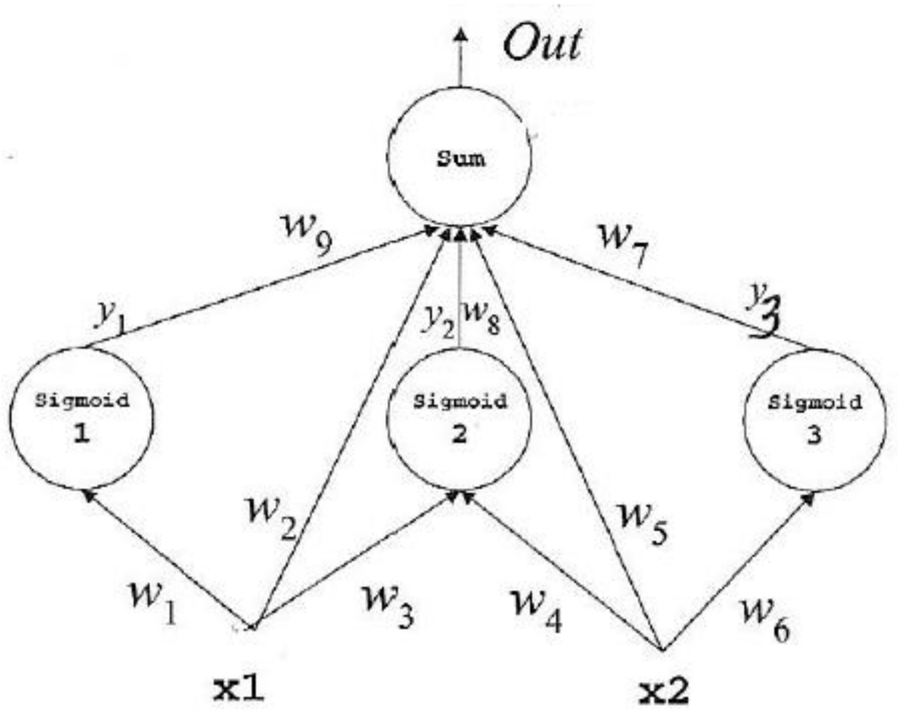
Neural Network

Incremental Multilayer Perceptron

Table of Contents

Part 1 – Requirements

Implement a program to demonstrate training of the multilayer perceptron (MLP) neural network with two inputs x1 and x2 given in the figure below. The network has one output summation unit (without a threshold) and three sigmoidal hidden units (also without thresholds). Assume the connections and their weights as shown in the figure below. Perform training of this MLP using the batch backpropagation algorithm with parameters: learning rate 1, and zero momentum.



With w1= -0.2, w2= 0.1, w3= 0.1, w4= -0.3, w5= -0.4, w6= 0.1, w7=0.2, w8= -0.3, w9= 0.2

Interpret the performance of the backprop algorithm using the following training vectors:

X1	X2	Out
1	0	1
1	1	1

Show the error derivatives beta, the weight updates, and the modified weights after processing each example (that is assuming the weights are updated incrementally after each example).

