Report for the Maltilayer NN code

Ali Nehrani

Resumen

Implementing simple neural network for learning given data. Modifying the gradient decent and plotting decition boundary. There are some parameters that we can tune them in order to make the algorithm work for the given data. Beside implementing decition boundary and Cross Entropy cost, I provided different version of the gradient decent and controlling.

Contents

1 Introduction 1

2 Parameters and My codes 1
2.1 Cross Entropy cost function 2
2.2 Decision Boundary code 2
2.3 Accelerated gradient control 2
2.4 Armijo linesearch method 2

3 Text Outputs 3

4 Plottings 3

5 Conclusion 4

6 Final Codes 6
6.1 MSE efficient code 6
6.2 Cross entropy efficient code 8

1. Introduction

In this home work I prepared 2 codes; one for MSE cost function and the other for cross entropy objective function. I designed a neural network with "5" neuron with MSE cost that can learn the data using the Nestrov's accelerated gradient decent and for cross entropy function my neural network can learn with 6 hidden layers. I described the parameters and corresponding results in the following.

2. Parameters and My codes

There are some parameters that we were allowed to modify and I modified them. I listed the my modified parameters in the following:
nV: number of neurons
eta: the learning rate in the gradient decent.
I used accelerated gradient decent which requires another parameter *mu* and I set the values as follows for the MSE case:

```
% Parameters for MSE cost function code
nV  = 5;        % TUNE  (number of hidden units)
eta = 0.87;     % TUNE  (learning rate)
mu  = 0.087     % Accelerated gradient parameter
```

The average iterations for converging in this case is: The value of the parameters are almost same in the Cross Entropy case

```
% Parameters for Cross Entropy cost function code
nV  = 6;        % TUNE  (number of hidden units)
eta = 0.47;     % TUNE  (learning rate)
mu  = 0.077     % Accelerated gradient parameter
```

The average iterations for converging in this case is:

I used Nestrov's accelerated gradient decent (NAG) to achieve faster convergence. My NAG code is

```
% Parameters for MSE and Cross Entropy cost function code
    if it==1
       vect_p = -eta*dw;
    end
    % Accelerated Gradient
    vect = mu*vect_p - eta*dw ; %
    w = w - mu * vect_p + (1+mu)*vect; % + beta*randn(nV,3)
    vect_p = vect;
```

The NAG that I used in both cases are almost same.
Because Starting point is random, sometimes the algorithm fall into local minima and doesnt converge. To solve this problem I check the convergency after several iterations and if the average error is above some value, then I reset the starting point. Beside I modify the learning rate to help it to learn faster.

```
% Parameters for MSE and Cross Entropy cost function code
%   Trying to reset the starting point
    if (mod(it,5000) == 0)
     if E >= 1e-3
         eta = eta + .001;
         mu =  mu + .001;
         W = (rand(1,nV)-0.5);    % hidden->output weights
```

Correo electrónico: (Ali Nehrani)

```
        w = (rand(nV,nX)-0.5);  % input->hidden
    weights
    elseif (eta > .1 && mu >.1)
        eta = eta - 0.01;
        mu = mu - .01;
    end
```

2.1. Cross Entropy cost function

I changed the MSE code into ceross entropy code stright forward

```
% Cross Entropy Cost function
E = -T.*log(O) - (1-T).*log(1-O);
```

The gradient of the cross entropy cost function is coded as

```
% Gradient of Cross Entropy Cost function

%% Gradient of the input-hidden layer
dW = -(T*sigder(Oh)/O).*V +((1-T)*sigder(Oh)/(1-O))
    .*V;
%%  Gradient of the hidden-output layer
dw = -(T*sigder(Oh)/O).*(W.*sigder(Vh))'*x + ((1-T)*
    sigder(Oh)/(1-O)).*(W.*sigder(Vh))'*x;
```

2.2. Decision Boundary code

I coded the decision boundary stright forward by using for loops and sigmoid function and trained nuron weights. I consists of two parts; at the first part I make the decision background:

```
xmin = -2;  %
xmax = 2;  %
ymin = -2;  %
ymax = 2;   %
figure;
hold on
dx=0.1;
dy=0.1;

for x1=xmin:dx:xmax
  for y1=ymin:dy:ymax
        activation = forwardPass([x1 y1 1],w,W);
            if activation > 0.5
      plot(x1,y1,'.c', 'markersize', 5)
             else
      plot(x1,y1,'.m', 'markersize', 5)
           end
    hold on;
    end
end
legend('Class 1', 'Class 0');
```

