1. Multilayer Perceptron based Neural Network (2 hidden layer)

A = load('D:\CMS\Sem3-1\Neural Networkand Fuzzy Logic\data5.mat');

A=A.x;

data=A;

%Input normalization

data(:,1:end-1) = (data(:,1:end-1)-mean(data(:,1:end-1)))./std(data(:,1:end-1));

X = [ones(size(data,1),1) data(:,1:end-1)]; %Inputs

Y = data(:,end); %Target outputs

MSETarget = 0.01; %Target MSE

H = 6; %Number of neurons in hidden layer 1

Q = 6; %Number of neurons in hidden layer 2

w = rand(size(X,2),H)/10; %Weights between input layer and hidden layer 1

v = rand(H+1,Q)/10; %Weights between hidden layer 1 and hidden layer 2

u = rand(Q+1,3)/10; %Weights between hidden layer 2 and output layer

lr = 0.01; %learning rate

%Converting target outputs to 3 output neurons

for i=1:length(Y)

if Y(i)==1

yt(i,:) = [1 0 0];

else

yt(i,:) = [0 0 1];

end

end

%Randomly divide the dataset into training (70%) and testing (30%) set

p = randperm(length(Y));

trainInput = X(p(1:ceil(0.7\*size(X,1))),:); trainOutput = yt(p(1:ceil(0.7\*size(X,1))),:);

testInput = X(p(ceil(0.7\*size(X,1))+1:end),:); testOutput = yt(p(ceil(0.7\*size(X,1))+1:end),:);

it = 1;

er=1000;

while er>MSETarget %Stopping condition

er = 0;

p = randperm(length(trainOutput)); %Shuffling data before each iteration

for t=1:length(trainOutput)

z1 = logsig(w'\*trainInput(p(t),:)'); %Output of hidden layer 1

z1 = [1;z1]; %Concatenating input with 1 for bias

z2 = logsig(v'\*z1); %Output of hidden layer 2

z2 = [1;z2]; %Concatenating hidden layer 1 output with 1 for bias

y = logsig(u'\*z2); %Output of output layer after sigmoid activation

del = (trainOutput(p(t),:)'-y).\*y.\*(1-y);

for q=1:Q+1

delu(q,:) = -lr\*del'\*z2(q); %Change in u

end

z2 = z2(2:end);

delv = zeros(H+1,Q); %Change in v

for h=1:H+1

for q=1:Q

for k=1:3

delv(h,q) = delv(h,q)-lr\*del(k)\*u(q+1,k)\*z2(q)\*(1-z2(q))\*z1(h);

end

end

end

delw = zeros(size(X,2),H); %Change in w

z1 = z1(2:end);

for j=1:size(trainInput,2)

for h=1:H

for q=1:Q

for k=1:3

delw(j,h) = delw(j,h)-lr\*del(k)\*u(q+1,k)\*z2(q)\*(1-z2(q))\*v(h+1,q)\*z1(h)\*(1-z1(h))\*trainInput(p(t),(j));

end

end

end

end

%Weight updates

w = w-delw;

v = v-delv;

u = u-delu;

er = er+sum((trainOutput(p(t),:)'-y).^2); %Error calculation

end

er = er/(3\*length(trainOutput)); %Cost calculation

cost(it,1) = er;

it = it+1;

end

plot(cost) %Plotting cost

xlabel('Number of iterations');

ylabel('Cost function');

title('Cost vs Number of iterations');

% Testing using testInput

for i=1:length(testOutput)

z1 = logsig(w'\*testInput(i,:)'); %Hidden layer 1 output

z1 = [1;z1]; %Concatenating with 1 for bias

z2 = logsig(v'\*z1); %Hidden layer 2 output

z2 = [1;z2]; %Concatenating with 1 for bias

y = logsig(u'\*z2); %Predicted output

[~,yp(i)] = max(y); %Determining the class of predicted output

end

[~,ya] = max(testOutput,[],2); %Determining the class of test output

[cm, ~] = confusionmat(yp,ya); %Calculating confusion matrix

IA = zeros(1,2);

