**Lab 03 – NN Tool box (Perceptrão e Rede Neuronal)**

* **perceptron: cria uma rede neuronal tipo *perceptrão***

nome\_rede = perceptron

* Por defeito, a função de ativação é a *hardlim* e a função de treino é a *learnp* (podem ser indicadas alternativas utilizando os argumentos opcionais da função *perceptron*)
* **feedforwardnet: cria uma rede neuronal tipo *feedforward***

nome\_rede = feedforwardnet(num\_unid\_hid\_layers, train\_algorithm)

Por defeito, cria uma rede neuronal com uma camada escondida com 10 nós (a arquitetura por defeito pode ser alterada utilizando os argumentos opcionais da função);

* Os inputs e outputs não são indicados neste ponto. A sua dimensão será automaticamente configurada mais tarde durante o processo de treino (também podem ser configurados explicitamente através da função **configure**);
* Funções de ativação por defeito: camadas escondidas (*tansig*) e saída (*purelin*);
* Algoritmo de treino: *trainln*.
* **patternnet Número de camadas escondidas**

nome\_rede=patternnet([num\_unid\_cam1 … num\_unid\_camn])

Exemplo:

% Rede com 2 camadas escondidas (5, 10 unidades) e 1 de saída (1 unidade)

net=patternnet([5 10 1]); view(net);

* **Número máximo de ciclos de treino**

nome\_rede.trainParam.epochs = 100

* **Função de Treino (traingdx = gradient descent)**

nome\_rede.trainFcn = 'nome da função'

Exemplo:

net.trainFcn = 'traingdx'

* **Funções de Ativação**

Nome\_rede{índice da camada}.transfer.Fcn = ‘nome da função’

Exemplo:

% FUNCAO DE ATIVACAO DA CAMADA ESCONDIDA

net.layers{1}.transferFcn = 'logsig';

% FUNCAO DE ATIVACAO DA CAMADA DE SAIDA

net.layers{2}.transferFcn = 'purelin';

* **Percentagem de exemplos de treino e teste (neste caso, 100% treino)**

nome\_rede.divideFcn = ''

* **train: treina a rede neuronal**

nome\_rede = train(nome\_rede, vetor\_de\_input, vetor\_target)

* **sim: testa/simula a rede neuronal**

out = sim(nome\_rede, vetor\_de\_input)

* **view: visualizar a rede neuronal**

view(nome\_rede)

* **configure: configurar o número de entradas e saídas de acordo com os vetores respetivos**

configure(net, vetor\_de\_Input, vetor\_targer)

Takes input data vetor\_de-input and target data vetor\_targer, and configures the network's inputs and outputs to match

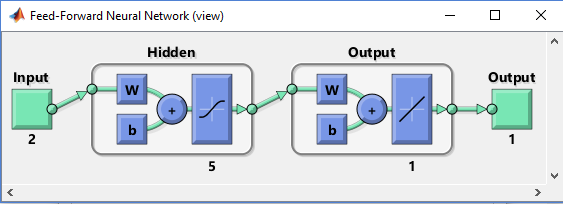
**MATLAB HELP: Usar “HELP nome da função” e/ou “HELP nnnetwork”**

**net.IW (doc IW)**

This property defines the weight matrices of weights going to layers from network inputs. It is always an Nl x Ni cell array, where Nl is the number of network layers (net.numLayers), and Ni is the number of network inputs (net.numInputs).

**net.b (doc b)**

This property defines the bias vectors for each layer with a bias. It is always an Nl x 1 cell array, where Nl is the number of network layers



% Ler todos os W's para 1 só vetor

weights = getwb(net);

fprintf('all weights \n')

disp(weights)

% Ler os W's separadamente por camada e tipo

wIN = net.IW{:}; % input to hidden

wHL = cell2mat(net.LW(2))'; % hidden to output

b1 = net.b{1}; % bias hidden

b2 = net.b{2}; % bias output

fprintf('input weights \n');

disp(wIN);

fprintf('input to output weights \n');

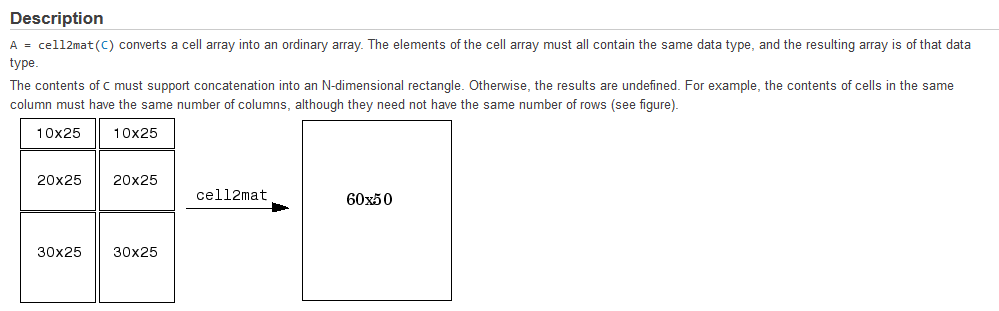
disp(wHL);

fprintf('bias input \n');

disp(b1);

fprintf('bias output \n');

disp(b2)

****

**Plotpv (doc plotpv)**

Plot perceptron input/target vectors

plotpv(P,T) P=input vectos; T=target vector

Plots column vectors in P with markers based on T

**Plotpc**

Plot classification line on perceptron vector plot

plotpc(W,B) takes these inputs (weights, bias) WS x R weight matrix (R must be 3 or less)BS x 1 bias vector and returns a handle to a plotted classification line.

