ARTIFICIAL NEURAL NETWORK LAB CLASS GUIDE

Task 1 – Compute a neuron output

Objective: Calculate and visualize the output of a single neuron.

1) Define neuron parameters

```
% Neuron weights
w = [4 -2];
% Neuron bias
b = -3;
% Activation function: Hyperbolic tangent sigmoid function
func = 'tansig';

% Activation function: Logistic sigmoid transfer function
% func = 'logsig'

% Activation function: Hard-limit transfer function (threshold)
% func = 'hardlim'

% Activation function: Linear transfer function
% func = 'purelin'
```

2) Define input vectors

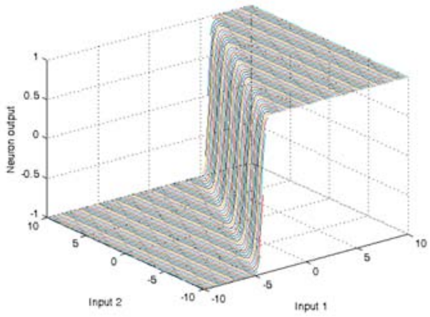
```
p = [2 3]
```

3) Calculate neuron output

```
activation_potential = p*w'+b;
neuron_output = feval(func, activation_potential)
```

4) Plot neuron output over the range of inputs

```
[p1,p2] = meshgrid(-10:.25:10);
z = feval(func, [p1(:) p2(:)]*w'+b );
z = reshape(z,length(p1),length(p2));
plot3(p1,p2,z);
grid on;
xlabel('Input 1');
ylabel('Input 2');
zlabel('Neuron output');
```



5) Change the activation function and plot neuron output again to see the different output surfaces

Activation/Transfer functions:
hardlim: Positive hard limit transfer function; hardlims: Symmetric hard limit transfer function; purelin: Linear transfer function; satlin: Positive saturating linear transfer function; logsig: Logistic sigmoid transfer function; tansig: Hyperbolic tangent sigmoid symmetric transfer function

Task 2 – Analyze a single neuron

Objective: Analyze the change in the output of a single neuron when changing the weight, the bias and the function.

1) Run demo nnd2n1

```
nnd2n1
```

2) Study how the different values of weight, bias, transfer function and input p modify the output of the neuron

Task 3 – Custom networks

Objective: Create and view custom neural networks.

1) Define one sample: inputs and outputs. An example of one instance of 6 input values that has as output two output values:

```
inputs = [1:6]'; % input vector (6-dimensional pattern)
outputs = [1 2]'; % corresponding target output vector
```

You can modify them if you like. For example you can have 6 instances of 1 input variable that have as output 1 output value.

```
inputs = [1:6]; % input vector (6-dimensional pattern)
outputs = [7:12]; % corresponding target output vector
```

2) Define and custom the network

```
% create the network: 1 input, 2 layer (1 hidden layer and 1 output layer), feed-forward
network
net = network( ...
1,              ... % numInputs (number of inputs)
2,              ... % numLayers (number of layers)
[1; 0],         ... % biasConnect (numLayers-by-1 Boolean vector)
[1; 0],         ... % inputConnect (numLayers-by-numInputs Boolean matrix)
[0 0; 1 0],     ... % layerConnect (numLayers-by-numLayers Boolean matrix); [a b; c d]
                ... % a: 1st-layer with itself, b: 2nd-layer with 1st-layer,
                ... % c: 1st-layer with 2nd-layer, d: 2nd-layer with itself
[0 1]           ... % outputConnect (1-by-numLayers Boolean vector)
);
% View network structure
view(net);
```

We can then see the properties of sub-objects as follows:

```
net.inputs{1}
net.layers{1}, net.layers{2}
net.biases{1}
net.inputWeights{1}, net.layerWeights{2}
net.outputs{2}
```

3) Define topology and transfer function

```
% number of hidden layer neurons
net.layers{1}.size = 5;
% hidden layer transfer function
net.layers{1}.transferFcn = 'logsig';
view(net);
```

4) Configure the network

```
net = configure(net,inputs,outputs);
view(net);
```

5) Train net and calculate neuron output

```
% initial network response without training (the network is resimulated)
initial_output = net(inputs)
```

