Problem# 01: Calculate and plot the different activation functionsused in neural networks.

Solution:

1. Plot the Logistic function using: tmp = exp(-x);y1 = 1./(1+tmp);

2. Plot the hyperbolic tangent function: y2 = (1-tmp)./(1+tmp);

3. Identity function;

4. Plot the sigmoid model using this formula:$\varphi(v) = \frac{1}{1+\exp(-av)}$;

5. Plot the Threshold function using the formula:$\varphi(v) = \begin{cases} 1 \; if \; v \geq 0 \\ 0 \; if \; v < 0 \end{cases}$

#Source Code:

```
clc
clearall
close all;

x = -10:0.01:10;
tmp = exp(-x);
y1 = 1./(1+tmp);
y2 = (1-tmp)./(1+tmp);
y3 = x;
tm = exp(-10.*x);
y4=1./(1+tm);
[xx,xy]=find(x>=0);
x3=zeros(1,2000);
x3(xy)=1;
subplot(231); plot(x, y1); grid on;
axis([min(x) max(x) -2 2]);
title('Logistic Function');
xlabel('(a)');
axis('square');
subplot(232); plot(x, y2); grid on;
axis([min(x) max(x) -2 2]);
title('Hyperbolic Tangent Function');
xlabel('(b)');
axis('square');
subplot(233); plot(x, y3); grid on;
axis([min(x) max(x) min(x) max(x)]);
title('Identity Function');
xlabel('(c)');
axis('square');
subplot(234); plot(x, y4); grid on;
axis([min(x) max(x) min(x) max(x)]);
title('Sigmoid Function');
xlabel('(d)');
```

```
axis('square');
subplot(235); plot(x, x3); grid on;
axis([min(x) max(x) min(x) max(x)]);
title('Threshold Function');
xlabel('(d)');
axis('square');
```

Problem# 2: Calculate and analyze the forward pass of a neural network.

Solution:

1. Initialize weight W1 and W2 according to input and hidden layer neuron.

2. Make the input column vector and feed into the network input layer.

3. Then weight will multiply with input and passed through the activation function sigmoid.

4. Finally gives the output of feed forward network which was denoted by Y.

Source Code:

```
clc
clearall
closeall

X=[2,3];
W1=ones(2,2)
W2=ones(2,2)
X=transpose(X)
A1=W1*X
A = sigmf(A1,[1 0])
Y1=W2*A
Y = sigmf(W2*A,[1 0])
Y = transpose(Y)
```

Problem# 3: Write a MATLAB program to calculate weight MATRIX.

Source Code:

```
%Hetro associative neural net for mapping input vectors to output vectors
clc;
clear;
closeall;
x=[1 1 0 0;1 0 1 0;1 1 1 0;0 1 1 0]
t=[1 0;1 0;0 1;0 1]
w=zeros(4,2)
for i=1:4
    w=w+x(i,1:4)'*t(i,1:2)
end
disp('weight matrix');
disp(w);
```

Problem# 4: To learn a single layer neural network by Stochastic Gradient descent-based (SGD) Delta rule.

Solution:

1. The SGD method of the delta rule given by Equation

$$\delta_i = \varphi(v_i)(1-\varphi(v_i))e_i$$
$$\Delta w_{ij} = \alpha \delta_i x_j$$
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

2.Consider a neuralnetwork that consists of three input nodes and one output node It takes the weights and training data of the neural network and returns thenewly trained weights.

3. Take one of the data points and calculate theoutput, y. Calculate the difference between this output and the correct output, d.Calculate the weight update, dW, according to the delta rule. Using this weightupdate, adjust the weight of neural network. Repeat the process for the numberof the training data points, N. This way, the SGD trains the neuralnetwork for every epoch.

