COMPUTER VISION

Machine Learning & Neural Networks

6.- Supervised Neural Networks: Multilayer Perceptron

by
Pascual Campoy
Grupo de Visión por Computador
U.P.M. - DISAM

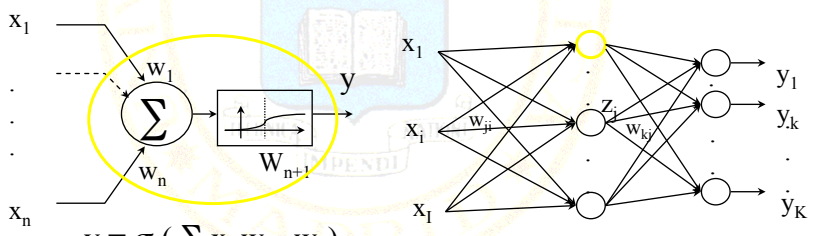
COMPUTER VISION

topics

- *Artificial Neural Networks*
- *Perceptron and the MLP structure*
- *The Back-Propagation learning algorithm*
- *MLP features and drawbacks*
- *The auto-encoder*

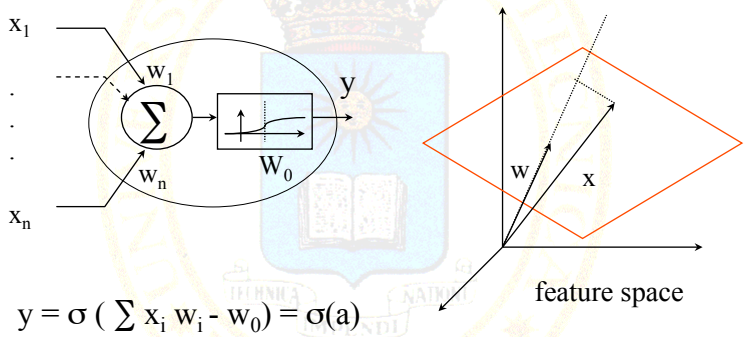
Artificial Neural Networks

- *"A net of simple, adaptable & interconnected units, having parallel processing capability, whose objective is to interact with the environment in a similar way as the natura neural network do"*



