Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

Checklist:

Completed:

X Pseudocode

X Well-documented

X Code works

X Learning parameters:

inputs, targets, nodeLayers, learning rate,

epochs, batchSize

X Termination conditions: max epochs, correct classification

X Mini-batch SGD

X Outputs as requested (iris, MNIST, and XOR)

Pseudo Code:

function [weight, bias] = BackProp (inputs, targets, nodeLayers, numEpochs, batchSize, eta)
Step 1:
Calculate the size to determine the number of nodeLayers – L
Initialize a cell to hold weights, biases, batch values, target values
Initialize a batch counter variable to index batch values
Calculate the size to determine the total number of inputs - inputSize

Loop through each layer starting from the 2nd layer
    Initialize weight and biases with randomized normal distribution values i.e. with mean=0 and SD=1

Step 2:
Divide the input matrix into mini batches by looping through the first index of each batch - incrementing by batchSize (which is the step) for each iteration
    If the difference between inputSize and starting position is >= batchSize

Assign the batch values to the respective mini batch by setting the inputs and targets as initPos: initPos + batchSize - 1
Update the batch index value
Else
Assign the batch values to the last batch by setting inputs and target values to initPos:end

Step 3:
Loop through each epoch
Loop through each batch
Initialize a cell to hold values for the intermediate nodes and activation function values
Feedforward the network – For each layer from the 2nd layer calculate z{layer} and a{layer}
Initialize a cell to hold error values
Invoke the SigmoidPrime function, compute the cost and then calculate the output layer error i.e. delta

Backpropogate error through the network from L-1 to 2nd layer – step size is -1
For each layer, compute the value of delta{layer}

Gradient descent – Loop through each layer from L to 2nd layer and update the adjusted and optimal Weight and bias values
End

Step 4:
Compute the final output values after updating weights
Initialize a cell to hold the final output values
Assign inputs to the first element of the output cell
For each layer from the 2nd layer, compute output{layer} by calling logsig function

At the end of each epoch, compute MSE, the number of correct cases and accuracy.
Terminate the training based on accuracy if the value of correct is same as inputSize and report the output
Report for each iteration if the number of epochs are less than 100
If running for a large number of epochs, report only the 100th epoch
End
Report the output values for the last iteration.

Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

Instructions:

The project code is stored in the file – **BackProp.m** and the function name is called BackProp. There are 2 additional files for the Sigmoid and SigmoidPrime functions – **Sigmod.m** and **SigmoidPrime.m**

Here's the list of required input parameters:

- inputs: a matrix with a column for each example, and a row for each input feature
- targets: a matrix with a column for each example, and a row for each output feature
- nodeLayers: a vector with the number of nodes in each layer (including the input and output layers).
- numEpochs: (scalar) desired number of epochs to run
- batchSize: (scalar) number of instances in a mini-batch
- eta: (scalar) learning rate

The output of the network will be obtained by invoking the function:
[weight, bias] = BackProp(inputs, targets, nodeLayers, numEpochs, batchSize,eta)

Set all the input parameters and invoke the function as shown in the example below:

```
>> iris = csvread('iris.csv');
>> inputs = iris(:, 1:4).';
>> targets = iris(:, 5:7).';
>> BackProp(inputs, targets, [4, 20, 3], 100, 10, 0.1);
```

In the above function call –
nodeLayers : [4,20,3] i.e. 4 input nodes,20 hidden nodes and 3 output nodes.
numEpochs: 10
eta/learning rate: 0.1

Description:

Fully matrix-based approach to backpropagation algorithm over a mini-batch problem is implemented in this project using Matlab. The function BackProp implements the Neural network and learns the training examples, applies a gradient descent learning step and finally outputs the desired result.

