CS6910 – Fundamentals of Deep Learning

Code Assignment - 1

TEAM 6

Lavanya L (EE18D414)

Chigullapally Sriharsha (CS17D012)

Sivaraman S (OE19S012)

1 Function Approximation

2 2D Non-Linear Data

3 Image Classification

Note: The codes are written in Matlab Environment

1. Function Approximation:

SNo	File name	Description
1	train.m	To load data and initialize parameters to start the training process and get results of both training and validation data
2	gradientupdate.m	Backpropagation algorithm to update the parameters using generalized delta rule
3	predict.m	To predict the results given an input using the final trained weights
4	sigmoid.m	To calculate the logistic function of a variable
5	Sigmgrad.m	To calculate the gradient of logistic function
6	plotresult.m	To plot the results
7	Fnplot.m	To plot the desired vs approximate function

train.m

```
load data.mat %loading data
load initweights.mat % loading initial weights

% Assign number of nodes for each layer
nx = size(x_train,2); %number of input features
nh1 = nx*2; %number of hidden layer 1 nodes
nh2 = nx*1; %number of hidden layer 2 nodes
ny = size(y_train,2); %number of output nodes

eeta = 0.9; %learning rate parameter
alpha = 0.3; % momentum parameter
beta = 1; % Beta for activation function

% Normalizing Inputs and Outputs
y = (y_train + 100) / 200;
x = (x_train + 10) / 20;
y1 = (y_val  + 100) / 200;
x1 = (x_val + 10) / 20;

%Initialize weights randomly
epsilon = 1;
%wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% wh1 = rand(nh1,nx+1)*2*epsilon - epsilon;
% wh2 = rand(nh2,nh1+1)*2*epsilon - epsilon;
% wo = rand(ny,nh2+1)*2*epsilon - epsilon;
%save('initweights.mat','wh1','wh2','wo');

% Calling Gradient update function to train the model and to
calculate
% error and output
%wh1f, wh2f, wof - final weights of hidden layer 1, 2, and output
layer
%after training
%err, s - average error and output of training data
%errval, sval - average error and output of validation data
[wh1f,wh2f,wof,err,s,errval,sval] =
gradientupdate(x,y,wh1,wh2,wo,eeta,alpha,beta,x1,y1);
%save('finalweights.mat','wh1f','wh2f','wof');

%Unnormalising the output of training data and validation data
```

```
 s = s';
 s = s * 200;
 s = s - 100;

sval = sval';
sval = sval * 200;
sval = sval - 100;
```

gradientupdate.m

```
function [wh1,wh2,wo,err,s,errval,sval] =
gradientupdate(ip,op,wh1,wh2,wo,eeta,alpha,beta,x1,y1)
% the function updates weights using generalized update rule
% ip - input
% op - target output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

%Initialize delta_weights of previous example for generalized
update rule
delta_wo_p = zeros(size(wo));
delta_wh2_p = zeros(size(wh2));
delta_wh1_p = zeros(size(wh1));

% e = error
err = [];% err = average error after each epoch of training data
errval = [];% err = average error after each epoch of validation
data
errdiff = 1;
while (abs(errdiff) > 10^(-6) || err(end) > 3*10^(-4))
    e = 0;
    %pattern mode
    for n = 1:N
        % x = n-th example
        x = ip(n,:)'; % Transpose - to make it a column vector

        % t = target output of nth example
        t = op(n);

        %s1 = output of input layer
        s1 = [1;x]; % adding x_0 (bias)

        %ah1 = activation value of hidden layer 1
        ah1 = wh1*s1; % calculating ah1

        %sh1 = output of hidden layer 1
        sh1 = sigmoid(beta*ah1); % calculating sh1
        sh1 = [1;sh1]; % adding sh1_0 (bias)

        %ah2 = activation value of hidden layer 2
        ah2 = wh2*sh1; % calculating ah2
```

```matlab
        %sh2 = output of hidden layer 2
        sh2 = sigmoid(beta*ah2); % calculating sh2
        sh2 = [1;sh2]; % adding sh2_0 (bias)

        %ao = activation value of output layer
        ao = wo*sh2; % calculating ao

        %so = final output
        so = sigmoid(beta*ao); % calculating so

        % average error calculation
        e = e + (1/(2*N))*(t - so)'*(t - so);
        %%%%%%%%%%%%%%% BACK PROPOGATION %%%%%%%%%%%%%%%%%

        % delo = error at output layer
        delo = (t - so).*sigmgrad(beta*ao);
        delo = beta*delo;

        % delh2 = error at hidden layer 2
        wo_nobias = wo(:,2:end);
        delh2 = beta*((wo_nobias'*delo).*sigmgrad(beta*ah2));

        % delh1 = error at hidden layer 1
        wh2_nobias = wh2(:,2:end);
        delh1 = beta*((wh2_nobias'*delh2).*sigmgrad(beta*ah1));

        % delta_wo = delta updates of output layer weights
        delta_wo = eeta*(delo*sh2');

        % delta_wh2 = delta updates of hidden layer 2 weights
        delta_wh2 = eeta*(delh2*sh1');

        % delta_wh1 = delta updates of hidden layer 1 weights
        delta_wh1 = eeta*(delh1*s1');

        %%%%%%%%%%%%%%%%%% UPDATE WEIGHTS %%%%%%%%%%%%%%%%%%

        wo = wo + delta_wo + alpha*(delta_wo_p);
        wh2 = wh2 + delta_wh2 + alpha*(delta_wh2_p);
        wh1 = wh1 + delta_wh1 + alpha*(delta_wh1_p);


        % storing delta weights for next example
        delta_wo_p = delta_wo;
        delta_wh2_p = delta_wh2;
        delta_wh1_p = delta_wh1;

    end
    [err1,sval] = predict(x1,y1,wh1,wh2,wo,beta);
    [e1,s] = predict(ip,op,wh1,wh2,wo,beta);
    if (size(err)>0)
        errdiff = e1 - err(end);
    end
    err = [err;e1];
    errval = [errval;err1];
end
```

```
end
```