Source Code:

```
%Take sigmoid function matlab file inside the current directory
clc
closeall
clearall
%Input
X = [ 0 0 1;
      0 1 1;
      1 0 1;
      1 1 1;
    ];
%Target output
D = [ 0
      0
      1
      1
    ];
%Random weight generation
W = 2*rand(1, 3) - 1;

for epoch = 1:100           % train
alpha = 0.9;

  N = 4; %no of input node
for k = 1:N
    x = X(k, :)';
    d = D(k);
% Network Layer
```

```
    v = W*x;
    y = Sigmoid(v);

    e     = d - y;
delta = y*(1-y)*e;

dW = alpha*delta*x;   % delta rule

W(1) = W(1) + dW(1);
W(2) = W(2) + dW(2);
W(3) = W(3) + dW(3);
end

end
YY=[];
N = 4;   % inference
for k = 1:N
  x = X(k, :)';
  v = W*x;
  y = Sigmoid(v);
  YY=[YY y];
end
disp('The prediction Of this network are: ')
disp(YY>.90)
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Sigmoid Function to calculate the activation

```
function y = Sigmoid(x)
  y = 1 / (1 + exp(-x));
end
```

%%%

Problem# 5: To learn a single layer neural network by Batch method-based delta rule.

Solution:

1. Consider a neuralnetwork that consists of three input nodes and one output node It takes the weights and training data of the neural network and returns thenewly trained weights.

2. This code does not immediately train the neural network with the weight update, dW, of the individual training data points. It adds the individual weight updates of the entire training data to dWsum and adjusts the weight just once using the average, dWavg. This is the fundamental difference that separates this method from the SGD method. The averaging feature of the batch method allows the training to be less sensitive to the training data. The weight update formula of batch method is:

$$\Delta w_{ij} = \frac{1}{N}\sum_{k=1}^{N}\Delta w_{ij}(k)$$

3. All the training data is fed into the trainedneural network, and the output is displayed. Check the output and correctoutput from the training data to verify the adequacy of the training.

Source Code:

```
clc
closeall
clearall

X = [ 0 0 1;
      0 1 1;
      1 0 1;
      1 1 1;
    ];

D = [ 0
      0
      1
      1
    ];

W = 2*rand(1, 3) - 1;

for epoch = 1:500
alpha = 0.9;

dWsum = zeros(3, 1);

  N = 4;
for k = 1:N
    x = X(k, :)';
    d = D(k);

    v = W*x;
    y = Sigmoid(v);
%error calculation
    e     = d - y;
delta = y*(1-y)*e;

dW = alpha*delta*x;

dWsum = dWsum + dW;
end
dWavg = dWsum / N;
%weight update
W(1) = W(1) + dWavg(1);
W(2) = W(2) + dWavg(2);
W(3) = W(3) + dWavg(3);
```

```
end
YY=[];
N = 4;
for k = 1:N
  x = X(k, :)';
  v = W*x;
  y = Sigmoid(v);
  YY=[YY y];
end
disp('The prediction Of this network are: ')
disp(YY>.90)
```

Problem# 6: To compare the efficiency of different types of Delta learning rules for neural network.

Solution:

1. Learn network by SGD based Delta rule.

2. Learn network by Batch method-based delta rule.

3. This code compares the mean error of the two methods and investigate thelearning speeds of the SGD andthe batch.

4. In order to evaluate a fair comparison, the weights of both methods are initialized with the same values.

Source Code:

```
clearall

X = [ 0 0 1;
      0 1 1;
      1 0 1;
      1 1 1;
    ];

D = [ 0
      0
      1
      1
    ];


E1 = zeros(1000, 1);
E2 = zeros(1000, 1);

W1 = 2*rand(1, 3) - 1;
W2 = W1;

for epoch = 1:1000           % train
alpha = 0.9;

  N = 4;
```

```
for k = 1:N
    x = X(k, :)';
    d = D(k);

    v = W1*x;
    y = Sigmoid(v);

    e     = d - y;
delta = y*(1-y)*e;

dW = alpha*delta*x;   % delta rule

W1(1) = W1(1) + dW(1);
W1(2) = W1(2) + dW(2);
W1(3) = W1(3) + dW(3);
end
%SGD END

alpha = 0.9;

dWsum = zeros(3, 1);

  N = 4;
for k = 1:N
    x = X(k, :)';
    d = D(k);

    v = W2*x;
    y = Sigmoid(v);

    e     = d - y;
delta = y*(1-y)*e;

dW = alpha*delta*x;

dWsum = dWsum + dW;
end
dWavg = dWsum / N;