We can get the weight matrices and bias vector as follows:

```
net.IW{1}
net.LW{2}
net.b{1}

% network training
net.trainFcn = 'trainlm';   % trainlm: Levenberg-Marquardt backpropagation
% trainlm is often the fastest backpropagation algorithm in the toolbox,
% and is highly recommended as a first choice supervised algorithm,
% although it does require more memory than other algorithms.
net.performFcn = 'mse';
net = train(net,inputs,outputs);
% network response after training (the network is resimulated)
final_output = net(inputs)

% final weight matrices and bias vector:

net.IW{1}
net.LW{2}
net.b{1}
```

Task 4 – Changing the number of neurons

Objective: Analyze the change in the number of neurons in the hidden layer. How increasing of hidden layer neurons affects to function approximation? Are there any side-effects if number of hidden layer neurons is high?

1) Run demo nnd11gn

```
nnd11gn
```

Task 5 – Classification of linearly separable data with a perceptron

Objective: Two clusters of data, belonging to two classes, are defined in a 2-dimensional input space. Classes are linearly separable. The task is to construct a Perceptron for the classification of data.

Recall: The simplest kind of neural network is a *single-layer perceptron* network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feed-forward network. Perceptrons can be trained by a simple learning algorithm that is usually called the *delta rule*. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights, thus implementing a form of gradient descent.

1) Define input and output data

```
% number of samples of each class
N = 20;
% define inputs and outputs
offset = 5; % offset for second class
x = [randn(2,N) randn(2,N)+offset]; % inputs
y = [zeros(1,N) ones(1,N)]; % outputs
% Plot input samples with plotpv (Plot perceptron input/target vectors)
figure(1)
plotpv(x,y);
```

2) Create and train the perceptron

```
net = perceptron;
net = train(net, x, y);
view(net);
```

3) Plot decision boundary

```
figure(1)
plotpc(net.IW{1},net.b{1});
% Plot a classification line on a perceptron vector plot
```

Task 6 – Classification of a 4-class problem with a perceptron

Objective: Perceptron network with 2-inputs and 2-outputs is trained to classify input vectors into 4 categories.

1) Define input and output data

```
close all, clear all, clc
```

```
% number of samples of each class
K = 30;
% define clases
q = .6; % offset of clases
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clases
plot(A(1,:),A(2,:),'bs')
hold on
grid on
plot(B(1,:),B(2,:),'r+')
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'m*')

% text labels for clases
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class C')
text(.5-q,.5-2*q,'Class D')

% define output coding for clases
a = [0 1]';
b = [1 1]';
c = [1 0]';
d = [0 0]';
```

2) Prepare inputs and outputs for perceptron training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...  % repmat: Replicate and tile an array
repmat(c,1,length(C)) repmat(d,1,length(D)) ];
plotpv(P,T);
```

3) Create a perceptron

```
net = perceptron;
```

4) Train a perceptron (step by step in order to allow the visual adjustment of the network) *Adapt* returns a new network object that performs as a better classifier, the network output, and the error. This loop allows the network to adapt for xx passes, plots the classification line and continues until the error is zero.

```
% To see the adaptation you need to look at the plot while the code is running
E = 1;
net.adaptParam.passes = 1;
linehandle = plotpc(net.IW{1},net.b{1});
n = 0;
while (sse(E) & n<1000)  % sse: Sum squared error
```

```
    n = n+1;
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end
% show perceptron structure
view(net);
```

5) How to use trained perceptron

```
% For example, classify an input vector of [0.7; 1.2]
p = [0.7; 1.2]
y = net(p)
% compare response with output coding (a,b,c,d)
```

Task 7 – Classification of a 4-class problem with a multilayer perceptron

Objective: 4 cluster of data (A,B,C,D) are defined in a 2-dimensional input space. The task is to define a neural network for classification of arbitrary point in the 2-dimensional space into one of the classes (A,B,C,D).

1) Define 4 clusters of input data

```
close all, clear all, clc
% number of samples of each class
K = 100;
% define 4 clusters of input data
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'b*')
plot(C(1,:),C(2,:),'kx')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class C')
text(.5-q,.5-2*q,'Class D')
```

2) Define output coding for all 4 clusters

```
% coding (+1/-1) of 4 separate classes
a = [-1 -1 -1 +1]';
b = [-1 -1 +1 -1]';
d = [-1 +1 -1 -1]';
c = [+1 -1 -1 -1]';
```

3) Prepare inputs and outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))...
repmat(c,1,length(C)) repmat(d,1,length(D))];
```