Predict.m

```
function [e,s] = predict(ip,op,wh1,wh2,wo,beta)
% the function updates weights using generalized update rule
% ip - input
% op - target output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

% e = average error
e = 0;
    %pattern mode
    for n = 1:N
        % x = n-th example
        x = ip(n,:)'; % Transpose - to make it a column vector

        % t = target output of nth example
        t = op(n);

        %s1 = output of input layer
        s1 = [1;x]; % adding x_0 (bias)

        %ah1 = activation value of hidden layer 1
        ah1 = wh1*s1; % calculating ah1

        %sh1 = output of hidden layer 1
        sh1 = sigmoid(beta*ah1); % calculating sh1
        sh1 = [1;sh1]; % adding sh1_0 (bias)

        %ah2 = activation value of hidden layer 2
        ah2 = wh2*sh1; % calculating ah2

        %sh2 = output of hidden layer 2
        sh2 = sigmoid(beta*ah2); % calculating sh2
        sh2 = [1;sh2]; % adding sh2_0 (bias)

        %ao = activation value of output layer
        ao = wo*sh2; % calculating ao

        %so = final output
        so = sigmoid(beta*ao); % calculating so
        s(n) = so;

        % average error calculation
        e = e + (1/(2*N))*(t - so)'*(t - so);
end
end
```

Sigmoid.m

```
function [s] = sigmoid(a)
% The function calculates sigmoid of a given matrix/vector/scalar

s = 1 ./ (1 + exp(-a));
end
```

Sigmgrad.m

```
function [g] = sigmgrad(a)
% The function calculates gradient of a sigmoid function of a
given matrix/vector/scalar

s = 1 ./ (1 + exp(-a));
g = s.*(1 - s);

end
```

Plotresult.m

```
subplot(2,2,1);
plot(err,'b');
 xlabel('Number of Epochs');
 ylabel('Average Error');
 title('Training Data');

 subplot(2,2,3);
 scatter(y_train,s,25,'filled');
 xlabel('Desired Output');
 ylabel('Model Output');
 title('Model Output vs Desired Output of Training Data');

 subplot(2,2,4);
 scatter(y_val,sval,25,'r');
 xlabel('Desired Output');
 ylabel('Model Output');
 title('Model Output vs Desired Output of Validation Data');

 subplot(2,2,2);
plot(errval,'r');
 xlabel('Number of Epochs');
 ylabel('Average Error');
 title('Validation Data');
```

Fnplot.m

```
load finalweights.mat
m = 1;
xfn = 0:0.05:1;
yfn = 0:0.05:1;
[xp,yp] = meshgrid (xfn,yfn);
for i = 1:21
    for j = 1:21
        [~,fn(i,j)] =
predict([xp(i,j),yp(i,j)],xp(i,j),wh1f,wh2f,wof,beta);
    end
end
```

```
fn = fn* 200;
fn = fn - 100;


surf(xp*20-10,yp*20-10,fn);
colormap('summer');
hold on;
plot3([x_train(:,1);x_val(:,1)],[x_train(:,2);x_val(:,2)],[y_train
;y_val],'.','color','b','MarkerSize',10);
legend('Aproximated Function','Desired function');
title('Approximated vs Desired Function');
grid on;
xlabel('x1');
ylabel('x2');
zlabel('y');
hold off;
```

