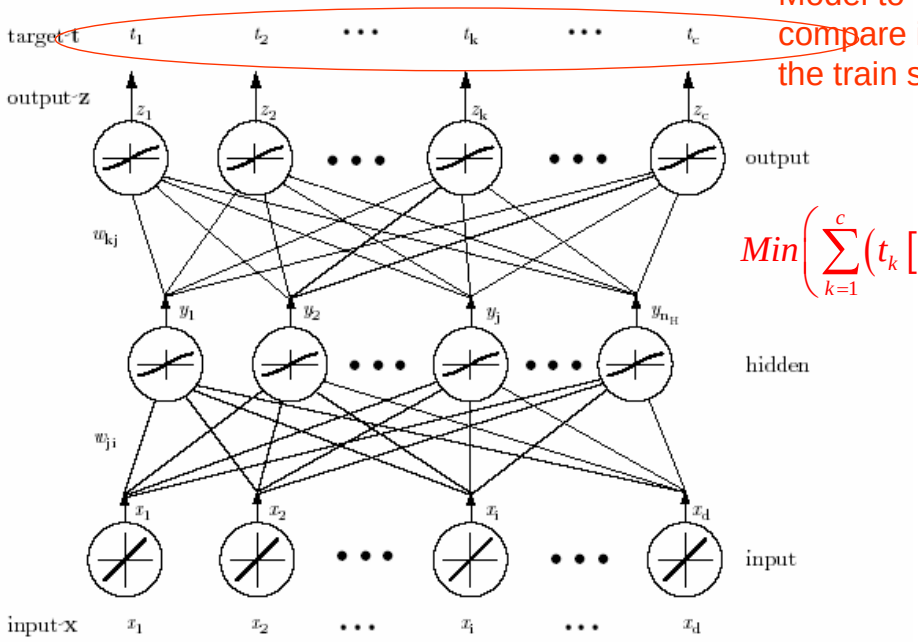
2 MULTILAYER NETWORK

- Caso de múltiples categorías C:



2 MULTILAYER NETWORK

- A three layer: d-h-c



Model to compare in the train step

$$Min\left(\sum_{k=1}^{c}\left(t_k[n]-z_k[n]\right)^2\right)$$

2 MULTILAYER NETWORK

- A three layer network is considered
- Input vector, dimension = *d* $\mathbf{x}=\left(x_1 \quad x_2 \quad .. \quad x_d\right)$
- h hidden nets $\mathbf{y}=\left(f\left(Net_1\right) \quad f\left(Net_2\right) \quad .. \quad f\left(Net_h\right)\right)$

$$y_j = f\left(Net_j\right) = f\left(\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right) = f\left(\sum_{i=0}^{d} w_{ji}x_i\right) = f\left(\mathbf{w}_j{}^T\mathbf{x}\right)$$

- c scalar binary output signals $z_k = f\left(Net_k\right); \quad k=1,2..,c$

$$z_k = f\left(Net_k\right) = f\left(\sum_{j=1}^{h} w_{kj}y_j + w_{k0}\right) = f\left(\sum_{j=0}^{h} w_{kj}y_j\right) = f\left(\mathbf{w}_k{}^T\mathbf{y}\right)$$

- Bias=1: It increases the freedom degree adding a new coordinate to all the weight vectors.

2 MULTILAYER NETWORK

A three layer: d-h-c

C discriminant functions are implemented

OUTPUT

$$z_k = f\left(\sum_{j=1}^{h} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}\right) = f\left(\sum_{j=0}^{h} w_{kj} y_j\right); \qquad k = 1,..,c$$

$$\mathbf{z} = \begin{pmatrix} z_1 \\ : \\ z_c \end{pmatrix} = \begin{pmatrix} g_1(\mathbf{x}) \\ : \\ g_c(\mathbf{x}) \end{pmatrix}$$

2 MULTILAYER NETWORK

Practical Implementation:

- A non linear scalar function *f*(.) enough smooth to learn gradient descent techniques.
- (*d*+1).*h*+(*h*+1).*c* scalar parameters to be designed

Back Propagation Algorithm

- Back Propagation is one of the simplest and most general methods for **supervised training** of MLNN. (Weight Layers computing)
- It is the natural extension of LMS algorithm for linear methods.
- The algorithm is developed in two steps: Feed forward and Learning.