W2(1) = W2(1) + dWavg(1);
W2(2) = W2(2) + dWavg(2);
W2(3) = W2(3) + dWavg(3);


  es1 = 0;
  es2 = 0;
  N   = 4;
for k = 1:N
    x = X(k, :)';
    d = D(k);

v1  = W1*x;
y1  = Sigmoid(v1);
    es1 = es1 + (d - y1)^2;

v2  = W2*x;
y2  = Sigmoid(v2);
    es2 = es2 + (d - y2)^2;
```

```
end
E1(epoch) = es1 / N;
E2(epoch) = es2 / N;
end

plot(E1, 'r')
holdon
plot(E2, 'b:')
xlabel('Epoch')
ylabel('Average of Training error')
legend('SGD', 'Batch')
gridon
```

Problem# 7: Create a simple Back Propagation neural network thatsolve the XOR problem.

Solution:

1. Initialize the weights with adequate values.

2. Enter the input from the training data {input, correct output} and obtain the neural network's output.Calculate the error of the output to the correct output and the delta, δ, of the output nodes.

$$e = d - y$$
$$\delta = \varphi'(v)e$$

3. Propagate the output node delta, δ, backward, andcalculate the deltas of the immediate next (left) nodes.

$$e^{(k)} = W^T\delta$$
$$\delta^{(k)} = \varphi'\left(v^{(k)}\right)e^{(k)}$$

4. Repeat Step 3 until it reaches the hidden layer that is on the immediate right of the input layer.
5. Adjust the weights according to the following learning rule.

$$\Delta w_{ij} = \alpha\delta_i x_j$$
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

6. Repeat Steps 2-5 for every training data point.
7. Repeat Steps 2-6 until the neural network is properlytrained.

Source Code:

```
clc
closeall
clearall
randn('seed',3)
rand('seed',3)
X = [ 0 0;
      0 1;
      1 0;
      1 1;
    ];

D = [ 0
      1
      1
      0
    ];

W1 = -1 +2.*rand(4,2);
W2 = -1 +2.*rand(1,4);

for epoch = 1:2000           % train
alpha = 0.9;

  N = 4;
for k = 1:N
    x = X(k, :)';
    d = D(k);

    v1 = W1*x;
    y1 = 1 ./ (1 + exp(-v1));
v  = W2*y1;
    y = 1 ./ (1 + exp(-v));

    e     = d - y;
delta = y.*(1-y).*e;

    e1     = W2'*delta;
    delta1 = y1.*(1-y1).*e1;

    dW1 = alpha*delta1*x';
W1  = W1 + dW1;

    dW2 = alpha*delta*y1';
W2  = W2 + dW2;
end

end
yy=[]
N = 4;   % inference
```

```
for k = 1:N
x  = X(k, :)';
  v1 = W1*x;
  y1 = 1 ./ (1 + exp(-v1));;
v  = W2*y1;
y  = 1 ./ (1 + exp(-v));
yy=[yy y]
end
disp('The prediction Of this network are: ');
disp(yy>0.90);
```

Problem# 8: Classify multiclass problem using back propagation neural network.

Solution:

The training process of the multiclass classification neural network is summarized in these steps.

1. Construct the output nodes to have the same value asthe number of classes. The softmax function is used asthe activation function.
2. Switch the names of the classes into numeric vectors viathe one-hot encoding method.

$$\text{Class 1} \rightarrow [1\,0\,0]$$
$$\text{Class 2} \rightarrow [0\,1\,0]$$
$$\text{Class 3} \rightarrow [0\,0\,1]$$

3. Initialize the weights of the neural network withadequate values.
4. Enter the input from the training data {input, correctoutput }into the neural network and obtain the output.Calculate the error between the output and correctoutput and determine the delta, δ, of the output nodes.

$$e = d - y$$
$$\delta = e$$

5. Propagate the output delta backwards and calculate thedelta of the subsequent hidden nodes.

$$e^{(k)} = W^T \delta$$
$$\delta^{(k)} = \varphi'\left(v^{(k)}\right) e^{(k)}$$

6. Repeat Step 5 until it reaches the hidden layer on the immediate right of the input layer.
7. Adjust the weights of the neural network using this learning rule:

$$\Delta w_{ij} = \alpha \delta_i x_j$$
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

8. Repeat Steps 4-7 for all the training data points.
9. Repeat Steps 4-8 until the neural network has beentrained properly.

Source Code:

```
clc
closeall
clearall
randn('seed',3)
rand('seed', 3)
X  = zeros(5, 5, 5);
X(:, :, 1) = [ 0 1 1 0 0;
               0 0 1 0 0;
               0 0 1 0 0;
               0 0 1 0 0;
               0 1 1 1 0
             ];

X(:, :, 2) = [ 1 1 1 1 0;
               0 0 0 0 1;
               0 1 1 1 0;
               1 0 0 0 0;
               1 1 1 1 1
             ];

X(:, :, 3) = [ 1 1 1 1 0;
               0 0 0 0 1;
```

```
                0 1 1 1 0;
                0 0 0 0 1;
                1 1 1 1 0
              ];

X(:, :, 4) = [ 0 0 0 1 0;
                0 0 1 1 0;
                0 1 0 1 0;
                1 1 1 1 1;
                0 0 0 1 0
              ];

X(:, :, 5) = [ 1 1 1 1 1;
                1 0 0 0 0;
                1 1 1 1 0;
                0 0 0 0 1;
                1 1 1 1 0
              ];

D = [ 1 0 0 0 0;
      0 1 0 0 0;
      0 0 1 0 0;
      0 0 0 1 0;
      0 0 0 0 1
    ];

W1 = randn(50, 25);
W2 = randn( 5, 50);

for epoch = 1:1000          % train
alpha = 0.9;

  N = 5;
for k = 1:N
    x = reshape(X(:, :, k), 25, 1);
    d = D(k, :)';

    v1 = W1*x;
    y1 = 1 ./ (1 + exp(-v1));
v  = W2*y1;
y  = v ./ sum(v);
    e     = d - y;
delta = e;

    e1     = W2'*delta;
    delta1 = y1.*(1-y1).*e1;

    dW1 = alpha*delta1*x';
    W1 = W1 + dW1;

    dW2 = alpha*delta*y1';
    W2 = W2 + dW2;
end

end
yy=[];
N = 5;                          % inference
for k = 1:N
x  = reshape(X(:, :, k), 25, 1);
```

```
  v1 = W1*x;
  y1 = 1 ./ (1 + exp(-v1));
v  = W2*y1;
y  = v ./ sum(v);
yy=[yy y]
end
disp('The prediction Of this network are: ');
disp(yy>0.90)
```

Problem# 9: Create a Discreate Hopfield neural network.

Source Code:

```
%Discrete Hopfield net
clc;
clear;
x=[1 1 1 0];
tx=[0 0 1 0];
w=(2*x'-1)*(2*x-1);
fori=1:4
w(i,i)=0;
end
con=1;
y=[0 0 1 0];
while con
up=[4 2 1 3];
fori=1:4
        yin(up(i))=tx(up(i))+y*w(1:4,up(i));
if yin(up(i))>0
y(up(i))=1;
end
end
if y==x
disp('Convergence has been obtained');
disp('The Converged Ouput');
disp(y);
con=0;
end
end
```

Problem# 10: Create Ahebb network to classify two-dimensional input pattern.

Solution:

The two-dimensional input pattern is:

E=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1];
F=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1];
And the target is: t=[1 -1]

This network finds the pattern from E and F where network will get the pattern like t it will classify it to 2.

Example last three digit of this pattern is 1 and -1. Hebb network will gives this value to 2

Source Code:

```
clear;
clc;
%Input Patterns
E=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1];
F=[1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 -1];
x(1,1:20)=E;
x(2,1:20)=F;
w(1:20)=0;
t=[1 -1];
b=0;
fori=1:2
    w=w+x(i,1:20)*t(i);
    b=b+t(i);
end
disp('Weight matrix');
disp(w);
disp('Bias');
disp(b);
```