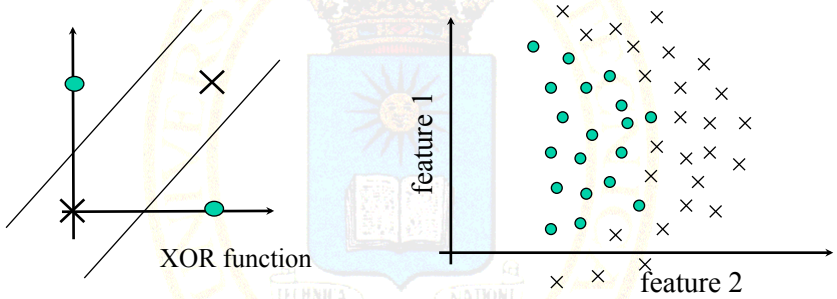
$$y = \sigma (\sum x_i w_i - w_0)$$

The perceptron: working principle

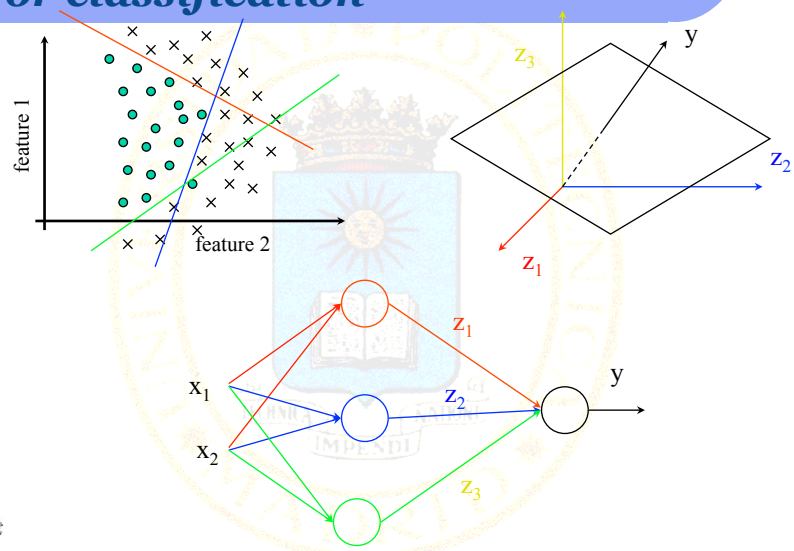


$$y = \sigma (\sum x_i w_i - w_0) = \sigma(a)$$

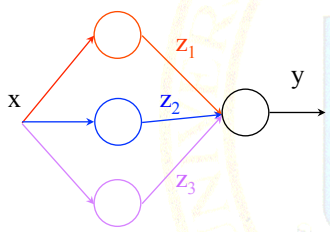
The perceptron for classification

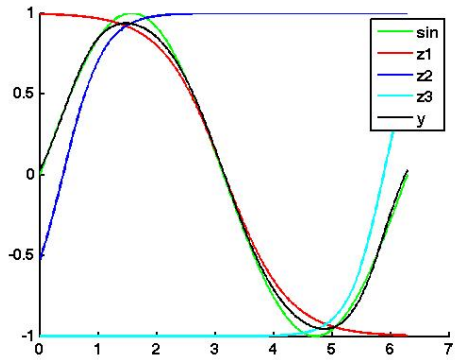


Multilayer Perceptron (MLP): for classification



Multilayer Perceptron (MLP): for function generalization

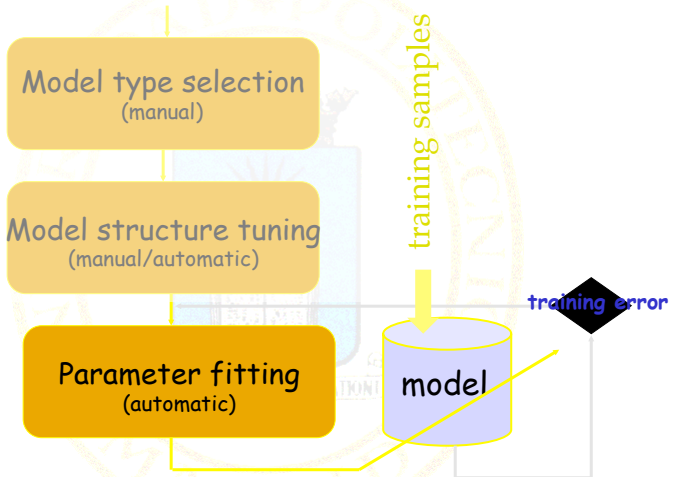




topics

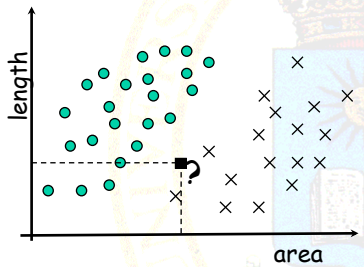
- *Artificial Neural Networks*
- *Perceptron and the MLP structure*
- *The back-propagation learning algorithm*
- *MLP features and drawbacks*
- *The auto-encoder*

Building machine learning models: levels



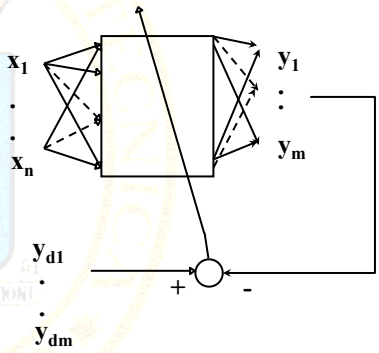
Supervised learning

Supervised learining concept



Feature space

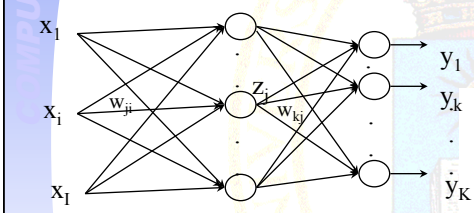
Working structure



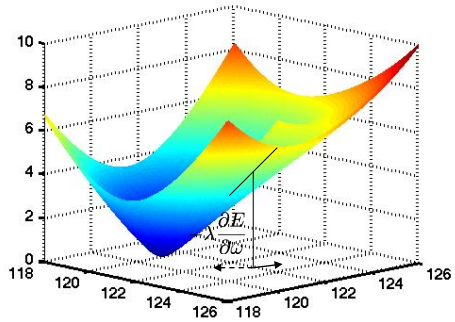
$R^n \Rightarrow R^m$ function generalitation