As shown in the Pseudo code the network is implemented in 4 steps:

Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

Step 1 includes Initializations and variable declarations. Weights and biases are initialized using the randn function to assign randomized normally distributed values.
Step 2 includes dividing the input matrix into a number of mini batches as specified by batchSize.
Step 3 includes implementing the Backpropagation algorithm and Gradient descent over a mini batch –

- Feedforward the network and compute z{layer} that represents the intermediate node values and a{layer} i.e. the values for the activation function.
- Backpropagate the error through the network and adjust weight and biases through every iteration.
- Gradient descent to further compute and assign the updated/adjusted weights and bias values for each layer

Step 4 computes the final output values and reports as per the requirement to display the desired output. Mean Squared error(MSE), the number of correct cases and Accuracy i.e. correct/inputSize is calculated and reported.

Code:

```
%Brunda Chouthoy
%CSC578 - Project 1
%Implementing a Neural Network - Backpropogation algorithm

function [ weight, bias ] = BackProp(inputs, targets, nodeLayers, numEpochs, batchSize,
eta)
  L = size(nodeLayers,2); %total number of node layers
  weight = cell(1,L); %Initialize a cell to hold the weight
  bias = cell(1,L); %Initialize a cell to hold the biases
  batchValues = {}; %Initialize a cell to hold batch values
  targetValues = cell(1,L); %Initialize a cell to hold target values
  batchIndex = 1; %Initialize a counter variable
  inputSize = size(inputs,2); %total number of inputs

  %For each layer from the 2nd layer
  for layer = 2:L
     %Initialize weight and biases with randomized normal distribution values i.e mean=0
and SD=1
     weight {layer} = randn(nodeLayers(layer), nodeLayers(layer-1));
     bias{layer} = randn(nodeLayers(layer), 1);
  end

  %Dividing the input matrix into mini batches
  %Increment by the value batchSize(step) for each iteration
```

```
  for initPos = 1:batchSize:inputSize
    if inputSize - initPos >= batchSize
        miniBatch = inputs(:, initPos: initPos + batchSize - 1);
        batchValues{batchIndex} = miniBatch;
        target = targets(:, initPos: initPos + batchSize - 1);
        targetValues{batchIndex} = target;
        batchIndex = batchIndex + 1;
    else
        miniBatch = inputs(:, initPos:end);
        batchValues{batchIndex} = miniBatch;
        target = targets(:, initPos:end);
        targetValues{batchIndex} = target;
    end
  end

  %Loop through each epoch and batch
  for epoch = 1:numEpochs
      for batch = 1:size(batchValues,2)
          z = cell(1,L); %Initialize a cell to hold values for the intermediate nodes
          a = cell(1,L); %Initialize a cell to hold the activation function
          a{1} = batchValues{batch}; %Input value of the batch is assigned to the first element
of the activation cell
          correct = 0;

          %Feedforward the network - For each layer calculate z{layer} and a{layer}
          for layer = 2:L
              z{layer} = weight{layer} * a{layer - 1} + bias{layer};
              a{layer} = Sigmoid(z{layer});
          end

          % Calculate the output layer error i.e. delta
          delta = cell(1,L); %Initialize a cell to hold the error values
          % Invoke the SigmoidPrime function
                error = (a{L} - targetValues{batch});
          cost = error .* SigmoidPrime(z{L});
          delta{L} = cost;

          % Back propagate error through the network from L to 2nd layer
          % Step size is -1
          for layer = (L - 1) : -1 : 2
              delta{layer} = (weight{layer + 1}.' * delta{layer + 1}) .* SigmoidPrime(z{layer});
          end
```

Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

```
        %Gradient Descent and finding the minimum
        %For each layer from L to 2nd layer
         for layer = L : -1 : 2
            weight{layer} = weight{layer} - (eta/length(batchValues{batch})) * (delta{layer} *
a{layer - 1}.');
            bias{layer} = bias{layer} - (eta/length(batchValues{batch})) * sum(delta{layer}, 2);
         end
      end

      % Compute final output values after updating weights
      output = {}; %Initialize a cell to hold the final output values
      output{1} = inputs; %Assign inputs to first element of the output cell
      %For each layer from the 2nd layer
      for layer = 2 : L
         z1 = (weight{layer}*output{layer-1})+(bias{layer});
         output{layer} = logsig(z1); %hold the activation for next layer
      end
      error = output{L}-targets;
      %Computing the value of MSE .. Diving by 2n
      MSE = sqrt(sum(sum(error.^2)))/(2*inputSize);

      %Compute the number of correct cases
      correct = correct + sum(all(targets==round(output{L}),1),2);

      %Computing accuracy
      accuracy = correct/inputSize;

     %Reporting based on Accuracy
     %If all the input cases are correct - accuracy=1
      if correct == inputSize
          fprintf('Epoch %d, MSE: %f, Correct: %d/%d, Acc: %f \n', epoch, MSE, correct,
inputSize, accuracy);
        break
      end

      % Reporting for each iteration if the number of epochs are less than 100
      if numEpochs <= 100
          fprintf('Epoch %d, MSE: %f, Correct: %d / %d,  Acc: %f \n', epoch, MSE, correct,
inputSize, accuracy);
      elseif mod(epoch, 100) == 0 && numEpochs > 100
      % if running a large number of epochs, reporting only the 100th epoch
          fprintf('Epoch %d, MSE: %f, Correct: %d / %d,  Acc: %f \n', epoch, MSE, correct,
inputSize, accuracy);
```

```
      end
  end

  if mod(numEpochs, 100) ~= 0 && numEpochs > 100
  % Reporting for the last iteration
     fprintf('Epoch %d, MSE: %f, Correct: %d / %d,  Acc: %f \n', epoch, MSE, correct,
inputSize, accuracy);
  end
end
```

Analysis:

The implemented network in this project works properly for all the datasets with different configurations i.e. with changes in the number of hidden layers, number of epochs, learning rates and batch sizes as specified in the problem statement.

The code was tested using 3 different datasets – iris.csv, MNIST and xor.csv. The outputs are displayed in the output section below. For iris, the highest accuracy obtained is 0.813333 (104/150) with the given configurations for 100 epochs.
For MNIST, the accuracy value is 0.667100 for 30 epochs. The accuracy for XOR problem remained 0.5 with all the given configurations.

Ideas for enhancement:

- Separate test and validation sets can be used to avoid overfitting problems.
- The cost function and sigmoid transformation activation function can be replaced with another.

Outputs:

Outputs for the following data-sets with the given number of epochs, hidden units, mini-batch size and learning rate is shown below:

1) Dataset – **iris.csv**, epochs – 100, hidden nodes – 20, batchSize – 10, eta – 0.1

Code:

```
>> iris = csvread('iris.csv');
>> inputs = iris(:, 1:4).';
>> targets = iris(:, 5:7).';
>> BackProp(inputs, targets, [4, 20, 3], 100, 10, 0.1);
```

```
Epoch 1, MSE: 0.043461, Correct: 0 / 150,  Acc: 0.000000
```

Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

Epoch 2, MSE: 0.041087, Correct: 4 / 150, Acc: 0.026667
Epoch 3, MSE: 0.036741, Correct: 47 / 150, Acc: 0.313333
Epoch 4, MSE: 0.030827, Correct: 92 / 150, Acc: 0.613333
Epoch 5, MSE: 0.028787, Correct: 82 / 150, Acc: 0.546667
Epoch 6, MSE: 0.027464, Correct: 94 / 150, Acc: 0.626667
Epoch 7, MSE: 0.026440, Correct: 97 / 150, Acc: 0.646667
Epoch 8, MSE: 0.025666, Correct: 94 / 150, Acc: 0.626667
Epoch 9, MSE: 0.025087, Correct: 94 / 150, Acc: 0.626667
Epoch 10, MSE: 0.024648, Correct: 94 / 150, Acc: 0.626667
Epoch 11, MSE: 0.024302, Correct: 94 / 150, Acc: 0.626667
Epoch 12, MSE: 0.024019, Correct: 95 / 150, Acc: 0.633333
Epoch 13, MSE: 0.023779, Correct: 95 / 150, Acc: 0.633333
Epoch 14, MSE: 0.023571, Correct: 98 / 150, Acc: 0.653333
Epoch 15, MSE: 0.023387, Correct: 99 / 150, Acc: 0.660000
Epoch 16, MSE: 0.023222, Correct: 100 / 150, Acc: 0.666667
Epoch 17, MSE: 0.023072, Correct: 100 / 150, Acc: 0.666667
Epoch 18, MSE: 0.022937, Correct: 100 / 150, Acc: 0.666667
Epoch 19, MSE: 0.022812, Correct: 100 / 150, Acc: 0.666667
Epoch 20, MSE: 0.022696, Correct: 100 / 150, Acc: 0.666667
Epoch 21, MSE: 0.022589, Correct: 100 / 150, Acc: 0.666667
Epoch 22, MSE: 0.022488, Correct: 100 / 150, Acc: 0.666667
Epoch 23, MSE: 0.022392, Correct: 101 / 150, Acc: 0.673333
Epoch 24, MSE: 0.022301, Correct: 101 / 150, Acc: 0.673333
Epoch 25, MSE: 0.022215, Correct: 101 / 150, Acc: 0.673333
Epoch 26, MSE: 0.022132, Correct: 101 / 150, Acc: 0.673333
Epoch 27, MSE: 0.022052, Correct: 101 / 150, Acc: 0.673333
Epoch 28, MSE: 0.021975, Correct: 101 / 150, Acc: 0.673333
Epoch 29, MSE: 0.021900, Correct: 101 / 150, Acc: 0.673333
Epoch 30, MSE: 0.021828, Correct: 102 / 150, Acc: 0.680000
Epoch 31, MSE: 0.021757, Correct: 102 / 150, Acc: 0.680000
Epoch 32, MSE: 0.021689, Correct: 102 / 150, Acc: 0.680000
Epoch 33, MSE: 0.021622, Correct: 102 / 150, Acc: 0.680000
Epoch 34, MSE: 0.021557, Correct: 102 / 150, Acc: 0.680000
Epoch 35, MSE: 0.021493, Correct: 102 / 150, Acc: 0.680000
Epoch 36, MSE: 0.021430, Correct: 102 / 150, Acc: 0.680000
Epoch 37, MSE: 0.021369, Correct: 102 / 150, Acc: 0.680000
Epoch 38, MSE: 0.021308, Correct: 102 / 150, Acc: 0.680000
Epoch 39, MSE: 0.021249, Correct: 102 / 150, Acc: 0.680000
Epoch 40, MSE: 0.021190, Correct: 102 / 150, Acc: 0.680000
Epoch 41, MSE: 0.021132, Correct: 102 / 150, Acc: 0.680000
Epoch 42, MSE: 0.021075, Correct: 102 / 150, Acc: 0.680000
Epoch 43, MSE: 0.021019, Correct: 103 / 150, Acc: 0.686667
Epoch 44, MSE: 0.020963, Correct: 104 / 150, Acc: 0.693333