2. 2D Nonlinear Data:

SNo	File name	Description
1	train.m	To load training data and initialize parameters to start the training process and get results of training data
2	gradientupdate.m	Back-propagation algorithm to update the parameters using generalized delta rule
3	calcerr.m	To predict the results and error of given input using the final trained weights
4	Val.m	To load validation data and initialize parameters to start the training process and get results of validation data
5	sigmoid.m	To calculate the logistic function of a variable
6	Sigmgrad.m	To calculate the gradient of logistic function
7	Sftmax.m	To calculate softmax function at output layer
8	Fnplot.m	To plot the outputs of hidden layers and output layer
9	Plot2_.m	To plot the decision region

Train.m

```
load data.mat %loading data
load initweights.mat

N = size(x_train,1); %number of examples
nx = size(x_train,2); %number of input features
nh1 = nx*3; %number of hidden layer 1 nodes
nh2 = nx*3; %number of hidden layer 2 nodes
ny = 3; %number of output nodes

% Assigning 1 to output node of corresponding class of training
data
y = zeros(N,3);
y(:,1) = (y_train == 0);
y(:,2) = (y_train == 1);
y(:,3) = (y_train == 2);

%Normalizing input data
x = x_train - min(x_train);
x = x ./ (max(x_train) - min(x_train));

eeta = 0.5; %learning rate parameter
alpha = 0.3; %momentum parameter
beta = 1; %beta for activation function

%  Initialize weights randomly
%   epsilon = 1;
  %wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
%   wh1 = rand(nh1,nx+1)*2*epsilon - epsilon;
%   wh2 = rand(nh2,nh1+1)*2*epsilon - epsilon;
%   wo = rand(ny,nh2+1)*2*epsilon - epsilon;
%  save ('initweights.mat','wh1','wh2','wo');


% Calling Gradient update function to train the model and to
calculate
% error and output
```

```
%wh1f, wh2f, wof - final weights of hidden layer 1, 2, and output
layer
%after training
%err, s - average error and output of training data
[wh1f,wh2f,wof,err,s] =
gradientupdate(x,y,wh1,wh2,wo,eeta,alpha,beta);
plot(log(err)); hold on;

%p - the predicted class of each example
[~, p] = max(s, [], 2);
p = p-1;

%conf - confusion matrix
conf = confusionmat(y_train,p);
save ('finalweights.mat','wh1f','wh2f','wof');
```

Gradientupdate.m

```
function [wh1,wh2,wo,err,s] =
gradientupdate(ip,op,wh1,wh2,wo,eeta,alpha,beta)
% the function updates weights using generalized update rule
% ip - input
% op - output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

%Initialize delta_weights of previous example for generalized
update rule
delta_wo_p = zeros(size(wo));
delta_wh2_p = zeros(size(wh2));
delta_wh1_p = zeros(size(wh1));

errdiff = 1;
err = []; % average error after each epoch
while ((abs(errdiff) > 10^(-6)) || (err(end) > 10^(-4)))
    e = 0;
%pattern mode
for n = 1:N
    % x = n-th example
    x = ip(n,:)'; % Transpose - to make it a column vector

    % t = target output of nth example
    t = op(n,:)';

    %s1 = output of input layer
    s1 = [1;x]; % adding x_0 (bias)

    %ah1 = activation value of hidden layer 1
    ah1 = wh1*s1; % calculating ah1

    %sh1 = output of hidden layer 1
    sh1 = sigmoid(beta*ah1); % calculating sh1
```

```
    sh1 = [1;sh1]; % adding sh1_0 (bias)

    %ah2 = activation value of hidden layer 2
    ah2 = wh2*sh1; % calculating ah2

    %sh2 = output of hidden layer 2
    sh2 = sigmoid(beta*ah2); % calculating sh2
    sh2 = [1;sh2]; % adding sh2_0 (bias)

    %ao = activation value of output layer
    ao = wo*sh2; % calculating ao

    %so = final output
    so = sftmax(ao); % softmax neuron
    s(n,:) = so';

    [~,l] = max(t);
    e = e - (log(so(l))/N);
   %%%%%%%%%%%%%%% BACK PROPOGATION %%%%%%%%%%%%%%%%
    % delo = error at output layer
    delo = -so;
    delo(l) = delo(l) + 1;

    % delh2 = error at hidden layer 2
    wo_nobias = wo(:,2:end);
    delh2 = beta*(wo_nobias'*delo).*sigmgrad(beta*ah2);

    % delh1 = error at hidden layer 1
    wh2_nobias = wh2(:,2:end);
    delh1 = beta*(wh2_nobias'*delh2).*sigmgrad(beta*ah1);

    % delta_wo = delta updates of output layer weights
    delta_wo = eeta*(delo*sh2');

    % delta_wh2 = delta updates of hidden layer 2 weights
    delta_wh2 = eeta*(delh2*sh1');

    % delta_wh1 = delta updates of hidden layer 1 weights
    delta_wh1 = eeta*(delh1*s1');

    %%%%%%%%%%%%%%%%%% UPDATE WEIGHTS %%%%%%%%%%%%%%%%%%
    wo = wo + delta_wo + alpha*(delta_wo_p);
    wh2 = wh2 + delta_wh2 + alpha*(delta_wh2_p);
    wh1 = wh1 + delta_wh1 + alpha*(delta_wh1_p);

    % storing delta weights for next example
    delta_wo_p = delta_wo;
    delta_wh2_p = delta_wh2;
    delta_wh1_p = delta_wh1;
end
if (size(err)>0)
errdiff = e - err(end);
end
 err = [err;e];
end
end
```