The back-propagation learning algorithm: working principle

$$E = \frac{1}{2}\sum_k (y_k^n - y_{dk}^n)^2 = \frac{1}{2}\sum_k (y_k^n(\omega_{kj}, \omega_{ji}, x_i) - y_{dk}^n)^2$$



$$\Delta\omega = -\lambda \frac{\partial E}{\partial \omega}$$



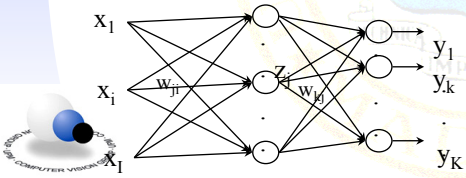
The back-propagation learning algorithm: equations

$$E = \frac{1}{2}\sum_k (y_k^n - y_{dk}^n)^2 = \frac{1}{2}\sum_k (y_k^n(\omega_{kj}, \omega_{ji}, x_i) - y_{dk}^n)^2$$

$$\boxed{\frac{\partial E}{\partial w_{kj}} = (y_k - y_{dk}) z_j}$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_k (y_k - y_{dk}) \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}} = \sum_k (y_k - y_{dk}) w_{kj} \frac{\partial z_j}{\partial a_j} x_i$$

$$\boxed{\frac{\partial E}{\partial w_{ji}} = \sum_k (y_k - y_{dk}) w_{kj} z_j (1 - z_j) x_i}$$



$$y_j = \frac{1}{1+e^{-a_j}} \quad \Rightarrow \quad \frac{\partial y_j}{\partial a_j} = \frac{e^{-a_j}}{(1+e^{-a_j})^2} = (1-y_j)y_j$$

$$y_j = tanh(a_j) \quad \Rightarrow \quad \frac{\partial y_j}{\partial a_j} = 1 - y_j^2$$

Matlab commands:

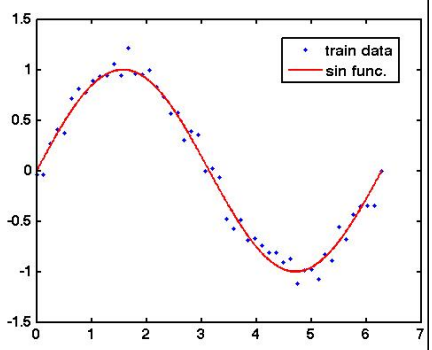
```
% MLP building
>> net = newff(minmax(p.valor),[nL1 nL2],{'tansig' 'purelin'},'trainlm');

% MLP training
>> [net,tr]=train(net,p.valor,p.salida);

% answer
>> anst=sim(net,t.valor);
>> errortest=mse(t.salida-anst);
```


Exercise 6.1: MLP for function generalization

```
% training data
  Ntra=50; xi=linspace(0,2*pi,Ntra); %xe= 2*pi*rand(1,Ntra);
  for i=1:Ntra
  yd(i)=sin(xi(i))+normrnd(0,0.1);
  end
% ground truth
  yt_gt=sin(xt);
% test data
  Ntest=100; xt=linspace(0,2*pi,Ntest);
  for i=1:Ntest
  yt(i)=yt_gt(i)+normrnd(0,0.1);;
  end
% plot
   plot(xi,yd,'b.'); hold on;
   plot(xt,yt_gt,'r-');
```



Exercise 6.1: MLP for function generalization

Using above mentioned data generation procedure:

a) Choose an **adequate MLP** structure and training set. Plot in the same figure the training set, the output of the MLP for the test set, and the ground truth *sin* function.