4) Create and train a multilayer perceptron

```
% create a neural network
net = feedforwardnet([4 3]); %Number of hidden neurons in each layer
% train net
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio = 0; % validation set [%]
net.divideParam.testRatio = 0; % test set [%]
% train a neural network
[net,tr,Y,E] = train(net,P,T); % Y:Output; E:Error

% show network
view(net)
```

5) Evaluate network performance and plot results

```
% evaluate performance: decoding network response
[m,i] = max(T); % target class
[m,j] = max(Y); % predicted class
N = length(Y); % number of all samples
k = 0; % number of missclassified samples
if find(i-j), % if there exist missclassified samples
  k = length(find(i-j)); % get a number of missclassified samples
end
fprintf('Correct classified samples: %.1f%% samples\n', 100*(N-k)/N)
% plot network output
figure;
subplot(211)
plot(T')
title('Targets')
ylim([-2 2])
grid on
subplot(212)
plot(Y')
title('Network response')
xlabel('# sample')
ylim([-2 2])
grid on
```

6) Plot classification result for the complete input space

```
% generate a grid
span = -1:.01:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simualte neural network on a grid
aa = net(pp);
% plot classification regions based on MAX activation
figure(1)
m = mesh(P1,P2,reshape(aa(1,:),length(span),length(span))-5);
set(m,'facecolor',[1 0.2 .7],'linestyle','none');
hold on
m = mesh(P1,P2,reshape(aa(2,:),length(span),length(span))-5);
```

```
set(m,'facecolor',[1 1.0 0.5],'linestyle','none');
m = mesh(P1,P2,reshape(aa(3,:),length(span),length(span))-5);
set(m,'facecolor',[.4 1.0 0.9],'linestyle','none');
m = mesh(P1,P2,reshape(aa(4,:),length(span),length(span))-5);
set(m,'facecolor',[.3 .4 0.5],'linestyle','none');
view(2)
```

Task 8 – Solving XOR problem with a multilayer perceptron

Objective: 4 cluster of data (A,B,C,D) are defined in a 2-dimensional input space. (A,C) and (B,D) clusters represent XOR classification problem. The task is to define a neural network for solving the XOR problem.

1) Define 4 clusters of input data

```
close all, clear all, clc, format compact
% number of samples of each class
K = 100;
% define 4 clusters of input data
q = .6; % offset of classes
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
% plot clusters
figure(1)
plot(A(1,:),A(2,:),'k+')
hold on
grid on
plot(B(1,:),B(2,:),'bd')
plot(C(1,:),C(2,:),'k+')
plot(D(1,:),D(2,:),'bd')
% text labels for clusters
text(.5-q,.5+2*q,'Class A')
text(.5+q,.5+2*q,'Class B')
text(.5+q,.5-2*q,'Class A')
text(.5-q,.5-2*q,'Class B')
```

2) Define output coding for XOR problem

```
% encode clusters a and c as one class, and b and d as another class
a = -1; % a | b
c = -1; % -------
b = 1; % d | c
d = 1; %
```

3) Prepare inputs and outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B C D];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
repmat(c,1,length(C)) repmat(d,1,length(D)) ];
% view inputs |outputs
% [P' T']
```

4) Create and train a multilayer perceptron

```
% create a neural network
% The sizes of the network are set to 0. These sizes will automatically be configured to
% match particular data by train
net = feedforwardnet([5 3]);
% train net
net.divideParam.trainRatio = 1; % training set
net.divideParam.valRatio = 0; % validation set
net.divideParam.testRatio = 0; % test set
% train a neural network
[net,tr,Y,E] = train(net,P,T);
% show network
view(net)
```

5) Plot targets and network response to see how good the network learns the data

```
figure(2)
plot(T','linewidth',2)
hold on
plot(Y','r--')
grid on
legend('Targets','Network response','location','best')
ylim([-1.25 1.25])
```

6) Plot classification result for the complete input space (separation by hyperplanes)

```
% generate a grid
span = -1:.005:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simulate neural network on a grid
aa = net(pp);
% translate output into [-1,1]
% aa = -1 + 2*(aa>0);
 % plot classification regions
figure(1)
mesh(P1,P2,reshape(aa,length(span),length(span))-5);
colormap cool
view(2)
```

Task 9 – Solving XOR problem with a RBFN

Objective: 2 groups of linearly inseparable data (A,B) are defined in a 2-dimensional input space. The task is to define a neural network for solving the XOR classification problem.

1) Create input data