At the second part I draw the output of the neural net on the decision areas

```
 nn_outpt = zeros(size(tr_x,1),1);
 for i=1:size(tr_x,1)
      nn_outpt(i) = forwardPass(tr_x(i,:),w,W);
 end
%
class1 = tr_x(find(nn_outpt>=.5),1:2);%
class0 = tr_x(find(nn_outpt<.5),1:2);%
plot(class1(:,1),class1(:,2), 'b>');
plot(class0(:,1),class0(:,2), 'go');
% include legend
legend('Class 1', 'Class 0');
%legend();
% label the axes.
xlabel('x');
ylabel('T');
```

2.3. Accelerated gradient control

I implemented accelerated gradient and as the input is random, sometimes the algorithm some times fall into local minima. In orther to avoid this I used some control in which the information of the error function is used. I sampled the error at some iterations and make decision based on the mean or max and min of the sampled data. The code for different controlling strategy is provided in the following:

```
% controlling section
    % For analysing the state of convergence I take
    samples and based on
    % tne average of the samples
    if  (mod(it,500) == 0)
           MeanE = [MeanE E];
 %           MeanE = MeanE_old + .9*(E-MeanE_old);%
 %           MeanE_old = MeanE
    end

%    dw_old = dw;
     if (mod(it,5000) == 0)
      if mean(MeanE) >= 1e-1
%         vect_p = -eta*dw;
%          eta = eta + .00001;
%         mu =  mu + .001;
% if not converged then start from point
%          w = .005*(rand(nV,nX)-0.5);  % input->
    hidden weights
%          W = (rand(1,nV)-0.5)*0.05;
%     else
%          eta = eta - 0.0001;
%          mu = mu - 0.001;
    if max(MeanE) < 1e-5
        beta = beta*.0001;
    elseif min(MeanE) > 1e-4
        beta = beta*1.2;
    end
      end
     MeanE=[];
    end
```

2.4. Armijo linesearch method

I also implemented the Armijo line search method in order to choose the optimal stepsize. But according to the stochastic nature of the starting point, this doesn't work in this case.

```
 % Armijo line search
     E_old = E;
     j    = 1;
     while (j>0)
         w1 = w + eta*dw;
         [O1, Oh1, V1, Vh1] = forwardPass(x,w1,W);
         E1 = 0.5*(T-O)^2;
         dw1 = -(T-O1)*sigder(Oh1)*(W.*sigder(Vh1))
    '*x;

          if ( E1 <= E_old+gamma*eta*(dw1'*dw1))
              j = 0;
  %            alpha_armijo = eta;
               mu = eta*delta;
          else
```

```
            eta = eta*delta
        end
        E_old = E1;
    end
   if j == 0
       clear O1 Oh1 V1 Vh1 E_old w1
   end
```

3. Text Outputs

With these parameters in both cases, my modified code was converged. Output of 5-layer perceptron after $10e5$ iterations:

```
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
RMS error over data points:0.00188
```

For more iterations I got even better results in both cases. In the following I provided output of 6-layer perceptron after $10e5$ iterations for cross entropy cost:

```
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 1.00  == 1.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
Desired Output 0.00  == 0.00 (Network Ouput)
RMS error over data points:0.00002
```

Based on these results I can conclude that the Cross entropy cost function works better that MSE cost function in terms of accuracy with constant equal iterations.

4. Plottings

I provided different plots from different running and I provided captions for them. As we have two classes, I denoted them with different colors. The training points are also denoted in the class areas that they are belonging (with different shapes).
Figure 1 represents sharp convergence with cross entropy const function and Figure 2 represents the corresponding decision boundary. I set number of iterations into $5 \times 10e5$. For comparing Cross Entropy case with MSE I run