Evaluate the evolution of the train error, the test error and the ground truth error in the following cases:

b) Changing the **training parameters**:
   initial values, (# of epochs, optimization algorithm)

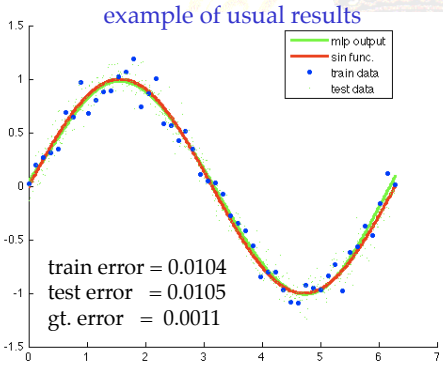
c) Changing the **training data**:
   # of samples (order of samples)

d) Changing the **net structure**:
   # of neurons

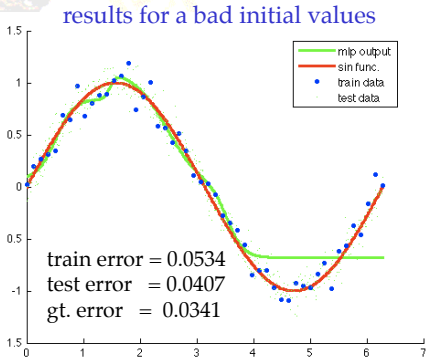
Results for exercise 6.1: a) adequate MLP structure and training set

- 50 training samples
- 4 neurons in hidden layer

example of usual results

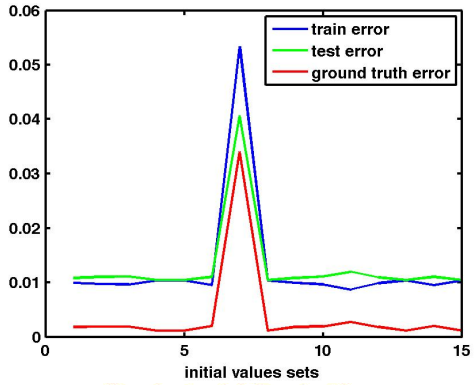


results for a bad initial values



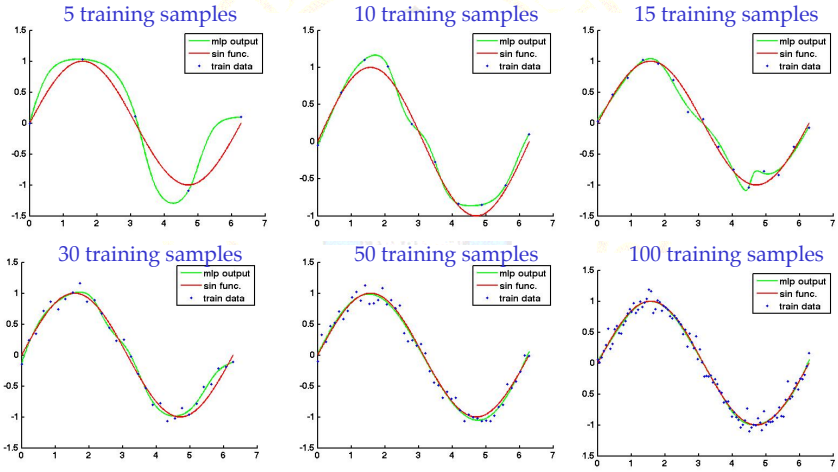
Results for exercise 6.1: b) changing initial values

- 50 training samples
- 4 neurons in hidden layer



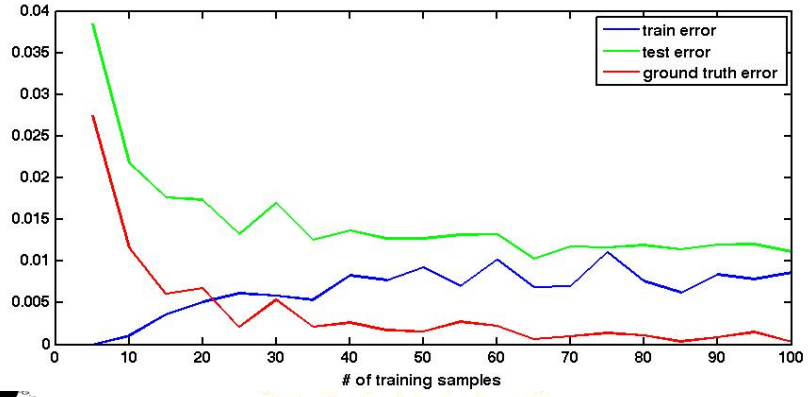
Results for exercise 6.1: c) changing # of training samples ...