Brunda Chouthoy
CSC 578: Project 1
Implementing a Neural Network

Epoch 45, MSE: 0.020907, Correct: 104 / 150, Acc: 0.693333
Epoch 46, MSE: 0.020852, Correct: 104 / 150, Acc: 0.693333
Epoch 47, MSE: 0.020798, Correct: 104 / 150, Acc: 0.693333
Epoch 48, MSE: 0.020743, Correct: 105 / 150, Acc: 0.700000
Epoch 49, MSE: 0.020690, Correct: 106 / 150, Acc: 0.706667
Epoch 50, MSE: 0.020636, Correct: 106 / 150, Acc: 0.706667
Epoch 51, MSE: 0.020582, Correct: 106 / 150, Acc: 0.706667
Epoch 52, MSE: 0.020529, Correct: 106 / 150, Acc: 0.706667
Epoch 53, MSE: 0.020476, Correct: 106 / 150, Acc: 0.706667
Epoch 54, MSE: 0.020423, Correct: 107 / 150, Acc: 0.713333
Epoch 55, MSE: 0.020370, Correct: 107 / 150, Acc: 0.713333
Epoch 56, MSE: 0.020318, Correct: 108 / 150, Acc: 0.720000
Epoch 57, MSE: 0.020265, Correct: 109 / 150, Acc: 0.726667
Epoch 58, MSE: 0.020212, Correct: 109 / 150, Acc: 0.726667
Epoch 59, MSE: 0.020159, Correct: 109 / 150, Acc: 0.726667
Epoch 60, MSE: 0.020106, Correct: 109 / 150, Acc: 0.726667
Epoch 61, MSE: 0.020054, Correct: 109 / 150, Acc: 0.726667
Epoch 62, MSE: 0.020001, Correct: 110 / 150, Acc: 0.733333
Epoch 63, MSE: 0.019948, Correct: 112 / 150, Acc: 0.746667
Epoch 64, MSE: 0.019894, Correct: 112 / 150, Acc: 0.746667
Epoch 65, MSE: 0.019841, Correct: 112 / 150, Acc: 0.746667
Epoch 66, MSE: 0.019788, Correct: 112 / 150, Acc: 0.746667
Epoch 67, MSE: 0.019734, Correct: 113 / 150, Acc: 0.753333
Epoch 68, MSE: 0.019680, Correct: 113 / 150, Acc: 0.753333
Epoch 69, MSE: 0.019626, Correct: 114 / 150, Acc: 0.760000
Epoch 70, MSE: 0.019572, Correct: 114 / 150, Acc: 0.760000
Epoch 71, MSE: 0.019518, Correct: 115 / 150, Acc: 0.766667
Epoch 72, MSE: 0.019464, Correct: 116 / 150, Acc: 0.773333
Epoch 73, MSE: 0.019409, Correct: 116 / 150, Acc: 0.773333
Epoch 74, MSE: 0.019354, Correct: 118 / 150, Acc: 0.786667
Epoch 75, MSE: 0.019299, Correct: 118 / 150, Acc: 0.786667
Epoch 76, MSE: 0.019244, Correct: 118 / 150, Acc: 0.786667
Epoch 77, MSE: 0.019189, Correct: 118 / 150, Acc: 0.786667
Epoch 78, MSE: 0.019134, Correct: 118 / 150, Acc: 0.786667
Epoch 79, MSE: 0.019078, Correct: 118 / 150, Acc: 0.786667
Epoch 80, MSE: 0.019023, Correct: 118 / 150, Acc: 0.786667
Epoch 81, MSE: 0.018967, Correct: 119 / 150, Acc: 0.793333
Epoch 82, MSE: 0.018912, Correct: 119 / 150, Acc: 0.793333
Epoch 83, MSE: 0.018856, Correct: 119 / 150, Acc: 0.793333
Epoch 84, MSE: 0.018800, Correct: 119 / 150, Acc: 0.793333
Epoch 85, MSE: 0.018745, Correct: 119 / 150, Acc: 0.793333
Epoch 86, MSE: 0.018689, Correct: 119 / 150, Acc: 0.793333
Epoch 87, MSE: 0.018633, Correct: 119 / 150, Acc: 0.793333

Epoch 88, MSE: 0.018578, Correct: 119 / 150, Acc: 0.793333
Epoch 89, MSE: 0.018522, Correct: 120 / 150, Acc: 0.800000
Epoch 90, MSE: 0.018467, Correct: 121 / 150, Acc: 0.806667
Epoch 91, MSE: 0.018411, Correct: 121 / 150, Acc: 0.806667
Epoch 92, MSE: 0.018356, Correct: 121 / 150, Acc: 0.806667
Epoch 93, MSE: 0.018301, Correct: 122 / 150, Acc: 0.813333
Epoch 94, MSE: 0.018246, Correct: 122 / 150, Acc: 0.813333
Epoch 95, MSE: 0.018191, Correct: 122 / 150, Acc: 0.813333
Epoch 96, MSE: 0.018136, Correct: 122 / 150, Acc: 0.813333
Epoch 97, MSE: 0.018081, Correct: 122 / 150, Acc: 0.813333
Epoch 98, MSE: 0.018026, Correct: 122 / 150, Acc: 0.813333
Epoch 99, MSE: 0.017971, Correct: 122 / 150, Acc: 0.813333
Epoch 100, MSE: 0.017916, Correct: 122 / 150, Acc: 0.813333