OA = 0;

for i = 1:2

IA(i) = cm(i,i)/sum(cm(i,:)); %individual accuracy

OA = OA + cm(i,i);

end

OA = OA/sum(cm(:)) %overall accuracy

IA = [ 0.9404, 0.9561]

OA = 0.9488



1. Radical Basic Function Neural Network(RBFNN)

A = load('D:\CMS\Sem3-1\Neural Networkand Fuzzy Logic\data5.mat');

A=A.x;

data=A;

%Input normalization

data(:,1:end-1) = (data(:,1:end-1)-mean(data(:,1:end-1)))./std(data(:,1:end-1));

X = data(:,1:end-1); %Inputs

Y = data(:,end); %Target outputs

%Converting target output to 2 output neurons

for i=1:length(Y)

if Y(i)==1

z(i,:) = [1 0 0];

else

z(i,:) = [0 0 1];

end

end

%Randomly divide the dataset into training (70%) and testing (30%) set

p = randperm(length(Y));

trainInput = X(p(1:(0.7\*size(X,1))),:);

trainOutput = z(p(1:(0.7\*size(X,1))),:);

testInput = X(p(0.7\*size(X,1)+1:end),:);

testOutput = z(p(0.7\*size(X,1)+1:end),:);

k = 1000; %Hidden neurons

[ind,c] = kmeans(trainInput,k); %kmeans clustering centres and indices

n = zeros(k,1); %number of inputs belonging to each cluster

for i=1:k

n(i) = sum(ind(:)==i);

end

sigma = zeros(k,1); %standard deviation

for i=1:k

sigma(i) = norm(trainInput(ind(:)==i,:)-c(i))/n(i);

end

beta = 0.5\*sigma.^-2;

%Hidden layer matrix evaluation

for i=1:length(trainOutput)

for j=1:size(c,1)

H(i,j) = exp(-beta(j)\*norm(trainInput(i,:)-c(j,:))^2);

end

end

W = pinv(H)\*trainOutput; %Weight evaluation

%Test data evaluation

for i=1:length(testOutput)

for j=1:size(c,1)

Ht(i,j) = exp(-beta(j)\*norm(testInput(i,:)-c(j,:))^2);

end

end

yp = Ht\*W; %Output evaluation

%Class determination

[~,pb]=max(testOutput,[],2);

[~,pa]=max(yp,[],2);

[cm, ~] = confusionmat(pa,pb); %calculating confusion matrix

IA = zeros(1,3);

OA = 0;

for i = 1:2

IA(i) = cm(i,i)/sum(cm(i,:)); %individual accuracy

OA = OA + cm(i,i);