FUNÇÕES DE ACTIVAÇÃO (Transfer Functions)

* compet ⇒ Competitive transfer function
* hardlim ⇒ Hard-limit transfer function (DEGRAU)
* hardlims ⇒ Symmetric hard-limit transfer function (SINAL)
* logsig ⇒ Log-sigmoid transfer function (SIGMOIDE)
* netinv ⇒ Inverse transfer function
* poslin ⇒ Positive linear transfer function
* purelin ⇒ Linear transfer function (LINEAR)
* radbas ⇒ Radial basis transfer function
* radbasn ⇒ Normalized radial basis transfer function
* satlin ⇒ Saturating linear transfer function
* satlins ⇒ Symmetric saturating linear transfer function
* softmax ⇒ Soft max transfer function
* tansig ⇒ Hyperbolic tangent sigmoid transfer function (TANGENTE HIPERBOLICA)
* tribas ⇒ Triangular basis transfer function

FUNÇÕES DE TREINO (Trainning Functions)

* trainbfg ⇒ BFGS quasi-Newton backpropagation
* trainbfgc ⇒ BFGS quasi-Newton backpropagation for use with NN model reference adaptive controller
* trainbr ⇒ Bayesian regulation backpropagation
* trainbu ⇒ Batch unsupervised weight/bias training
* trainc ⇒ Cyclical order weight/bias training
* traincgb ⇒ Conjugate gradient backpropagation with Powell-Beale restarts
* traincgf ⇒ Conjugate gradient backpropagation with Fletcher-Reeves updates
* traincgp ⇒ Conjugate gradient backpropagation with Polak-Ribiére updates
* traingd ⇒ Gradient descent backpropagation
* traingda ⇒ Gradient descent with adaptive learning rate backpropagation
* traingdm ⇒ Gradient descent with momentum backpropagation
* traingdx ⇒ Gradient descent with momentum and adaptive learning rate backpropagation
* trainlm ⇒ Levenberg-Marquardt backpropagation
* trainoss ⇒ One-step secant backpropagation
* trainr ⇒ Random order incremental training with learning functions
* trainrp ⇒ Resilient backpropagation
* trainru ⇒ Unsupervised random order weight/bias training
* trains ⇒ Sequential order incremental training with learning functions
* trainscg ⇒ Scaled conjugate gradient backpropagation

FUNÇÕES DE APRENDIZAGEM

* learncon ⇒ Conscience bias learning function
* learngd ⇒ Gradient descent weight and bias learning function
* learngdm ⇒ Gradient descent with momentum weight and bias learning function
* learnh ⇒ Hebb weight learning rule
* learnhd ⇒ Hebb with decay weight learning rule
* learnis ⇒ Instar weight learning function
* learnk ⇒ Kohonen weight learning function
* learnlv1 ⇒ LVQ1 weight learning function
* learnlv2 ⇒ LVQ2.1 weight learning function
* learnos ⇒ Outstar weight learning function
* learnp ⇒ Perceptron weight and bias learning function
* learnpn ⇒ Normalized perceptron weight and bias learning function
* learnsom ⇒ Self-organizing map weight learning function
* learnsomb ⇒ Batch self-organizing map weight learning function
* learnwh ⇒ Widrow-Hoff weight/bias learning function

**Diferença entre funções de treino (training) e aprendizagem (learning)**

The training function is the overall algorithm that is used to train the neural network to recognize a certain input and map it to an output. A common example is [backpropagation](http://en.wikipedia.org/wiki/Backpropagation) and its many variations and weight/bias training. A learning function deals with individual weights and thresholds and decides how those would be manipulated. These usually (but not always) employ some form of [gradient descent](http://en.wikipedia.org/wiki/Gradient_descent). Examples include [simulated annealing](http://en.wikipedia.org/wiki/Simulated_annealing), [Silva and Almeida's algorithm](http://www.lx.it.pt/~lbalmeida/papers/AlmeidaHNC.pdf), using [momentum](http://www.willamette.edu/~gorr/classes/cs449/Momentum/momentum.html) and [adaptive learning-rates](http://www.math.upatras.gr/~dgs/papers/reports/tr98-02.pdf), and weight-learning (examples include [Hebb](http://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/slides/hebbpca.pdf), [Kohonen](http://www.cs.bham.ac.uk/~jlw/sem2a2/Web/Kohonen.htm), etc.) algorithms.