2 MULTILAYER NETWORK

Power of MLNN:

Kolmogorov Theorem: Any continuous function g(x) defined on the unit hypercube $I^d = [0,1]$, $d>=2$ can be represented as:

$$g(x) = \sum_{j=0}^{2d} \Xi \left(\sum_{i=1}^{d} \psi_{ij}(x_i) \right)$$

With properly chosen functions Ξ; $\psi_{ij} = \lambda^i \psi\left(x_i + j\varepsilon\right) + j$

Practical Conclusions:

Choose λ and ψ depends on *d* and are independent of the mapping function

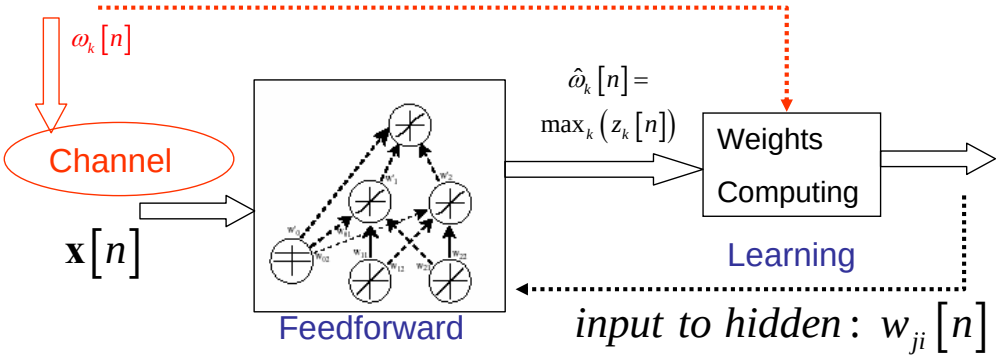
Function Ξ depends on the mapping function to be implemented.

Theoretical interest more than practical

REFERENCE:IEEE-Explore
Neural Networks, 1991. 1991 IEEE International Joint Conference on
Date: 18-21 Nov 1991
Can multilayer mapping networks with finite number of real parameters harness the computational power of Kolmogorov's representation theorem?
Kowalczyk, A. Page(s): 2722-2728 vol.3

3 BACKPROPAGATION ALGORITHM

TRAINING:

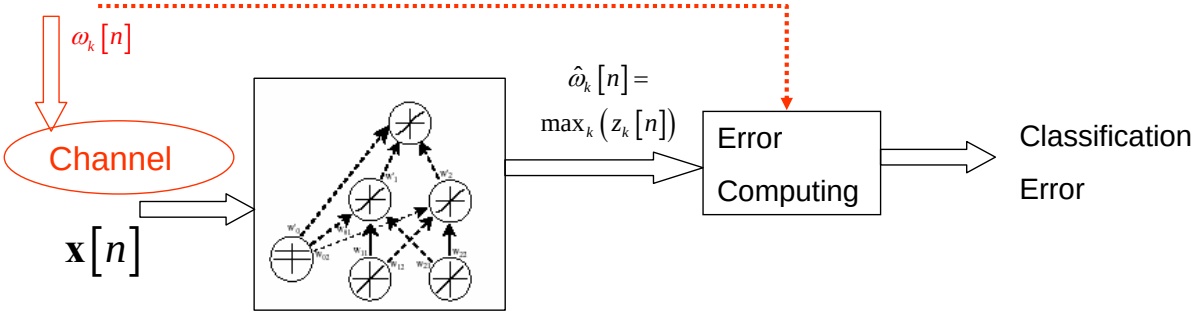


$input\ to\ hidden:\ w_{ji}[n]$

$hidden\ to\ output:\ w_{kj}[n]$

3 BACKPROPAGATION ALGORITHM

TEST:



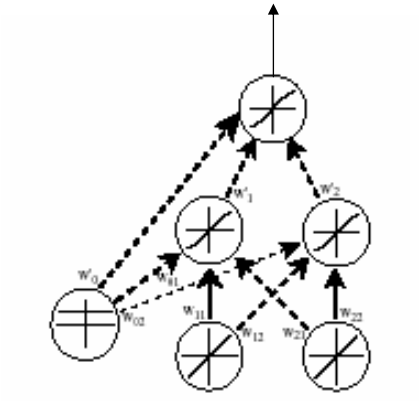
3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING:

- For a supervised database a Training Error (scalar function) is defined.
- The error measures the difference between “the desired output t_k” and “the actual output z_k”.
- The back-propagation learning rule is based on gradient descent (LMS).
- The weights are initialized with random values, and then they are changed in a direction that will reduce the error.
- The learning rate indicates the relative size of the change in weights.
- The exact behavior depends on the starting point.