Calcerror.m

```
function [e,s] = calcerr(ip,op,wh1,wh2,wo,beta)
% the function updates weights using generalized update rule
% ip - input
% op - output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

%Initialize delta_weights of previous example for generalized
update rule
e = 0;
%pattern mode
for n = 1:N
    % x = n-th example
    x = ip(n,:)'; % Transpose - to make it a column vector

    % t = target output of nth example
    t = op(n,:)';

    %s1 = output of input layer
    s1 = [1;x]; % adding x_0 (bias)

    %ah1 = activation value of hidden layer 1
    ah1 = wh1*s1; % calculating ah1

    %sh1 = output of hidden layer 1
    sh1 = sigmoid(beta*ah1); % calculating sh1
    sh1 = [1;sh1]; % adding sh1_0 (bias)

    %ah2 = activation value of hidden layer 2
    ah2 = wh2*sh1; % calculating ah2

    %sh2 = output of hidden layer 2
    sh2 = sigmoid(beta*ah2); % calculating sh2
    sh2 = [1;sh2]; % adding sh2_0 (bias)

    %ao = activation value of output layer
    ao = wo*sh2; % calculating ao

    %so = final output
    so = sftmax(ao); % softmax neuron
    s(n,:) = so';

    [~,l] = max(t);
    e = e - (log(so(l))/N);
end
end
```

Val.m

```
%load data.mat %loading data
load finalweights.mat
N1 = size(x_val,1); %number of examples in validation data
beta = 1;

% Assigning 1 to output node of corresponding class of validation
data
y1 = zeros(N1,3);
y1(:,1) = (y_val == 0);
y1(:,2) = (y_val == 1);
y1(:,3) = (y_val == 2);

%Normalizing input data
x1 = x_val - min(x_val);
x1 = x1 ./ (max(x_val) - min(x_val));

% Calling calcerr function to calculate error and output of
validation data
% errval, sval - average error and output of validation data
[errval,sval] = calcerr(x1,y1,wh1f,wh2f,wof,beta);

%pval - the predicted class of each example in validation data
[~, pval] = max(sval, [], 2);
pval = pval - 1;

%confval - confusion matrix
confval = confusionmat(y_val,pval);

%acc - accuracy of prediction
acc = sum(diag(confval))/sum(sum(confval));
```

Sigmoid.m

```
function [s] = sigmoid(a)
% The function calculates sigmoid of a given matrix/vector/scalar

s = 1 ./ (1 + exp(-a));
end
```

Sigmgrad.m

```
function [g] = sigmgrad(a)
% The function calculates gradient of a sigmoid function of a
given matrix/vector/scalar

s = 1 ./ (1 + exp(-a));
g = s.*(1 - s);
end
```