- 4 neurons in hidden layer



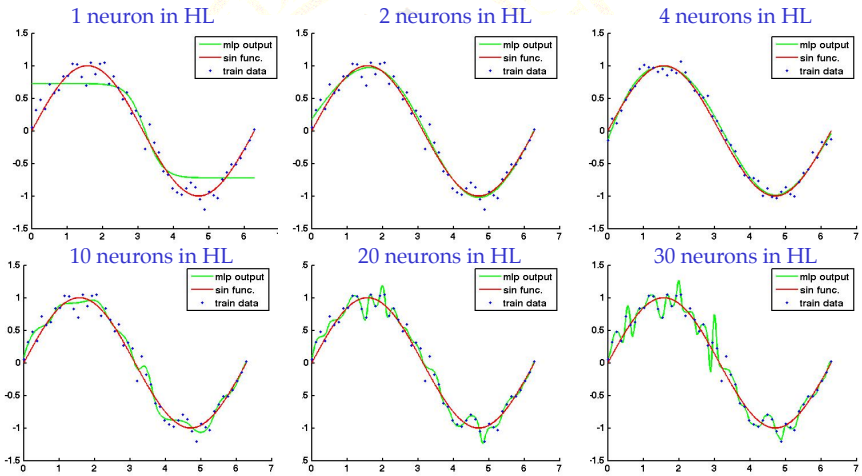
Results for exercise 6.1: c) ... changeing # of training samples

- 4 neurons in hidden layer (mean error over 4 tries)



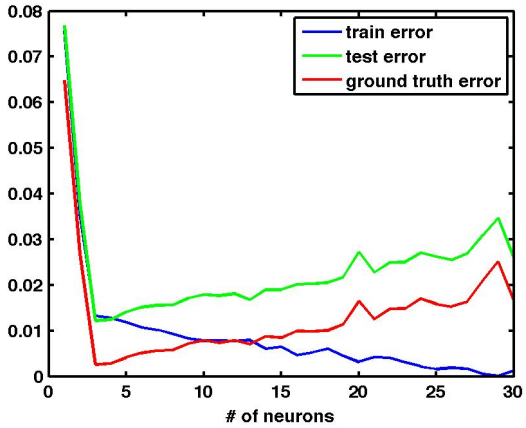
Results for exercise 6.1: d) changing # neurons ...

- 50 training samples



Results for exercise 6.1: d) ... changing # neurons

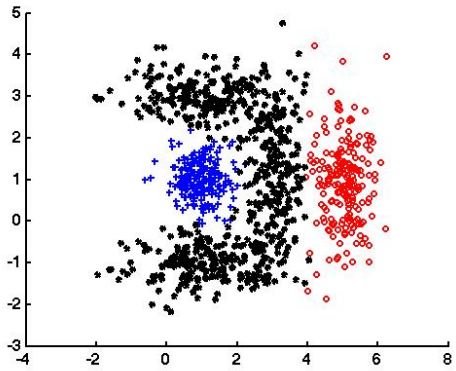
- 50 training samples (mean error over 4 tries)



Exercise 6.2: MLP as a classifier

-The output is a discriminant function

>> load datos_2D_C3_S1.mat



COMPUTER VISION

Exercise 6.2: MLP as a classifier

Using the classified data: >> load datos_2D_C3_S1.mat

a) Choose an **adequate** MLP structure, training set and test set. Compute the confusion matrix and plot the linear classification limits defined by each perceptron of the intermediate layer.

Compute the confusion matrix in the following cases:

b) Changing the **training set** and **test set**

c) Changing the **net structure** (i.e. # of neurons)

COMPUTER VISION

topics

- *Artificial Neural Networks*
- *Perceptron and the MLP structure*
- *The back-propagation learning algorithm*
- *MLP features and drawbacks*
- *The auto-encoder*

MLP features: Mathematical issues

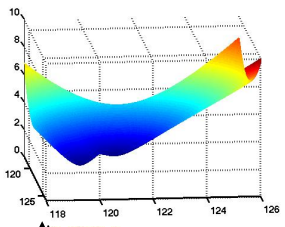
- A two layers MLP can implement any logic function with convex frontier.
- A three layer MLP can impement any logic fucntion with arbitrary frontier.