3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING:



$$\mathbf{w} = \begin{pmatrix} w'_0 \\ w'_1 \\ w'_2 \\ w_{01} \\ w_{11} \\ w_{21} \\ w_{02} \\ w_{12} \\ w_{22} \end{pmatrix}$$

Hidden to output

Input to hidden

3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING:

Scalar error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t_k - z_k\|^2$$

Weight Adaptation Rule:

$$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}} \qquad \Delta w_{pq} = -\eta\frac{\partial J}{\partial w_{pq}}$$

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \Delta\mathbf{w}$$

Output Assignment values:

$$z_k = z_k(x[n]); \Rightarrow t_k = \begin{cases} +1 & x[n] \in \text{Class } k \\ -1 & x[n] \notin \text{Class } k \end{cases}$$

3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING (k output, j hidden node):

- Hidden to Output weight derivative $net_k = \sum_{j=0}^{n_H} w_{kj} y_j$

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -(t_k - z_k) f'(net_k)\frac{\partial net_k}{\partial w_{kj}} = -(t_k - z_k) f'(net_k) y_j = -\delta_k y_j$$

Sensitivity of unit k

- Input to Hidden weight derivative $y_j = f(net_j); net_j = \sum_{i=0}^{d} w_{ji} x_i$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = -\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{kj} f'(net_j) x_i = \sum_{k=1}^{c} \delta_k w_{kj} f'(net_j) x_i = -\delta_j x_i$$

-Sensitivity of unit j

3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING: The sensitiviy of a node describes how the overall error changes with the unit's net activation and it is actually computed from the output layer bacwards. "Backpropagated error"

- Hidden to Output Sensitivity

$$\delta_k = +(t_k - z_k) f'(net_k)$$

- Input to Hidden Sensitivity

$$\delta_j = -\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{kj} f'(net_j) = f'(net_j)\sum_{k=1}^{c}\delta_k w_{kj}$$

3 BACKPROPAGATION ALGORITHM

SENSITIVITY AT A HIDDEN UNIT:

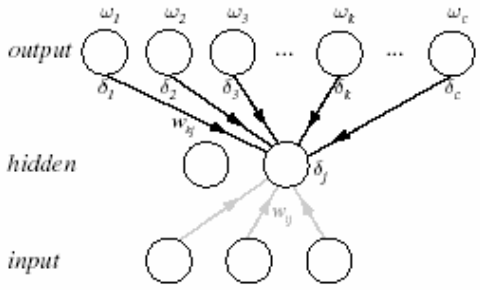


FIGURE 6.5. The sensitivity at a hidden unit is proportional to the weighted sum of the sensitivities at the output units: $\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k$. The output unit sensitivities are thus propagated "back" to the hidden units. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

3 BACKPROPAGATION ALGORITHM

LEARNING STRATEGIES:

- Algorithm 1: Stochastic BackPropagation

Algorithm 1 (Stochastic backpropagation)

```
1 begin initialize network topology (# hidden units), w, criterion θ, η, m ← 0
2   do m ← m + 1
3       x^m ← randomly chosen pattern
4       w_ij ← w_ij + ηδ_j x_i;   w_jk ← w_jk + ηδ_k y_j
5   until ∇J(w) < θ
6 return w
7 end
```

Sens itivity

Sensi tivity

3 BACKPROPAGATION ALGORITHM

LEARNING STRATEGIES:

- Algorithm 2: Batch Propagation

Algorithm 2 (Batch backpropagation)

```
1  begin initialize  network topology (# hidden units), w, criterion θ, η, r ← 0
2    do r ← r + 1 (increment epoch)
3       m ← 0; Δw_ij ← 0; Δw_jk ← 0
4       do m ← m + 1
5          x^m ← select pattern
6          Δw_ij ← Δw_ij + ηδ_j x_i;   Δw_jk ← Δw_jk + ηδ_k y_j
7       until m = n
8       w_ij ← w_ij + Δw_ij;   w_jk ← w_jk + Δw_jk
9    until ∇J(w) < θ
10 return w
11 end
```