Sftmax.m

```
function [s] = sftmax(a)
s = exp (a);
s = s / sum(s);
end
```

Fnplot.m

```
% generate data to plot output of hidden layers and output layer
xfn = 0:0.05:1;
yfn = 0:0.05:1;
[xp,yp] = meshgrid (xfn,yfn);
for i = 1:21
for j = 1:21
[h11(i,j),h12(i,j),h13(i,j),h14(i,j),h15(i,j),h16(i,j),h21(i,j),h2
2(i,j),h23(i,j),h24(i,j),h25(i,j),h26(i,j),y11(i,j),y21(i,j),y31(i
,j)] = nodesval([xp(i,j),yp(i,j)],wh1f,wh2f,wof);
end
end
% nodesval.m - same as calcerr.m except that it returns back the
output of hidden layers and output layer

figure();
suptitle('Outputs of Hidden layer 1 - After Training');
subplot(3,2,1);
surf(xp,yp,h11);
xlim([-1 2]);
ylim([-1 2]);
xlabel('x1');
ylabel('x2');
zlabel('y');

title('Node 1');
subplot(3,2,2);
surf(xp,yp,h12);
xlim([-1 2]);
ylim([-1 2]);
title('Node 2');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,3);
surf(xp,yp,h13);
xlim([-1 2]);
ylim([-1 2]);
title('Node 3');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,4);
surf(xp,yp,h14);
xlim([-1 2]);
ylim([-1 2]);
title('Node 4');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,5);
surf(xp,yp,h15);
xlim([-1 2]);
```

```
ylim([-1 2]);
title('Node 5');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,6);
surf(xp,yp,h16);
xlim([-1 2]);
ylim([-1 2]);
% colormap(jet);
title('Node 6');
xlabel('x1');
ylabel('x2');
zlabel('y');

figure();
suptitle('Outputs of Hidden layer 2 - After Training');
subplot(3,2,1);
surf(xp,yp,h21);
xlim([-1 2]);
ylim([-1 2]);
title('Node 1');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,2);
surf(xp,yp,h22);
xlim([-1 2]);
ylim([-1 2]);
title('Node 2');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,3);
surf(xp,yp,h23);
xlim([-1 2]);
ylim([-1 2]);
title('Node 3');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,4);
surf(xp,yp,h24);
xlim([-1 2]);
ylim([-1 2]);
title('Node 4');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,5);
surf(xp,yp,h25);
```

```
xlim([-1 2]);
ylim([-1 2]);
title('Node 5');
xlabel('x1');
ylabel('x2');
zlabel('y');

subplot(3,2,6);
surf(xp,yp,h26);
xlim([-1 2]);
ylim([-1 2]);
colormap(jet);
title('Node 6');
xlabel('x1');
ylabel('x2');
zlabel('y');

figure();
surf(xp,yp,y11);
xlim([-1 2]);
ylim([-1 2]);
title('Node 1 Output Layer after After Training');
colormap(jet);
xlabel('x1');
ylabel('x2');
zlabel('y');

figure();
surf(xp,yp,y21);
xlim([-1 2]);
ylim([-1 2]);
title('Node 2 Output Layer after After Training');
colormap(jet);
xlabel('x1');
ylabel('x2');
zlabel('y');

figure();
surf(xp,yp,y31);
xlim([-1 2]);
ylim([-1 2]);
title('Node 3 Output Layer after After Training');
colormap(jet);
xlabel('x1');
ylabel('x2');
zlabel('y');
```

Plot2_.m

```
xfn = 0:0.001:1;
yfn = 0:0.001:1;
[xp,yp] = meshgrid (xfn,yfn);
for i = 1:1001
for j = 1:1001
```

```
[~,~,~,~,~,~,~,~,~,~,~,~,y11(i,j),y21(i,j),y31(i,j)] =
nodesval([xp(i,j),yp(i,j)],wh1f,wh2f,wof);
% nodesval.m - same as calcerr.m except that it returns back the
output of hidden layers and output layer

[~, p1(i,j)] =  max([y11(i,j) y21(i,j) y31(i,j)]);
end
end
p1=p1-1;
% p1 - predicted classes

xp = xp(:);% vectorize
yp = yp(:);
p1 = p1(:);
pointsize = 100;
scatter(xp, yp, 100, p1,'.');
hold on;
for i = 1:size(y_train,1)
if y_train(i) == 0
scatter(x(i,1),x(i,2),10,'+','w'); hold on;
end
if y_train(i) == 1
scatter(x(i,1),x(i,2),10,'+','k'); hold on;
end
if y_train(i) == 2
scatter(x(i,1),x(i,2),10,'+','r'); hold on;
end
end
```