2) Dataset – **MNIST**, epochs – 30, hidden nodes – 30, batchSize – 10, eta – 3.0
Code:

```
>> load('mnistTrn.mat');
>> BackProp(trn,trnAns,[784,30,10],30,10,3);
```

Output:
Epoch 1, MSE: 0.002160, Correct: 2288 / 50000, Acc: 0.045760
Epoch 2, MSE: 0.002101, Correct: 4668 / 50000, Acc: 0.093360
Epoch 3, MSE: 0.001999, Correct: 9026 / 50000, Acc: 0.180520
Epoch 4, MSE: 0.001913, Correct: 12790 / 50000, Acc: 0.255800
Epoch 5, MSE: 0.001855, Correct: 15300 / 50000, Acc: 0.306000
Epoch 6, MSE: 0.001821, Correct: 16726 / 50000, Acc: 0.334520
Epoch 7, MSE: 0.001795, Correct: 17589 / 50000, Acc: 0.351780
Epoch 8, MSE: 0.001759, Correct: 18386 / 50000, Acc: 0.367720
Epoch 9, MSE: 0.001702, Correct: 20396 / 50000, Acc: 0.407920
Epoch 10, MSE: 0.001665, Correct: 21927 / 50000, Acc: 0.438540
Epoch 11, MSE: 0.001639, Correct: 22993 / 50000, Acc: 0.459860
Epoch 12, MSE: 0.001618, Correct: 23781 / 50000, Acc: 0.475620
Epoch 13, MSE: 0.001602, Correct: 24378 / 50000, Acc: 0.487560
Epoch 14, MSE: 0.001588, Correct: 24882 / 50000, Acc: 0.497640
Epoch 15, MSE: 0.001577, Correct: 25303 / 50000, Acc: 0.506060
Epoch 16, MSE: 0.001565, Correct: 25661 / 50000, Acc: 0.513220
Epoch 17, MSE: 0.001441, Correct: 29845 / 50000, Acc: 0.596900
Epoch 18, MSE: 0.001404, Correct: 31014 / 50000, Acc: 0.620280
Epoch 19, MSE: 0.001388, Correct: 31455 / 50000, Acc: 0.629100
Epoch 20, MSE: 0.001376, Correct: 31775 / 50000, Acc: 0.635500
Epoch 21, MSE: 0.001366, Correct: 31995 / 50000, Acc: 0.639900
Epoch 22, MSE: 0.001358, Correct: 32197 / 50000, Acc: 0.643940

Epoch 23, MSE: 0.001351, Correct: 32392 / 50000, Acc: 0.647840
Epoch 24, MSE: 0.001344, Correct: 32587 / 50000, Acc: 0.651740
Epoch 25, MSE: 0.001338, Correct: 32752 / 50000, Acc: 0.655040
Epoch 26, MSE: 0.001332, Correct: 32871 / 50000, Acc: 0.657420
Epoch 27, MSE: 0.001327, Correct: 33007 / 50000, Acc: 0.660140
Epoch 28, MSE: 0.001323, Correct: 33138 / 50000, Acc: 0.662760
Epoch 29, MSE: 0.001318, Correct: 33259 / 50000, Acc: 0.665180
Epoch 30, MSE: 0.001314, Correct: 33355 / 50000, Acc: 0.667100
>>

3) Dataset – **xor.csv**, epochs – 10, hidden nodes – 2, batchSize – 4, eta – 0.1

Code:

```
>> xor = csvread('xor.csv');
>> inputs = xor(:,1:2).';
>> targets = xor(:,3:3).';
>> BackProp(inputs, targets, [2, 2, 1], 10, 4, 0.1);
```

Output:

Epoch 1, MSE: 0.157179, Correct: 2 / 4, Acc: 0.500000
Epoch 2, MSE: 0.157030, Correct: 2 / 4, Acc: 0.500000
Epoch 3, MSE: 0.156880, Correct: 2 / 4, Acc: 0.500000
Epoch 4, MSE: 0.156729, Correct: 2 / 4, Acc: 0.500000
Epoch 5, MSE: 0.156576, Correct: 2 / 4, Acc: 0.500000
Epoch 6, MSE: 0.156421, Correct: 2 / 4, Acc: 0.500000
Epoch 7, MSE: 0.156266, Correct: 2 / 4, Acc: 0.500000
Epoch 8, MSE: 0.156108, Correct: 2 / 4, Acc: 0.500000
Epoch 9, MSE: 0.155949, Correct: 2 / 4, Acc: 0.500000
Epoch 10, MSE: 0.155789, Correct: 2 / 4, Acc: 0.500000

4) Dataset – **xor.csv**, epochs – 10, hidden nodes – 2, batchSize – 1, eta – 0.1

Code:

```
>> BackProp (inputs,targets,[2,2,1],10,1,0.1);
```

Output:

Epoch 1, MSE: 0.145683, Correct: 2 / 4, Acc: 0.500000
Epoch 2, MSE: 0.145385, Correct: 2 / 4, Acc: 0.500000
Epoch 3, MSE: 0.145086, Correct: 2 / 4, Acc: 0.500000
Epoch 4, MSE: 0.144786, Correct: 2 / 4, Acc: 0.500000
Epoch 5, MSE: 0.144487, Correct: 2 / 4, Acc: 0.500000
Epoch 6, MSE: 0.144187, Correct: 2 / 4, Acc: 0.500000
Epoch 7, MSE: 0.143887, Correct: 2 / 4, Acc: 0.500000

Epoch 8, MSE: 0.143587, Correct: 2 / 4, Acc: 0.500000
Epoch 9, MSE: 0.143287, Correct: 2 / 4, Acc: 0.500000
Epoch 10, MSE: 0.142987, Correct: 2 / 4, Acc: 0.500000

5) Dataset – **xor.csv**, epochs – 20, hidden nodes – [3 2], batchSize – 1, eta – 0.1

Code:

```
>> BackProp (inputs,targets,[2,3,2,1],20,1,0.1);
```

Output:

Epoch 1, MSE: 0.148481, Correct: 2 / 4, Acc: 0.500000
Epoch 2, MSE: 0.148136, Correct: 2 / 4, Acc: 0.500000
Epoch 3, MSE: 0.147788, Correct: 2 / 4, Acc: 0.500000
Epoch 4, MSE: 0.147437, Correct: 2 / 4, Acc: 0.500000
Epoch 5, MSE: 0.147083, Correct: 2 / 4, Acc: 0.500000
Epoch 6, MSE: 0.146728, Correct: 2 / 4, Acc: 0.500000
Epoch 7, MSE: 0.146370, Correct: 2 / 4, Acc: 0.500000
Epoch 8, MSE: 0.146010, Correct: 2 / 4, Acc: 0.500000
Epoch 9, MSE: 0.145648, Correct: 2 / 4, Acc: 0.500000
Epoch 10, MSE: 0.145285, Correct: 2 / 4, Acc: 0.500000
Epoch 11, MSE: 0.144920, Correct: 2 / 4, Acc: 0.500000
Epoch 12, MSE: 0.144555, Correct: 2 / 4, Acc: 0.500000
Epoch 13, MSE: 0.144188, Correct: 2 / 4, Acc: 0.500000
Epoch 14, MSE: 0.143820, Correct: 2 / 4, Acc: 0.500000
Epoch 15, MSE: 0.143452, Correct: 2 / 4, Acc: 0.500000
Epoch 16, MSE: 0.143084, Correct: 2 / 4, Acc: 0.500000
Epoch 17, MSE: 0.142715, Correct: 2 / 4, Acc: 0.500000
Epoch 18, MSE: 0.142347, Correct: 2 / 4, Acc: 0.500000
Epoch 19, MSE: 0.141979, Correct: 2 / 4, Acc: 0.500000
Epoch 20, MSE: 0.141612, Correct: 2 / 4, Acc: 0.500000