It works by epochs

Sensitivity

Sensitivity

3 BACKPROPAGATION ALGORITHM

BackPropagation Algorithm and Jacobian Matrix

$$J(\mathbf{w}) = \tfrac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \tfrac{1}{2}\|t_k - z_k\|^2 = \tfrac{1}{2}\sum_{k=1}^{c}(e_k)^2 = \tfrac{1}{2}\|\mathbf{e}\|^2 \Rightarrow$$

$$\nabla J_{\mathbf{w}} = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_L} \end{pmatrix}; \qquad \frac{\partial J}{\partial w_i} = \sum_{k=1}^{c} e_k \frac{\partial e_k}{\partial w_i}; \qquad \nabla J_{\mathbf{w}} = \begin{pmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_2}{\partial w_1} & \vdots & \frac{\partial e_c}{\partial w_1} \\ \frac{\partial e_1}{\partial w_2} & \frac{\partial e_2}{\partial w_2} & \vdots & \frac{\partial e_c}{\partial w_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e_1}{\partial w_L} & \frac{\partial e_2}{\partial w_L} & \vdots & \frac{\partial e_c}{\partial w_L} \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_c \end{pmatrix} = \mathbf{J}^T\mathbf{e};$$

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\nabla J_{\mathbf{w}}[n] = \mathbf{w}[n] - \eta\mathbf{J}^T[n]\mathbf{e}[n]$$

3 BACKPROPAGATION ALGORITHM

NETWORK LEARNING:

- Error evolution with the number of epochs. (Decreases for the training set and has a minimum with the test set)
- The error function can have some local minimum. How to find the global one?
- Overfitting against danger to fall in a local minima.
- Local minima: Difficult analytic solution. Descent gradient techniques with multiple initialization points are preferred.

4 Activation function *f*(.)

- Hard limiting (*It's optimal for a high number of patterns in the data base*)
  - With this function the boundaries are combination of linear boundaries: Hyperplanes

$$f(x) = sign(x)$$

- Sigmoid (*Continuous Differentiability)*
  - The linear boundaries are connected in a smooth fashion because of the sigmoid non linearity.

$$f(x) = a\tanh(bx) = a\left(\frac{1-e^{-bx}}{1+e^{-bx}}\right) = \frac{2a}{1+e^{-bx}} - a$$

4 Activation function *f*(.)

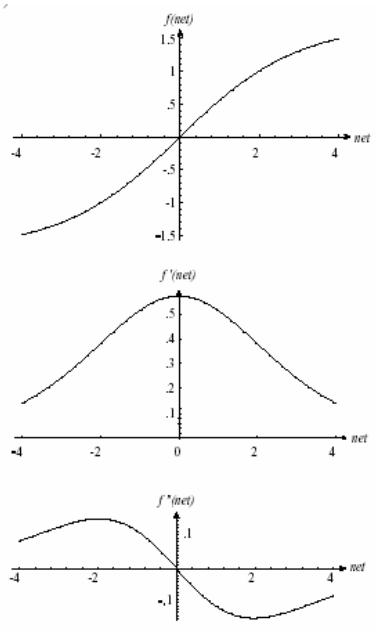
- Activation function *f*(.) must be
  - Continuous and derivable
  - Non linear. Otherwise three layer network results two layer network
  - Linearity for small values
  - Saturation (have some minimum and maximum output values)
  - Monotonicity helps to avoid undesirable local extrema

- Sigmoid (a=1.716, b=2/3)

$$f(x) = a\tanh(bx) =$$

$$a\left(\frac{1-e^{-bx}}{1+e^{-bx}}\right) =$$

$$\frac{2a}{1+e^{-bx}} - a$$



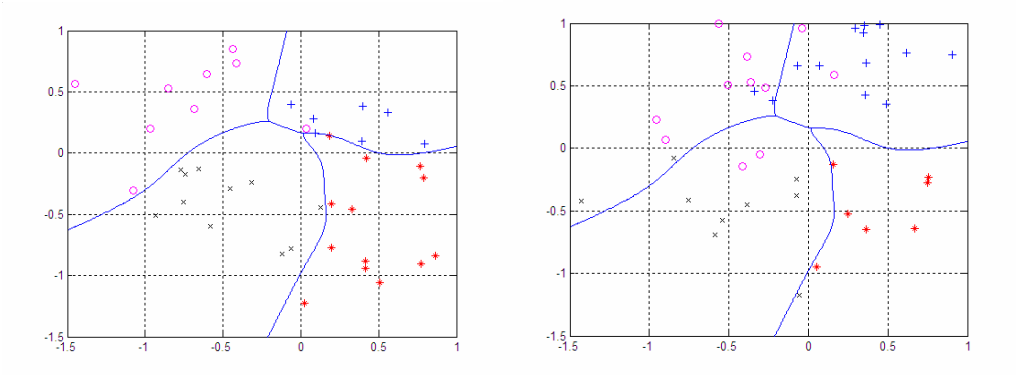
5 Simulation Results

C=4 classes (QPSK, SNR=3 dB)