3. Image Classification

SNo	File name	Description
1	train.m	To load data and initialize parameters to start the training process and get results of both training and validation data
2	gradientupdate.m	Back-propagation algorithm to update the parameters using generalized delta rule
3	Adamgradientupdate.m	Back-propagation algorithm to update the parameters using ADAM method
4	calcerr.m	To predict the results and error of given input using the final trained weights
5	Relu.m	To calculate relu activation function
6	relugrad.m	To calculate the gradient of logistic function
7	Sftmax.m	To calculate softmax function at output layer

Train.m

```
load data1.mat %loading data
% load initweights.mat %loading initial weights

pv = pca(x_train);% PCA of training data
%pv - projection matrix from PCA
pv = pv(:,1:360);% Extracting the prominent principal components
finalx = (x_train - mean(x_train))*pv;%projecting the training
data on the prominent principal components to get reduced
dimension

N = size(finalx,1); %number of examples
nx = size(finalx,2); %number of input features
nh1 = nx/2; %number of hidden layer 1 nodes
nh2 = nx/4; %number of hidden layer 2 nodes
ny = 5;%number of output nodes

%assigning values to output nodes
y1 = zeros(N,5);
y1(:,1) = (y_train == 1);
y1(:,2) = (y_train == 2);
y1(:,3) = (y_train == 3);
y1(:,4) = (y_train == 4);
y1(:,5) = (y_train == 5);

eeta = 0.00001; %learning rate parameter
alpha = 0.9; %momentum parameter
beta = 1; %beta for activation function
lambda =3; %regularisation parameter
r1 = 0.001; %r1 - rho1 for ADAM method
r2 = 0.001; %r2 - rho2 for ADAM method

% Initialize weights randomly
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
   wh1 = initialweights(nh1,nx+1);
   wh2 = initialweights(nh2,nh1+1);
   wo = initialweights(ny,nh2+1);

%finding reduced dimension of validation data
```

```
finalxval = (x_val-mean(x_val))*pv;

Nval = size(finalxval,1); %number of examples in validation data

%assigning values to output nodes of validation data
y1val = zeros(Nval,5);
y1val(:,1) = (y_val == 1);
y1val(:,2) = (y_val == 2);
y1val(:,3) = (y_val == 3);
y1val(:,4) = (y_val == 4);
y1val(:,5) = (y_val == 5);

% Calling Gradient update function to train the model and to
calculate
% error and output
%wh1f, wh2f, wof - final weights of hidden layer 1, 2, and output
layer
%after training
%err, s - average error and output of training data
%errval, sval - average error and output of validation data
% [wh1f,wh2f,wof,err,s,errval] =
adamgradientupdate(finalx,y1,wh1,wh2,wo,eeta,r1,r2,lambda,finalxva
l,y1val);
[wh1f,wh2f,wof,err,s,errval,sval] =
gradientupdate(finalx,y1,wh1,wh2,wo,eeta,alpha,beta,lambda,finalxv
al,y1val);

%p , pval - the predicted class of each example for training data
and
%validation data
[~, p] = max(s, [], 2);
[~, pval] = max(sval, [], 2);

%conf - confusion matrix of training data
conf = confusionmat(y_train,p);
%confval - confusion matrix of validation data
confval = confusionmat(y_val,pval);

%acc, accval - accuracy of prediction of training data and
validation data
acc = sum(diag(conf))/sum(sum(conf))
accval = sum(diag(confval))/sum(sum(confval));

er1 = [0;err(1:end-1)];
cerr = abs(err - er1);%error differnce

plot(err); hold on;
plot(errval); hold off;

% save ('finalweights.mat','wh1f','wh2f','wof');
```