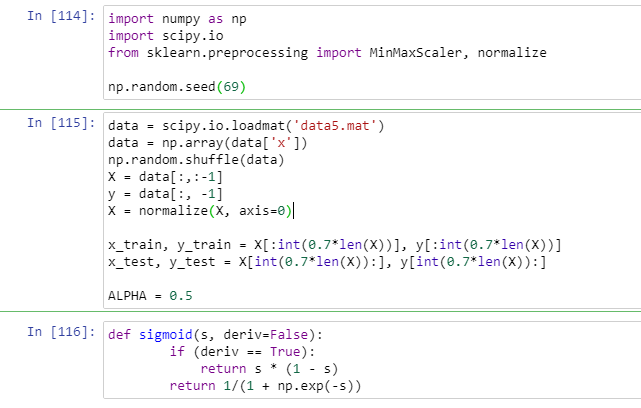
end

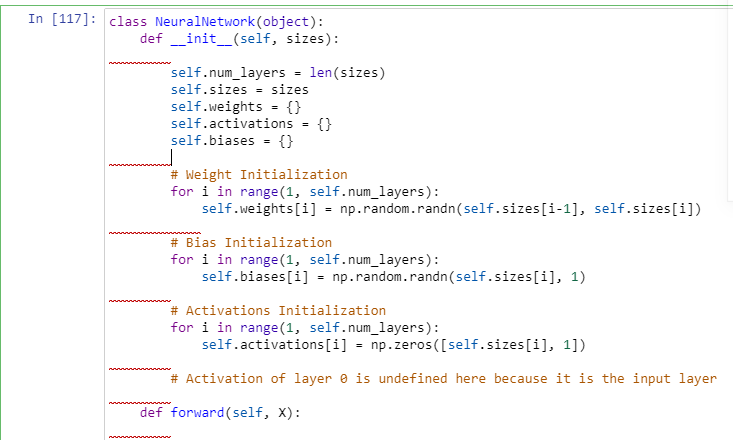
OA = OA/sum(cm(:)) %overall accuracy

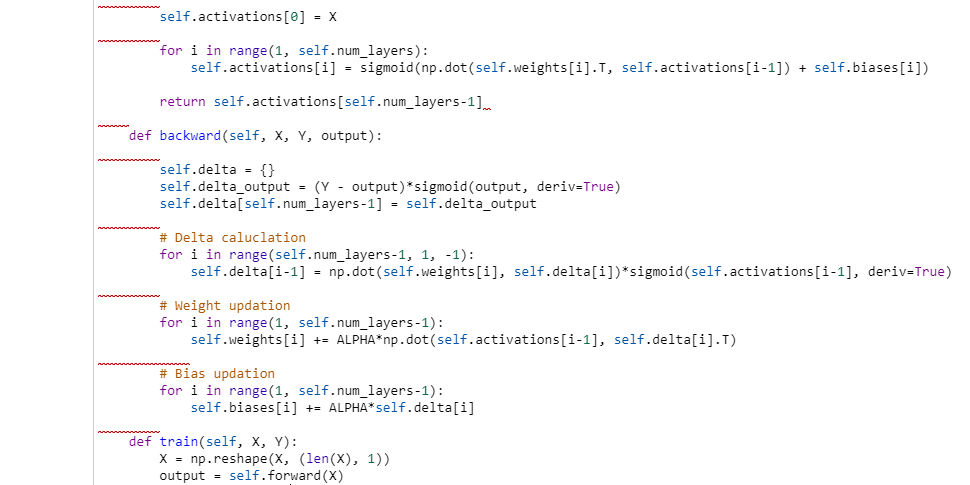
IA= [0.9511,0.9748,0]

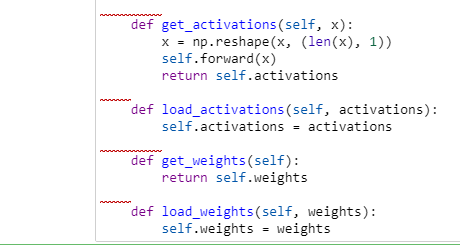
OA=0.9627

1. AutoEncoders









1. ELM

X = load('D:\CMS\Sem3-1\Neural Networkand Fuzzy Logic\data5.mat');

X=X.x;

X(:,1:72) = (X(:,1:72)-mean(X(:,1:72)))./std(X(:,1:72));

p = randperm(size(X,1));

X1 = ones(2148,73);

for i=1:size(X,1)

X1(i,:) = X(p(i),:);

if(X1(i,73)==0)

X1(i,73)=-1;

end

end

IA = [0 0];

W = zeros(10000,500); %number of nodes in hidden layer

Xtrain = X1(1:2000,1:72);

Ytrain = X1(1:2000,73);

Xtest = X1(2001:2148,1:72);

Ytest = X1(2001:2148,73);

randommat = randn(73,1000);

Xtrain = [ones(size(Xtrain,1),1) Xtrain];

G = Xtrain\*randommat;

H = tanh(G);

W = pinv(H)\*Ytrain;

count=0;

for j=1:148

b(j) = testELM(Xtest(j,:),randommat,W);

if b(j)>0

b(j)=1;

IA(1)=IA(1)+(Ytest(j)==1);

else

b(j)=-1;

IA(2) = IA(2)+(Ytest(j)==-1);

end

if(b(j)==Ytest(j))

count=count+1;

end

end

IA(1) = IA(1)/sum(Ytest(:)==1);

IA(2) = IA(2)/sum(Ytest(:)==-1);

Accuracy = count/148;

%function%

function y = testELM(features,randomperm,w)

a = [1 features];

g = a\*randomperm;

h = cos(g);

y = h\*w;

end

IA= [0.9526,0.9694]

OA(Accuracy) =0.9608

1. ELM Autoencoders