Two 5 nodes hidden layers: $f(x) = \text{sigmoide}(\mathbf{w}^T\mathbf{x}) = \frac{2}{1+\exp(-\mathbf{w}^T\mathbf{x})} - 1$

2-5-5-4 NN $a = 1; b = 1$

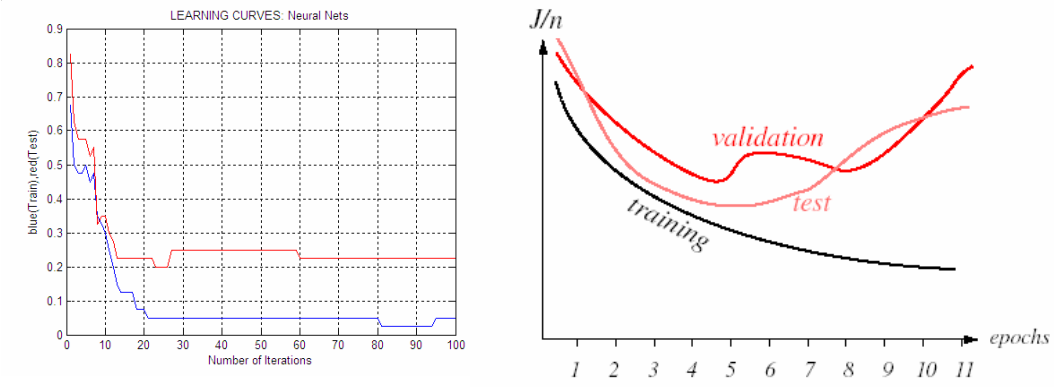
Back Propagation algorithm



5 Simulation Results

C=4 classes (QPSK, SNR=3 dB)

Two 5 nodes hidden layers: 2-5-5-4; Back Propagation algorithm



6 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

- Scaling Input:
  - Average over the training set of each feature is zero
  - The full data should be scaled to have the same variance in each feature component. (1 with the sigmoid function).
  - Any test set must be subjected to the same transformation before its classification.
- Output Values for classification :+1,-1 for normalization proposes
- Gaussian d-multidimensonal noise can be added to increase then train set size.
- Number of hidden units h:
  - It represents the number of degree of freedom
  - Governs the expressive power of the net
  - Few hidden units are needed if the patterns are well separated.
  - Pruning, h=N/10 with N the size of the training test, etc…

6 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

- Initializing Weights:
  - Different to zero
  - Uniform learning (all weights reach final equilibrium simultaneously) and all category are learned at the same time.
  - Data standardization also helps uniform learning
  - Choose them randomly from a uniform distribution (-W,+W) helps uniform learning. With a Sigmoid
  - Input to hidden choose W=1/sqrt(d)
  - Hidden to output choose W=1/sqrt(h)
- Learning Rate
  - It determines the speed at which the network attains the minimum
  - The optimal learning rate leads to the local error minimum in one learning step
  - For a quadratic criterion function:

$$\frac{\partial^2 J}{\partial w^2}\Delta w = \frac{\partial J}{\partial w} \qquad \eta_{opt} = \left(\frac{\partial^2 J}{\partial w^2}\right)^{-1}$$

6 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

- Stopped Training:
  - Excessive Training can lead to poor generalization (Overtraining)
- Other Criterion Function:
  - Minkowski error
  - Kullback Leibler Criteria
- Second order method to compute the gradient.
  - Hessian Matrix
  - Newton's method
- Conjugate gradient method

6 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

Pruning: To eliminate weights least needed.

- Sometimes weights with small magnitudes are important for learning.
- Algorithm OPTIMAL BRAIN estimates the importance of a weight in the final solution based on second derivative
- The J gradient is expanded to a Taylor series. The first order term drop out automatically because the network has converged to a local minimum.