Gradientupdate.m

```
function [wh1,wh2,wo,err,s,errval,sval] =
gradientupdate(ip,op,wh1,wh2,wo,eeta,alpha,beta,lambda,finalxval,y
1)
```

```
% the function updates weights using generalized update rule
% ip - input
% op - output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

%Initialize delta_weights of previous example for generalized
update rule
delta_wo_p = zeros(size(wo));
delta_wh2_p = zeros(size(wh2));
delta_wh1_p = zeros(size(wh1));

errval = [];% average error for each epoch of validation data
errdiff = 1; % error difference between every epoch
err = []; % average error for each epoch of training  data
while ((abs(errdiff) > 3*10^(-3)) || (err(end) > 0.8))
    e = 0;
%pattern mode
for n = 1:N
    % x = n-th example
    x = ip(n,:)'; % Transpose - to make it a column vector

    % t = target output of nth example
    t = op(n,:)';

    %s1 = output of input layer
    s1 = [1;x]; % adding x_0 (bias)

    %ah1 = activation value of hidden layer 1
    ah1 = wh1*s1; % calculating ah1

    %sh1 = output of hidden layer 1
    sh1 = relu(beta*ah1); % calculating sh1
    sh1 = [1;sh1]; % adding sh1_0 (bias)

    %ah2 = activation value of hidden layer 2
    ah2 = wh2*sh1; % calculating ah2

    %sh2 = output of hidden layer 2
    sh2 = relu(beta*ah2); % calculating sh2
    sh2 = [1;sh2]; % adding sh2_0 (bias)

    %ao = activation value of output layer
    ao = wo*sh2; % calculating ao

    %so = final output
    so = sftmax(ao); % softmax neuron
    s(n,:) = so';

    % calculate error
    [~,l] = max(t);
    e = e - (log(so(l))/N);
```

```
    %%%%%%%%%%%%%%% BACK PROPOGATION %%%%%%%%%%%%%%%%

    % delo = error at output layer
    delo = -so;
    delo(l) = delo(l) + 1;

    % delh2 = error at hidden layer 2
    wo_nobias = wo(:,2:end);
    delh2 = beta*(wo_nobias'*delo).*relugrad(beta*ah2);

    % delh1 = error at hidden layer 1
    wh2_nobias = wh2(:,2:end);
    delh1 = beta*(wh2_nobias'*delh2).*relugrad(beta*ah1);

    % delta_wo = delta updates of output layer weights
    delta_wo = eeta*(delo*sh2'-lambda*wo);

    % delta_wh2 = delta updates of hidden layer 2 weights
    delta_wh2 = eeta*(delh2*sh1'-lambda*wh2);

    % delta_wh1 = delta updates of hidden layer 1 weights
    delta_wh1 = eeta*(delh1*s1'-lambda*wh1);

    %%%%%%%%%%%%%%%%%% UPDATE WEIGHTS %%%%%%%%%%%%%%%%%
    wo = wo + delta_wo + alpha*(delta_wo_p);
    wh2 = wh2 + delta_wh2 + alpha*(delta_wh2_p);
    wh1 = wh1 + delta_wh1 + alpha*(delta_wh1_p);

    % storing delta weights for next example
    delta_wo_p = delta_wo;
    delta_wh2_p = delta_wh2;
    delta_wh1_p = delta_wh1;

    end
    if (size(err)>0)
    errdiff = e - err(end);
    end
    err = [err;e];
   [err1,sval] = calcerr(finalxval,y1,wh1,wh2,wo,beta);
    errval = [errval;err1];
end
end
```