Part 2 – Matlab Code

File Name: bpnn Assign2.m

```
% Backpropagtion learning.  One hidden layer.
% by Dave Touretzky (modified by Nikolay Nikolaev and Frederic Marechal)
% https://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/matlab/
% Uses the following global variables for input and/or output:
%    Inputs1                  -  input patterns
%    Desired                  -  desired output patterns
%    LearnRate                -  learning rate parameter
%    Weights1                 -  first weight layer (updated by this routine)
%    Weights1_perceptron      -  perceptron layer weights
%    Weights2                 -  second weight layer (updated by this routine)
%    deltaW1                  -  initialize to 0 before first call
%    deltaW2                  -  initialize to 0 before first call
%    deltaW1Perceptron        -  initialize to 0 before first call
%    NDPS                     -  number of decimals

clearvars variables

fprintf('********** Model Definition **********\n');
fprintf('1. Incremental version (on-line)\n');
fprintf('2. No biais\n');
fprintf('3. Fwd/Backward prop implemented with loops (not matrices)\n');

%Load the data
load TrainingData_Assign1.dat;
load TrainingLabels_Assign1.dat;
Patterns = TrainingData_Assign1';
Desired = TrainingLabels_Assign1';

mseList = []; epochList =[];
LearnRate = 1.0;
%LearnRate = 0.15;
%LearnRate = 0.1;
%LearnRate = 0.05;
%LearnRate = 0.01;
Momentum = 0;
deltaW1 = 0; deltaW2 = 0; deltaW1Perceptron = 0;
TSS_Limit = 0.02;NDPS = 4;

[NINPUTS,NPATS] = size(Patterns);
[NOUTPUTS,NPATS] = size(Desired);

%No biais => else Inputs1 = [ones(1,NPATS); Patterns];
Inputs1 = [Patterns];
Inputs1Trans = Inputs1';
%Weights input
Weights1 = [-0.2 0 0.1 -0.3 0 0.1];
Weights1_perceptron = [0.1 -0.4];
Weights2 = [0.2,-0.3,0.2];

intialWeight1 = Weights1;
```

```
m = size(intialWeight1,1);
r = size(intialWeight1,2);
n = size(Inputs1,1);
o = size(Inputs1,2);

fprintf('********** Training Data & Initial Params **********\n');
fprintf('Learning: %0.4f\n',LearnRate);
fprintf('Momentum: %0.4f\n',Momentum);
PrintWeights(Weights1,Weights1_perceptron,Weights2 )

for epoch = [1:100]
    fprintf('********** Epoch %1d **********\n',epoch);
    epochList(epoch) = epoch;
    TSS = 0;
    %for each example
    for ex=1:NPATS
      NetIn0 = zeros(1);
      NetIn1 = [zeros(1);zeros(1);zeros(1)];
      NetIn2 = zeros(1);
      dW1Perceptron = 0;
      fprintf('***Example %1d\n',ex);

      %Step0 - Perceptron Layer forward propagation
      for k =1:n
        res = Weights1_perceptron(1,k) * Inputs1(k,ex);
        NetIn0(1) = NetIn0(1) + res;
      end
      %Out_Perceptron = round(NetIn0, NDPS);
      Out_Perceptron = round(NetIn0, NDPS);
      fprintf('Out_Perceptron = %0.4f\n',Out_Perceptron);

      %Step1 - Hidden Layer forward propagation
      %Step1.1 - Calculate Weight * Input for each hidden node
      windex = 1; p = 1;
      for k=1:n:r %step by 2 here
         cum = 0;
        for i=1:n
            res = (Weights1(1,windex) * Inputs1(i,ex));
            cum = cum + res;
            windex = k+i;
        end
        NetIn1(p) = NetIn1(p) + cum;
        p=p+1;
      end
      %Step1.2 Hidden layer output
      Hidden = round(1.0 ./( 1.0 + exp( -round(NetIn1, NDPS) )), NDPS);
      %Step1.3 - Calculate Weight * Input for each Hidden nodes to the output
      node
      Inputs2 = Hidden;
      for k =1:size(Inputs2,1)
        res = Weights2(1,k) * Inputs2(k);
        NetIn2(1) = NetIn2(1) + res;
      end
      Out_Hidden = round(NetIn2, NDPS);
      fprintf('Out_Hidden = %0.4f\n',Out_Hidden);
      %Step1.4 Output of the final and perceptron layers
      Out = round(Out_Hidden + Out_Perceptron,NDPS);
      fprintf('Out = %0.4f\n',Out);
```

```
%Step2 - Backward propagation of errors
Error = round(Desired(ex) - Out,NDPS);
fprintf('Error = %0.4f\n',Error);
Beta =  Error;
fprintf('Beta = %0.4f\n',Beta);
%Step2.1 - Calculate Beta * Hidden_Out_Weights
Weights2Trans =  Weights2';
m3 = size(Weights2Trans,1);
bperr = [m3];
for j =1:m3
  res = Weights2Trans(j) * Beta;
  bperr(j) = res;
end
bperr = round(bperr, NDPS);
%Step2.1 - Calculate Hidden Betas
for i =1:size(Hidden,1)
  HiddenBeta(i) = Hidden(i) * (1.0 - Hidden(i)) * bperr(i);
end
HiddenBeta = round(HiddenBeta,NDPS);
PrintHiddenBetas(HiddenBeta)

%Step3 - Generate Delta Weights:
%Step3.1 - Delta Hidden to Ouput Weights:
Inputs2Trans =  Inputs2';
dW2 = [0,0,0];
for i =1:size(Inputs2Trans,2)
  dW2(1,i) = Beta(1) * Inputs2Trans(1,i) ;
end
dW2 = round(dW2, NDPS);
%Step3.2 - Delta Input to Hidden Weights:
dW1 = [0 0 0 0 0 0];
k=1; windex = 1;
for i=1:size(HiddenBeta,2)
  cum = 0;
  for j=1:n
      if (intialWeight1(1,windex) ~= 0)
          dW1(windex) = HiddenBeta(k) * Inputs1(j,ex);
      end
      windex = windex+1;
  end
  k = k +1;
end
%Step3.3 - Delta Input to Out Weights:
for i=1:n
  dW1Perceptron(1,i) = Error * Inputs1Trans(ex,i) ;
end
dW1Perceptron = round(dW1Perceptron, NDPS);
%Step3.4 - Add the learning date and momentum
deltaW2 = round(LearnRate * dW2 + Momentum * deltaW2,NDPS);
deltaW1 = round(LearnRate * dW1 + Momentum * deltaW1,NDPS);
deltaW1Perceptron = round(LearnRate * dW1Perceptron + Momentum *
deltaW1Perceptron,NDPS);

%Step4 - Update Weights incrementally:
Weights2 = round(Weights2 + deltaW2,NDPS);
Weights1 = round(Weights1 + deltaW1,NDPS);
Weights1_perceptron = round(Weights1_perceptron +
deltaW1Perceptron,NDPS);
PrintWeights(Weights1,Weights1_perceptron,Weights2 )
```

```
      %Step5 - Calc TSS errors:
      TSS = TSS + round(sum( Error^2 ),NDPS);
      fprintf('TSS = %0.4f\n', TSS);
    end

      %Step6 - Calcualte the MSE:
      MSE = round(TSS/NPATS,NDPS);
      mseList(epoch) = MSE;
      fprintf('---> MSE = %0.4f\n',MSE);

      %Step7 - Stop when convergence is achieved
      %if TSS < TSS_Limit, break, end
end

%Plot Learning Rate
PlotMse(epochList,mseList, LearnRate, Momentum);
```

