

## Neural and Evolutionary Computing.

### Lab 2: Approximation and prediction using neural networks

---

#### 1. Approximation and prediction

Aim: find a (nonlinear) model which describes the dependence between some input data (predictors) and some output data (predicted values) .

Examples:

- Experimental data analysis:
  - The input data are results of measurements of some characteristics (e.g. features of a car, computer or house)
  - The results are values which are considered related to the measured values (the price of the car, computer or house)
- Time series prediction:
  - The input data are previous values in a time series (e.g. daily temperature recorded during one month/year, daily exchange rate etc.)
  - The result is the next value in the series

#### 2. Using feedforward neural networks for approximation/prediction

*Main steps:*

- a) Construction of the training set (it depends on the problem to be solved)
- b) Choice of the network architecture (number of layers, number of units on each layer, activation functions)
- c) Choice of the training algorithm and of its parameters (learning rate, training accuracy, maximal number of training epochs).
- d) Training
- e) Testing

*Implementation of a feedforward neural network trained using BackPropagation – SciLab functions ([www.scilab.org](http://www.scilab.org))*

Network creation:

```
function network=NNcreate(N, K, M, activationFunction1, activationFunction2)
    network=tlst(["Retea feedforward", "N", "K", "M", "X0", "Y0", "X1", "Y1", "X2", "Y2", "W1", "W2",
                "f1", "f2"], N, K, M, zeros(N+1,1), zeros(N+1,1), zeros(K,1), zeros(K+1,1), zeros(M,1),
                zeros(M,1), zeros(K,N+1), zeros(M,K+1), activationFunction1, activationFunction2);
endfunction
```

Activation functions (used in BackPropagation neural networks)

```
// logistic function
function output=logistic(x)
    output=1/(1+exp(-x));
endfunction
```

```
// linear function
function output=linear(x)
    output=x;
endfunction
```

Remark. The hyperbolic tangent (tanh) is predefined in SciLab

Network functioning: computing the output signal (Forward step in the BackPropagation algorithm):

```
function network=forward(network, X)
    network.X0=[-1;X]; network.Y0=network.X0;
    network.X1=network.W1*network.Y0;
    network.Y1=[-1; feval(network.X1, network.f1)];
    network.X2=network.W2*network.Y1;
    network.Y2=feval(network.X2, network.f2);
endfunction
```

Error computation (Mean Squared Error):

```
function err=error_computation(network, tset)
    err=0;
    L=tset.L;
    for i=1:L
        network=forward(network, tset.X(:,i));
        delta=(tset.d(i)-network.Y2).^2;
        err=err+sum(delta);
    end
    err=err/L;
endfunction
```

Weights adjustment (BackPropagation algorithm):

```
function [network, ap, aE]=BP(network, tset, pmax, Emax, eta)
    L=tset.L;
    // initialization of the weights with random values in [-1,1]
    network.W1=2*rand(network.W1)-ones(network.W1);
    network.W2=2*rand(network.W2)-ones(network.W2);
    E=error_computation(network, tset); // calcul eroare
    p=0; // training epoch counter
    ap=[]; aE=[];
    while p<pmax & E>Emax
        for l=1:L
            network=forward(network, tset.X(:,l));
            delta2=tset.d(l)-network.Y2; // error corresponding to the output layer
            if (network.f2==logistic) delta2=delta2.*(network.Y2.*(ones(network.Y2)-network.Y2)); end;
            if (network.f2==tanh) delta2=delta2.*(ones(network.Y2)-network.Y2.^2); end;
            delta=network.W2'*delta2; // error propagation
            if (network.f1==logistic) delta=delta.*(network.Y1.*(ones(network.Y1)-network.Y1)); end;
            if (network.f1==tanh) delta=delta.*(ones(network.Y1)-network.Y1.^2); end;
            delta1=delta(2:length(delta)); // ignoring the dummy component
            network.W1=network.W1+eta*(delta1*network.Y0');
            network.W2=network.W2+eta*(delta2*network.Y1');
        end;
        p=p+1;
        E=error_computation(network, tset);
        ap=[p; E];
    end;
end;
```

```

E=error_computation(network,tset);
disp(E,"Eroare=",p,"p=");
ap=[ap p]; aE=[aE E];
p=p+1; // incrementation of the training epoch counter
end
endfunction