A two layer MLP can approximate any continuous function with an arbitrary small error

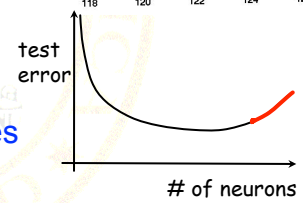
MLP drawbacks

- Learning by minimizing non-linear functions:
  - local minima
  - slow convergence

  (depending on initial values & minimization algorithm)



- Over-learning
- Extrapolation in non learned zones

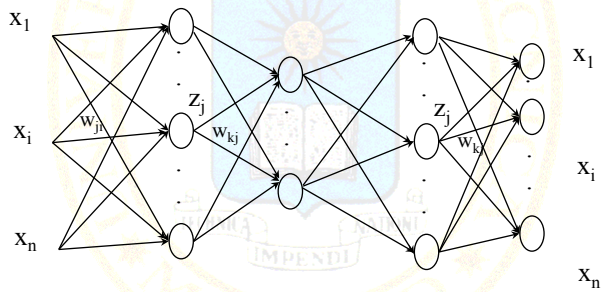


topics

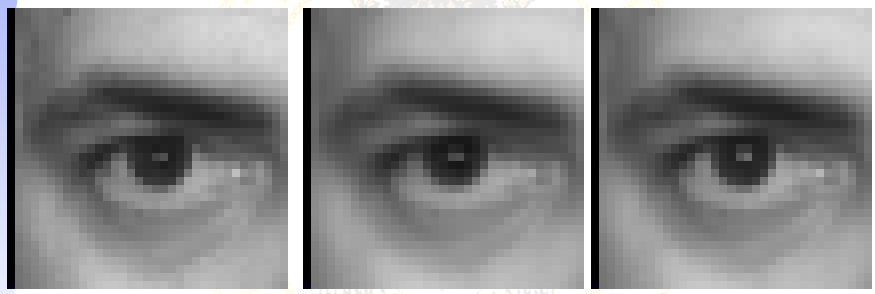
- *Artificial Neural Networks*
- *Perceptron and the MLP structure*
- *The back-propagation learning algorithm*
- *MLP features and drawbacks*
- *The auto-encoder*

Auto-encoder: MLP for dimensionality reduction

- *The desired output is the same as the input and there is a hiden layer having less neurons than dim(x)*

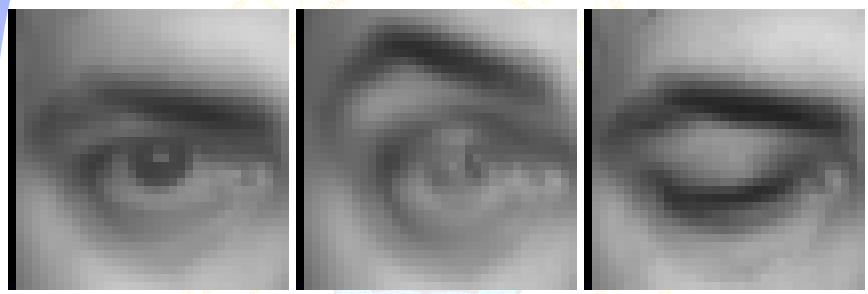


Example: auto-encoder for compression



original | PCA 5 | PCA 25 - MLP 5

Example: auto-encoder for synthesis



1 D (test 1) | 1 D (test 2) | 1 D (test 3) escaled

Auto-encoder: Matlab code

```
% Procesamiento con una MLP para compresioón (salida=entrada)
net=newff(minmax(p_entr),[floor((Dim+ndimred)/2),ndimred,floor
   ((Dim+ndimred)/2),Dim],{'tansig' 'purelin' 'tansig' 'purelin'},
   'trainlm');
[net,tr]=train(net,p_entr,p_entr);

% Creación de una red mitad de la anterior que comprime los datos
netcompr=newff(minmax(p_entr),[floor((Dim+ndimred)/2), ndimred],
   {'tansig' 'purelin'},'trainlm');
netcompr.IW{1}=net.IW{1}; netcompr.LW{2,1}=net.LW{2,1};
netcompr.b{1}=net.b{1}; netcompr.b{2}=net.b{2};

%creación de una red que descomprime los datos
netdescompr=newff(minmax(p_compr),[floor((Dim+ndimred)/2),Dim],
   {'tansig' 'purelin'}, 'trainlm');
netdescompr.IW{1}=net.LW{3,2}; netdescompr.LW{2,1}=net.LW{4,3};
netdescompr.b{1}=net.b{3};      netdescompr.b{2}=net.b{4};
```