6 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

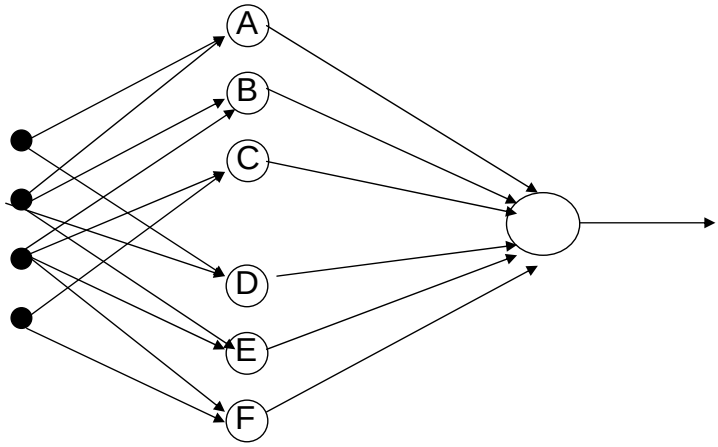
GENERALIZATION ALTERNATIVES:

- Input Units include a Bias Unit
- Input Units are connected to hidden and output units
- Number of hidden layers
- Different nonlinearities for different layers and/or units.
- Different Learning rates for different units.

4 PRACTICAL TECHNIQUES FOR IMPROVING BACK PROPAGATION

Weight Reduction and Sharing.

- Application: Hand written OCR



7 Second Gradient Order Methods

Gauss-Newton Method:

- Second Order Gradient Descent Algorithm
  - H is the Hessian Matrix and J is the Jacobian Matrix

$$J[n+1] \simeq J[n] + \nabla J_{\mathbf{w}}{}^{\mathrm{T}} \Delta\mathbf{w}[n] + \tfrac{1}{2}\Delta\mathbf{w}[n]^{\mathrm{T}}\,\mathbf{H}\Delta\mathbf{w}[n]$$

$$\mathbf{H} \simeq \mathbf{J}^{\mathrm{T}}\mathbf{J}$$

$$0 = \nabla J_{\mathbf{w}} + \mathbf{H}\Delta\mathbf{w}[n] \Rightarrow \Delta\mathbf{w}[n] = -\mathbf{H}^{-1}\nabla J_{\mathbf{w}} \simeq -\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{T}\mathbf{e}$$

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\nabla J_{\mathbf{w}}[n] = \mathbf{w}[n] - \eta\left(\mathbf{J}[n]^{\mathrm{T}}\,\mathbf{J}[n]\right)^{-1}\mathbf{J}^{T}[n]\mathbf{e}[n]$$

7 Second Gradient Order Methods

LEVENBERG-MARQUADT ALGORITHM

(PrTools) : *lmnc* as a blend of "BP-Gradient Descent" and "Gauss-Newton Methods"):

BP (Safe): $\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\mathbf{J}[n]^{T}\,\mathbf{e}[n]$

GN (Quick) $\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\left(\mathbf{J}[n]^{T}\,\mathbf{J}[n]\right)^{-1}\mathbf{J}[n]^{T}\,\mathbf{e}[n]$

LM (Augmented λ):

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\left(\mathbf{J}[n]^{T}\,\mathbf{J}[n] + \lambda\mathbf{I}\right)^{-1}\mathbf{J}^{T}[n]\mathbf{e}[n]$$

- The LMS algorithm computes the approximation to the Bayes discriminant function for two layer nets.

$$\Pr(\omega_k|\mathbf{x}) = \frac{\Pr(\mathbf{x}|\omega_k)\Pr(\omega_k)}{\sum_{i=1}^{c}\Pr(\mathbf{x}|\omega_i)\Pr(\omega_i)}$$

$$z_k(\mathbf{x}) = \Pr(\omega_k|\mathbf{x})$$

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2$$

$$t_k(\mathbf{x}) = \begin{cases} 1; & \mathbf{x} \in \omega_k \\ 0; & \mathbf{x} \notin \omega_k \end{cases}$$

- Contribution to the criterion function based on a single output unit k:

$$J_k(\mathbf{w}) \cong \sum_{\mathbf{x}}(g_k(\mathbf{x},\mathbf{w}) - t_k(\mathbf{x}))^2 = \sum_{\mathbf{x}\in\omega_k}(g_k(\mathbf{x},\mathbf{w})-1)^2 + \sum_{\mathbf{x}\notin\omega_k} g_k(\mathbf{x},\mathbf{w})^2 =$$

$$n\left\{\frac{n_k}{n}\frac{1}{n_k}\sum_{\mathbf{x}\in\omega_k}(g_k(\mathbf{x},\mathbf{w})-1)^2 + \frac{n-n_k}{n}\frac{1}{n-n_k}\sum_{\mathbf{x}\notin\omega_k} g_k(\mathbf{x},\mathbf{w})^2\right\}$$