```
close all, clear all, clc
% number of samples of each cluster
K = 100;
% offset of clusters
q = .6;
% define 2 groups of input data
A = [rand(1,K)-q rand(1,K)+q;
rand(1,K)+q rand(1,K)-q];
```

```
B = [rand(1,K)+q rand(1,K)-q;
rand(1,K)+q rand(1,K)-q];
% plot data
plot(A(1,:),A(2,:),'k+',B(1,:),B(2,:),'b*')
grid on
hold on
```

2) Define output coding

```
% coding (+1/-1) for 2-class XOR problem
a = -1;
b = 1;
```

3) Prepare inputs and outputs for network training

```
% define inputs (combine samples from all four classes)
P = [A B];
% define targets
T = [repmat(a,1,length(A)) repmat(b,1,length(B))];
```

4) Create a RBFN

```
% NEWRB algorithm
% The following steps are repeated until the network's mean squared error
% falls below goal:
% 1. The network is simulated
% 2. The input vector with the greatest error is found
% 3. A radial base neuron is added with weights equal to that vector
% 4. The purelin layer weights are redesigned to minimize error


% choose a spread constant
%The larger spread is, the smoother the function approximation. Too large a spread means
%a lot of neurons are required to fit a fast-changing function. Too small a spread means
%many neurons are required to fit a smooth function, and the network might not generalize
%well. Call newrb with different spreads to find the best value for a given problem.
spread = 2;
% choose max number of neurons
K = 20;
% performance goal (SSE)
goal = 0;
% number of neurons to add between displays
Ki = 4;
% create a neural network
net = newrb(P,T,goal,spread,K,Ki);
% view network
view(net)
```

5) Evaluate network performance

```
% simulate RBFN on training data
Y = net(P);
% calculate [%] of correct classifications
correct = 100 * length(find(T.*Y > 0)) / length(T);
fprintf('\nSpread = %.2f\n',spread)
fprintf('Num of neurons = %d\n',net.layers{1}.size)
fprintf('Correct class = %.2f %%\n',correct)
% plot targets and network response
```

```
figure;
plot(T')
hold on
grid on
plot(Y','r')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
legend('Targets','Network response')
xlabel('Sample No.')
```

6) Plot classification result

```
% generate a grid
span = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp = [P1(:) P2(:)]';
% simualte neural network on a grid
aa = sim(net,pp);
% plot classification regions based on MAX activation
figure(1)
ma = mesh(P1,P2,reshape(-aa,length(span),length(span))-5);
mb = mesh(P1,P2,reshape( aa,length(span),length(span))-5);
set(ma,'facecolor',[1 0.2 .7],'linestyle','none');
set(mb,'facecolor',[1 1.0 .5],'linestyle','none');
view(2)
```

7) Plot RBFN centers (separation by RBF gaussian neurons)

```
plot(net.iw{1}(:,1),net.iw{1}(:,2),'gs')
```

Remember similarities/differences MLP and RBFNN

SIMILARITIES
1. Both have the Universal Approximation property: they can approximate any continuous mapping with arbitrary accuracy (with only one hidden layer)
2. Both can be expressed as a feed-forward network (any number of hidden layers for the MLP and one hidden layer for the RBFNN)
3. Both can be trained with gradient-based methods to optimize the whole set of weights

DIFFERENCES
1. An MLP performs a global and distributed approximation of the underlying function, whereas the RBFNN performs a local approximation
2. An MLP partitions the input space with hyperplanes; the RBFNN decision boundaries are hyperellipsoids (or hyperspehres)
3. The distributed representation of MLPs causes the error surface to have multiple local minima and nearly flat regions. As a result training times are usually larger than those for RBFNNs
4. MLPs typically require fewer neurons than RBFNNs to approximate a non-linear function with the same accuracy
5. MLPs typically generalize better than RBFNNs in regions outside the local neighborhoods defined by the training set
6. All the MLP parameters are trained simultaneously; in the RBFNN the non-linear parameters are typically trained prior and separately, leading to an efficient, much faster algorithm

7. The RBFNN arises naturally from regularization theory, whereas the MLP needs ad hoc regularization

Task 10 – NN Apps Matlab

Take a look to the NN Apps (NN Fitting, NN Time Series, NN Pattern Recognition)