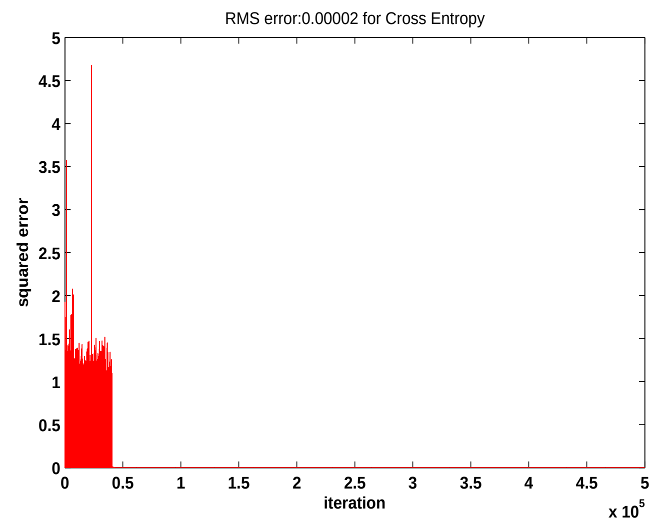

Figure 1: RMS error for 6 layer net with cross entropy case - sharp convergence

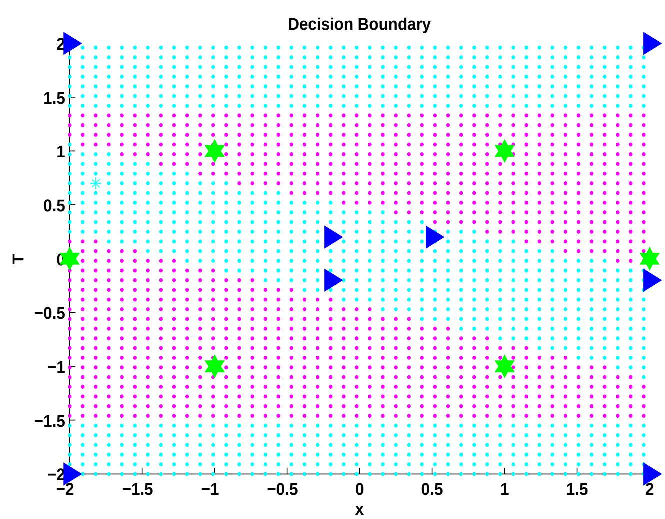

Figure 2: Decision boundary for 6-layer perceptron- after less than $10e5$ iterations converged-cross entropy case. Blue triangle are corresponding to class 1 and green stars with class 0 (cyan 1-area and magnet 0-area)

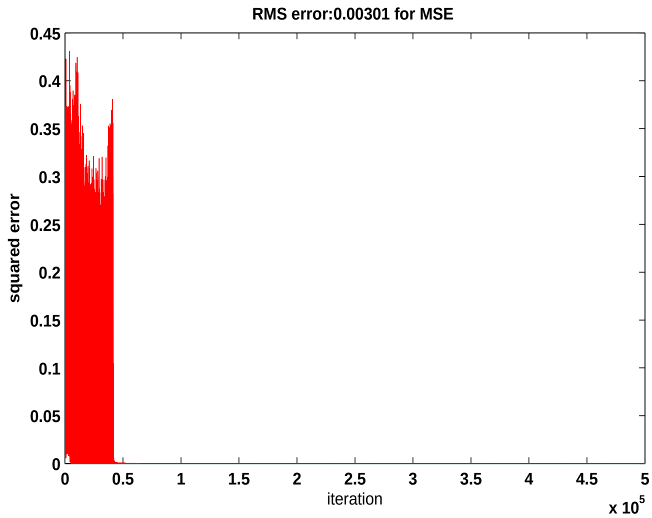

Figure 3: MSE error for 6-layer perceptron- after less than $5 \times 10e5$ iterations converged-MSE case.

the code for MSE cost with the same parameters and set number of layers into 6 .

Figure 3 represents the error vs iteration report with MSE cost function and Figure 4 is the corresponding decision boundary.

I also find the proper parameters ans also proper changes in the code so that it can solve the 5-layer network. I provided the code at the end of the report. Figure 5 represents the error vs iteration and Figure 6 represents the corresponding decision boundary.

In order to complete the report I run the codes with smoth gradient (the codes are noted in the MATLAB file and also in the report) and I provided the results in the report. I tested different layers with these softer gradients and as we can see smoother gradient reduction in the error plot. Figure 7 represents running for 5-layer perceptron, and Figure 8 represents the corresponding decision boundary. Figure 9 represents the error report over 6-layer network.