- n goes to infinite

$$\lim_{n\to\infty}\left(\frac{J_k(\mathbf{w})}{n}\right) \cong$$

$$\Pr(\omega_k)\int(g_k(\mathbf{x},\mathbf{w})-1)^2\Pr(\mathbf{x}|\omega_k)d\mathbf{x} + (1-\Pr(\omega_k))\int g_k(\mathbf{x},\mathbf{w})^2\Pr(\mathbf{x}|\omega_i \neq \omega_k)d\mathbf{x}$$

8 MLP and BAYES THEORY

$$\lim_{n\to\infty}\left(\frac{J_k(\mathbf{w})}{n}\right)\cong$$

$$\Pr(\omega_k)\int\left(g_k(\mathbf{x},\mathbf{w})-1\right)^2\Pr(\mathbf{x}|\omega_k)\,d\mathbf{x}+\left(1-\Pr(\omega_k)\right)\int g_k(\mathbf{x},\mathbf{w})^2\Pr(\mathbf{x}|\omega_i\neq\omega_k)\,d\mathbf{x}=$$

$$\int g_k(\mathbf{x},\mathbf{w})^2\left(\Pr(\omega_k)\Pr(\mathbf{x}|\omega_k)+\left(1-\Pr(\omega_k)\right)\Pr(\mathbf{x}|\omega_i\neq\omega_k)\right)d\mathbf{x}+$$

$$-2\int g_k(\mathbf{x},\mathbf{w})\Pr(\omega_k)\Pr(\mathbf{x}|\omega_k)\,d\mathbf{x}+\int\Pr(\omega_k)\Pr(\mathbf{x}|\omega_k)\,d\mathbf{x}=$$

$$\int g_k(\mathbf{x},\mathbf{w})^2\Pr(\mathbf{x})\,d\mathbf{x}-2\int g_k(\mathbf{x},\mathbf{w})\Pr(\mathbf{x},\omega_k)\,d\mathbf{x}+\int\Pr(\mathbf{x},\omega_k)\,d\mathbf{x}=$$

$$\int\left(g_k(\mathbf{x},\mathbf{w})-\Pr(\omega_k|\mathbf{x})\right)^2\Pr(\mathbf{x})\,d\mathbf{x}-\int\Pr(\omega_k|\mathbf{x})^2\Pr(\mathbf{x})\,d\mathbf{x}+\int\Pr(\mathbf{x},\omega_k)\,d\mathbf{x}=$$

$$\int\left(g_k(\mathbf{x},\mathbf{w})-\Pr(\omega_k|\mathbf{x})\right)^2\Pr(\mathbf{x})\,d\mathbf{x}+\int\Pr(\omega_k|\mathbf{x})\left(1-\Pr(\omega_k|\mathbf{x})\right)\Pr(\mathbf{x})\,d\mathbf{x}=$$

$$\int\left(g_k(\mathbf{x},\mathbf{w})-\Pr(\omega_k|\mathbf{x})\right)^2\Pr(\mathbf{x})\,d\mathbf{x}+\int\Pr(\omega_k|\mathbf{x})\Pr(\omega_i\neq\omega_k|\mathbf{x})\Pr(\mathbf{x})\,d\mathbf{x}$$

8 MLP and BAYES THEORY

$$J(\mathbf{w})=\tfrac{1}{2}\sum_{k=1}^{c}\left(t_k-z_k\right)^2=\tfrac{1}{2}\sum_{k=1}^{c}J_k(\mathbf{w})\Rightarrow\lim_{n\to\infty}\left(\frac{J(\mathbf{w})}{n}\right)\cong$$

$$\sum_{k=1}^{c}\int\left(g_k(\mathbf{x},\mathbf{w})-\Pr(\omega_k|\mathbf{x})\right)^2\Pr(\mathbf{x})\,d\mathbf{x}+\textit{term independent of }\mathbf{w}$$

In the limit of infinite data the outputs of the trained network will give:

$$z_k=g_k(\mathbf{x},\mathbf{w})\simeq\Pr(\omega_k|\mathbf{x})$$

(In pr tools it is used to compute the ROC)

With n finite number, some approaches toward approximate probabilities can be chosen:

- To choose the output Non-linearity to be exponential rather than sigmoidal
- To normalize the outputs to sum to 1.0