```

### Application 1. XOR function representation (fișier BP\_XOR.sci)

Construction of the training set:

```

function tset=trainingSet()
    tset=tlist(["Set antrenare","L","X","d"],4,[-1 -1 1 1;-1 1 -1 1],[-1 1 1 -1]);
endfunction

```

Approximation of the XOR function using a neural network with two hidden units and tanh activation functions

```

function rna=approximation_XOR(hiddenUnits, eta, epochs)
rn=NNcreate(2,hiddenUnits,1,tanh,tanh); // creation of the network
tset=trainingSet(); // training set construction
rna=BP(rn,tset,epochs,0.0001,eta); // network training
// testing
for i=1:tset.L;
    rna=forward(rna,tset.X(:,i));
    disp(tset.X(:,i),"input=");
    disp(rna.Y2,"output=");
end
endfunction

```

Call of the approximation function:

```

approximation_XOR(10,0.1,1000);

```

### Exercises:

1. Analyze the influence of the number of hidden units and of the learning rate value on the network performance. (Hint: values for hiddenUnits: 1,2,5,20; values for eta: 0.01,0.1,0.5)
2. Analyze the influence of the activation function on the network performance: (logistic, logistic), (tanh,logistic), (logistic, tanh). (Hint: when a logistic is used at the output layer the training set should contain [0,1,1,0] instead of [-1,1,1,-1])

### Application 2. Approximation of a nonlinear function (f:[a,b]->R) starting from a sets of points which are close to its graph. (file BP\_regression.sci)

The input data are selected by placing the mouse cursor on the desired position and by clicking the right button; the points selection is finalized once the left button of the mouse is clicked. This is done using the following function.

```

function tset=trainingSet()
    tset=tlst(["Training set","L","X","d"],0,zeros(1,100),zeros(1,100));
    clf;
    plot2d(0,0,0,rect=[0,0,10,10]); // defining the data ranges
    xgrid(3);
    x=locate(-1,1);
    tset.L=size(x,2);
    tset.X(1:tset.L) = x(1,:);
    tset.d(1:tset.L) = x(2,:);
endfunction

```

Approximating the function starting with the selected points:

```

function tset=regression(hiddenUnits, eta, epochs)
rn=NNcreate(1,hiddenUnits,1,logistic,linear); // network creation
disp("dupa creare retea");
tset=trainingSet(); // training set construction
inf=min(tset.X); sup=max(tset.X);
rna=BP(rn,tset,epochs,0.0001,eta); // network training
// network testing
x=inf:(sup-inf)/100:sup;
y=[];
for i=1:size(x,2);
    rna=forward(rna,x(1,i));
    y=[y rna.Y2];
end
plot(tset.X,tset.d,'b*',x,y,'r-');
endfunction

```

Call of the regression function:

```
regression(10,0.01,10000)
```

### Exercises:

1. Analyze the influence of the number of hidden units and the value of the learning rate on the network performance (Hint: testing values for hiddenUnits: 1,2,5,20; testing values for eta: 0.01,0.1,0.5)
2. Analyze the influence of the hidden layer activation function (tanh instead of logistic)

### Application 3. Modelling a time series (file BP\_prediction.sci)

Let us consider a time series  $x_1, x_2, \dots, x_n$ . The aim is to estimate the value corresponding to time moment  $(n+1)$ . The main idea is to suppose that  $x_i$  depends on  $N$  previous values:  $x_{i-1}, x_{i-2}, \dots, x_{i-N}$ .

Using this assumption one can design a neural network which learns the dependence between a value in the series and  $N$  previous values. Thus the network will have  $N$  input units and an output unit. The training set will have  $L=n-N$  pairs of (input, desired output) i.e:

$$\{(x_1, x_2, \dots, x_N), x_{N+1}), ((x_2, x_3, \dots, x_{N+1}), x_{N+2}), \dots, ((x_{n-N}, x_{n-N+1}, \dots, x_{n-1}), x_n)\}.$$

Let us suppose that a text file contains values of the Euro-Lei exchange rate (one value per line – first line contains the most recent value).