File Name: PrintHiddenBetas.m

```
function PrintHiddenBetas(HiddenBeta)
fprintf('Beta_Y1 = %0.4f\n',HiddenBeta(1));
fprintf('Beta_Y2 = %0.4f\n',HiddenBeta(2));
fprintf('Beta_Y3 = %0.4f\n',HiddenBeta(3));
```

File Name: PrintWeights.m

```
function PrintWeights(Weights1,Weights1_perceptron,Weights2)
fprintf('W1: %0.4f\n',Weights1(1,1));
fprintf('W2: %0.4f\n',Weights1_perceptron(1,1));
fprintf('W3: %0.4f\n',Weights1(1,3));
fprintf('W4: %0.4f\n',Weights1(1,4));
fprintf('W5: %0.4f\n',Weights1_perceptron(1,2));
fprintf('W6: %0.4f\n',Weights1(1,6));
fprintf('W7: %0.4f\n',Weights2(1,3));
fprintf('W8: %0.4f\n',Weights2(1,2));
fprintf('W9: %0.4f\n',Weights2(1,1));
```

Part 3 – Detailed Output

This is a partial output for the first epoch of the two training examples.

********** Model Definition **********
1. Incremental version (on-line)
2. No biais
3. Fwd/Backward prop implemented with loops (not matrices)
********** Training Data & Initial Params **********
Learning: 1.0000
Momentum: 0.0000
W1: -0.2000
W2: 0.1000
W3: 0.1000
W4: -0.3000
W5: -0.4000
W6: 0.1000
W7: 0.2000
W8: -0.3000
W9: 0.2000
********** Epoch 1 **********
***Example 1
Out_Perceptron = 0.1000
Out_Hidden = 0.0325
Out = 0.1325
Error = 0.8675
Beta = 0.8675
Beta_Y1 = 0.0429
Beta_Y2 = -0.0649
Beta_Y3 = 0.0434
W1: -0.1571
W2: 0.9675
W3: 0.0351
W4: -0.3000
W5: -0.4000
W6: 0.1000
W7: 0.6338
W8: 0.1554
W9: 0.5905
TSS = 0.7526

***Example 2
Out_Perceptron = 0.5675
Out_Hidden = 0.6723
Out = 1.2398
Error = -0.2398
Beta = -0.2398
Beta_Y1 = -0.0352
Beta_Y2 = -0.0092
Beta_Y3 = -0.0379
W1: -0.1923
W2: 0.7277
W3: 0.0259
W4: -0.3092
W5: -0.6398
W6: 0.0621
W7: 0.5079
W8: 0.0513
W9: 0.4800
TSS = 0.8101
---> MSE = 0.4051

Part 4 – MSE Graphs

The below graphs show the MSE shape change for different learning rate.