Adamgradientupdate.m

```
function [wh1,wh2,wo,err,s,errval,sval] =
adamgradientupdate(ip,op,wh1,wh2,wo,eeta,r1,r2,lambda,finalxval,y1
)
% the function updates weights using adam method
% ip - input
% op - output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum
beta =1;
%N = Number of examples
```

```
N = size(ip,1); % N = Number of rows in ip

%Initialize r and q weights
rwo = zeros(size(wo));
rwh2 = zeros(size(wh2));
rwh1 = zeros(size(wh1));

qwo = zeros(size(wo));
qwh2 = zeros(size(wh2));
qwh1 = zeros(size(wh1));

errval = []; % error difference between every epoch for validation
data
errdiff = 1; % error difference between every epoch for training
data
err = []; % average error for each epoch
i=1;
while ((abs(errdiff) > 10^(-3)) || (err(end) > 0.8))
    e = 0;
%pattern mode
for n = 1:N
    % x = n-th example
    x = ip(n,:)'; % Transpose - to make it a column vector

    % t = target output of nth example
    t = op(n,:)';

    %s1 = output of input layer
    s1 = [1;x]; % adding x_0 (bias)

    %ah1 = activation value of hidden layer 1
    ah1 = wh1*s1; % calculating ah1

    %sh1 = output of hidden layer 1
    sh1 = relu(beta*ah1); % calculating sh1
    sh1 = [1;sh1]; % adding sh1_0 (bias)

    %ah2 = activation value of hidden layer 2
    ah2 = wh2*sh1; % calculating ah2

    %sh2 = output of hidden layer 2
    sh2 = relu(beta*ah2); % calculating sh2
    sh2 = [1;sh2]; % adding sh2_0 (bias)

    %ao = activation value of output layer
    ao = wo*sh2; % calculating ao

    %so = final output
    so = sftmax(ao); % softmax neuron
    s(n,:) = so';

    % calculate error
    [~,l] = max(t);
    e = e - (log(so(l))/N);
    %%%%%%%%%%%%%%% BACK PROPOGATION %%%%%%%%%%%%%%%%%
```

```
% delo = error at output layer
delo = -so;
delo(l) = delo(l) + 1;

% delh2 = error at hidden layer 2
wo_nobias = wo(:,2:end);
delh2 = beta*(wo_nobias'*delo).*relugrad(beta*ah2);

% delh1 = error at hidden layer 1
wh2_nobias = wh2(:,2:end);
delh1 = beta*(wh2_nobias'*delh2).*relugrad(beta*ah1);

% gwo = gradient of output layer weights
gwo = -(delo*sh2'-lambda*wo);

% gwh2 = gradient of hidden layer 2 weights
gwh2 = -(delh2*sh1'-lambda*wh2);

% gwh1 = gradient of hidden layer 1 weights
gwh1 = -(delh1*s1'-lambda*wh1);

%%%%%%%%%%%%%%%%%% UPDATE WEIGHTS %%%%%%%%%%%%%%%%%%

qwo = r1*qwo + (1-r1)*gwo;
qwh2 = r1*qwh2 + (1-r1)*gwh2;
qwh1 = r1*qwh1 + (1-r1)*gwh1;

rwo = r2*rwo + (1-r2)*(gwo.^2);
rwh2 = r2*rwh2 + (1-r2)*(gwh2.^2);
rwh1 = r2*rwh1 + (1-r2)*(gwh1.^2);

q_wo = qwo / (1 - r1^i);
q_wh1 = qwh1 / (1 - r1^i);
q_wh2 = qwh2 / (1 - r1^i);

r_wo = rwo / (1 - r2^i);
r_wh1 = rwh1 / (1 - r2^i);
r_wh2 = rwh2 / (1 - r2^i);

eps = 10^(-8);
delta_wo = -eeta*(q_wo ./ (eps + r_wo.^0.5));
delta_wh2 = -eeta*(q_wh2 ./ (eps + r_wh2.^0.5));
delta_wh1 = -eeta*(q_wh1 ./ (eps + r_wh1.^0.5));

wo = wo + delta_wo;
wh2 = wh2 + delta_wh2;
wh1 = wh1 + delta_wh1;

end
if (size(err)>0)
errdiff = e - err(end);
end
err = [err;e];
i = i+1;
[err1,sval] = calcerr(finalxval,y1,wh1,wh2,wo,beta);
errval = [errval;err1];
```

```
end
end
```

Calcerr.m

```
function [e,s] = calcerr(ip,op,wh1f,wh2f,wof,beta)
% the function updates weights using generalized update rule
% ip - input
% op - output
% wh1, wh2, wo - weights of hidden layer 1, 2, and output layer
% eeta - learning rate
% alpha - momentum

%N = Number of examples
N = size(ip,1); % N = Number of rows in ip

e = 0;
%pattern mode
for n = 1:N
    % x = n-th example
    x = ip(n,:)'; % Transpose - to make it a column vector

    % t = target output of nth example
    t = op(n,:)';

    %s1 = output of input layer
    s1 = [1;x]; % adding x_0 (bias)

    %ah1 = activation value of hidden layer 1
    ah1 = wh1f*s1; % calculating ah1

    %sh1 = output of hidden layer 1
    sh1 = relu(beta*ah1); % calculating sh1
    sh1 = [1;sh1]; % adding sh1_0 (bias)

    %ah2 = activation value of hidden layer 2
    ah2 = wh2f*sh1; % calculating ah2

    %sh2 = output of hidden layer 2
    sh2 = relu(beta*ah2); % calculating sh2
    sh2 = [1;sh2]; % adding sh2_0 (bias)

    %ao = activation value of output layer
    ao = wof*sh2; % calculating ao

    %so = final output
    so = sftmax(ao); % softmax neuron
    s(n,:) = so';
    %%%%%%%%%%%%%%% BACK PROPOGATION %%%%%%%%%%%%%%%%%

    [~,l] = max(t);
    e = e - (log(so(l))/N);

end
end
```

Relu.m

```
function [s] = relu(a)
    s = a.*(a > 0);
end
```

Relugrad.m

```
function [g] = relugrad(a)
g = (a >= 0);
end
```

Sftmax.m

```
function [s] = sftmax(a)
s = exp (a);
s = s / sum(s);
end
```