Training set construction:

```
function tset=trainingSet(inputUnits)
    tset=tblstruct("Training set","L","X","d",0,zeros(1,100),zeros(1,100));
    date1=csvread("d:/z/cne/lab/lab2/cursEuroZilnic.csv");
    date2(1:length(date1))=date1(length(date1):-1:1);
    tset.L=size(date2,2)-inputUnits;
    tset.X=zeros(inputUnits,tset.L);
    tset.d=zeros(1,tset.L);
    for i=1:tset.L
        tset.X(:,i)=date2(1,i+inputUnits-1);
        tset.d(i)=date2(1,i+inputUnits);
    end
endfunction
```

Network training and testing:

```
function tset=prediction(inputUnits, hiddenUnits, eta, epochs)
    rn=NNcreate(inputUnits,hiddenUnits,1,logistic,linear);
    tset=trainingSet(inputUnits);
    rna=BP(rn,tset,epochs,0.0001,eta);
    y=[];
    for i=1:tset.L;
        rna=forward(rna,tset.X(:,i));
        y=[y rna.Y2];
    end
    plot(tset.d,'b-',y,'r-');
endfunction
```

*SciLab implementation of RBF networks:*

Creating an RBF network§

```
function network=RBFcreate(N, K, M, sigma)
    network=tlst(["RBF NN", "N", "K", "M", "X0", "Y0", "X1", "Y1", "X2", "Y2", "C", "W", "f", "sigma"],
                N, K, M, zeros(N,1), zeros(N,1), zeros(K,1), zeros(K+1,1), zeros(M,1), zeros(M,1),
                zeros(K,N), zeros(M,K+1), gaussian,sigma);
endfunction
```

Activation function:

```
function output=gaussian(x)
    output=exp(-x^2/2);
endfunction
```

Aggregation function:

```
function d=dist(x, y)
    d=sqrt((x-y)*(x-y));
endfunction
```

Output signal computation:

```
function network=forward(network, X)
    network.X0=X; network.Y0=network.X0;
    for k=1:network.K
        network.X1(k)=dist(network.C(k)',network.Y0);
    end
    network.Y1=[-1; feval(network.X1./network.sigma,network.f)];
    network.X2=network.W*network.Y1;
    network.Y2=network.X2;
endfunction
```

Error function computation:

```
function err=error_computation(network, tset)
    err=0;
    [N,L]=size(tset.X);
    for i=1:L
        network=forward(network,tset.X(:,i));
        delta=(tset.d(i)-network.Y2).^2;
        err=err+sum(delta);
    end
    err=err/L
endfunction
```

Training algorithm (the centers are identical with the input data)

```
function [network, ap, aE]=train(network, tset, pmax, Emax, eta)
L=tset.L;
network.C=tset.X';
network.W=2*rand(network.W)-ones(network.W);
E=error_computation(network,tset);
p=0;
ap=[];aE=[];
while p<pmax & E>Emax
    E=error_computation(network,tset);
    for i=1:L
        network=forward(network,tset.X(:,i));
        y=network.W*network.Y1;
        delta=tset.d(i)-y;
        network.W=network.W+eta*(delta*network.Y1')
    end;
    E=error_computation(network,tset);
    if (modulo(p,10)==0) then
        disp(p,"Iteration:");
        disp(E,"error=");
        ap=[ap p]; aE=[aE E];
    end
    p=p+1;
end
disp(E,"error=");
endfunction
```

**Example:** nonlinear regression (file RBFnetwork.sci)

Training set construction (noisy sinus data):

```
function tset=trainingSet()
tset=lst("Training set", "L", "X", "d", 0, zeros(1,100), zeros(1,100));
tset.X=[0:1:10*pi];
tset.L=length(tset.X);
tset.d=sin(tset.X)+0.2*rand(tset.X)-0.1*ones(tset.X);
endfunction
```

Network creation, training and output visualization:

```
function tset=regressionRBF(eta, epochs, sigma)
tset=trainingSet();
rbf=RBFcreate(1, length(tset.X), 1, sigma);
disp(tset, "tset=");
[rbfa, ap, aE]=train(rbf, tset, epochs, 0.0001, eta);
test_plot(tset, rbfa);
pause; // continue by typing <resume>
clf;
plot(ap, aE);
endfunction
```

**Homework:**

Change the training algorithm of the RBF network such that the prototype vectors are established in an incremental way (see lecture 2-3).