Kullback-Leibler directed divergence Criterion

- Neural Network dedicated to estimate the probability Pr{ω_i/**x**}
- It is always Non Negative

$$J_{KL} = \sum_{i=1}^{c} \int \Pr\{\omega_i | \mathbf{x}\} \log\left(\frac{\Pr\{\omega_i | \mathbf{x}\}}{y_i(\mathbf{x}, \mathbf{w})} \right) \Pr\{\mathbf{x}\}\, d\mathbf{x}$$

- It is equivalent to minimize:

$$J_{KL} = -\sum_{i=1}^{c} \int \Pr\{\omega_i, \mathbf{x}\} \log\left(y_i(\mathbf{x}, \mathbf{w})\right) d\mathbf{x} \cong -\tfrac{1}{N} \sum_{i=1}^{c} \sum_{\mathbf{x}[n] \in c_i} \log\left(y_i[n]\right)$$

9 RADIAL BASIS FUNCTION NETWORKS

- A RBF Network is a two-layer network whose output nodes form a linear combination of the basis (or kernel) functions computed by the hidden layer nodes.
- It produces a localized response to a input stimulus.
- The most common basis is a Gaussian kernel function.
- Solution requires standard linear techniques
- The method is generally confined to problems of moderate size (Matrix Inversion)

9 RADIAL BASIS FUNCTION NETWORKS

A RBF Network with linear output units implements:

$$z_k(\mathbf{x}) = \sum_{j=0}^{h} w_{kj}\phi_j(\mathbf{x})$$

$$\mathbf{z}(\mathbf{x}) = \begin{pmatrix} z_1(\mathbf{x}) \\ z_2(\mathbf{x}) \\ : \\ z_C(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & : & w_{1h} \\ w_{21} & w_{22} & : & w_{2h} \\ : & : & : & : \\ w_{C1} & w_{C1} & : & w_{Ch} \end{pmatrix} \begin{pmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ : \\ \phi_h(\mathbf{x}) \end{pmatrix} = \mathbf{W}\boldsymbol{\varphi}(\mathbf{x})$$

Criterion Function (with N: number of patterns or samples)

$$J(\mathbf{W}) = \tfrac{1}{2}\sum_{n=1}^{N} J_n(\mathbf{W}) = \tfrac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{z}(\mathbf{x}[n],\mathbf{W}) - \mathbf{t}[n]\right\|^2 =$$

$$\tfrac{1}{2}\sum_{n=1}^{N}\left\|\begin{pmatrix}\sum_{j=0}^{h} w_{1j}\phi_j(\mathbf{x}[n]) \\ \vdots \\ \sum_{j=0}^{h} w_{Cj}\phi_j(\mathbf{x}[n])\end{pmatrix} - \begin{pmatrix} t_1[n] \\ \vdots \\ t_C[n]\end{pmatrix}\right\|^2 = \tfrac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{W}\boldsymbol{\varphi}(\mathbf{x}[n]) - \mathbf{t}[n]\right\|^2 =$$

$$\tfrac{1}{2}Trace\left(\left(\Phi^{\mathbf{T}}(\mathbf{x})\mathbf{W}^{\mathbf{T}} - \mathbf{T}^{\mathbf{T}}\right)\left(\mathbf{W}\Phi(\mathbf{x}) - \mathbf{T}\right)\right)$$

With:

$$\Phi(\mathbf{x}) = \left(\boldsymbol{\varphi}(\mathbf{x}[1]) \quad \boldsymbol{\varphi}(\mathbf{x}[2]) \quad : \quad \boldsymbol{\varphi}(\mathbf{x}[N])\right)$$

$$\mathbf{T} = \left(\mathbf{t}[1] \quad \mathbf{t}[2] \quad : \quad \mathbf{t}[N]\right)$$

Solution Weights:

$$\Phi(\mathbf{x})\Phi^{\mathbf{T}}(\mathbf{x})\mathbf{W}^{\mathbf{T}} = \Phi(\mathbf{x})\mathbf{T}^{\mathbf{T}} \Rightarrow$$

$$\mathbf{W}^{T} = \left(\Phi(\mathbf{x})\Phi^{\mathbf{T}}(\mathbf{x})\right)^{-1}\Phi(\mathbf{x})\mathbf{T}^{\mathbf{T}}$$

With:

$$\Phi(\mathbf{x}) \in \mathbb{R}^{hxN}$$

$$\mathbf{T} \in \mathbb{R}^{CxN}$$

$$\mathbf{W} \in \mathbb{R}^{Cxh}$$

10 CONCLUSIONS

Useful for non-linear classification problems

- Caution with the topology ¡¡¡¡
- Number of hidden nodes
- When to stop training??