5. Conclusion

I noticed that the patterns that created over running the code with two cost functions; MSE and Cross Entropy are different. This difference are sometimes significant and sometimes the corresponding patterns are similar. I successfuly desined 5-layer network with MSE but doing this task with Cross Entropy was difficult and in the best case I successfuly modified parameters for 6-layer network although I tried different steepest decent algorithms (Accelerated, Armijo and other controls). With MSE I the network learned slower comparing to the Cross Entropy and accuracu is also worse w.r.t Cross Ent. In conclusion for this problem MSE converges better but slower and with slightly higher error.
When I set parameters, Cross Entropy case works better

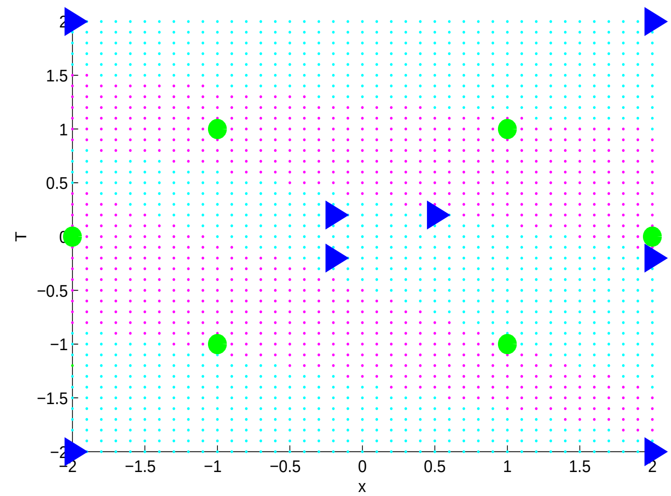

Figure 4: Decision boundary for 6-layer perceptron- after less than 10000 iterations converged-MSE case. Blue triangle are corresponding to class 1 and green stars with class 0 (cyan 1-area and magnet 0-area)

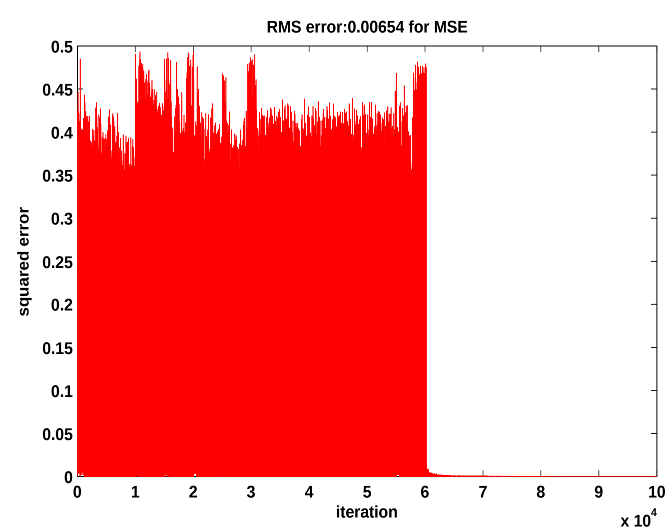

Figure 5: MSE error for 5-layer perceptron- after less than $10e5$ iterations with MSE cost case

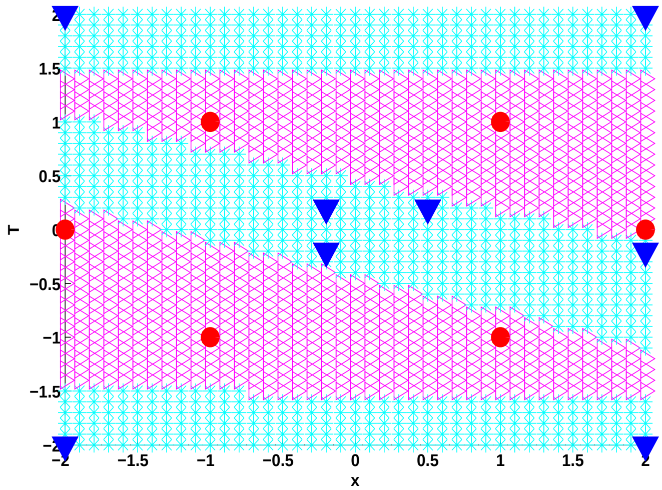

Figure 6: Decision boundary for 5-layer perceptron- after less than $10e5$ iterations with MSE cost case. Blue circles belonging to class 0 denoted with magnet area, blue triangles belong to class 1 denoted with cyan area

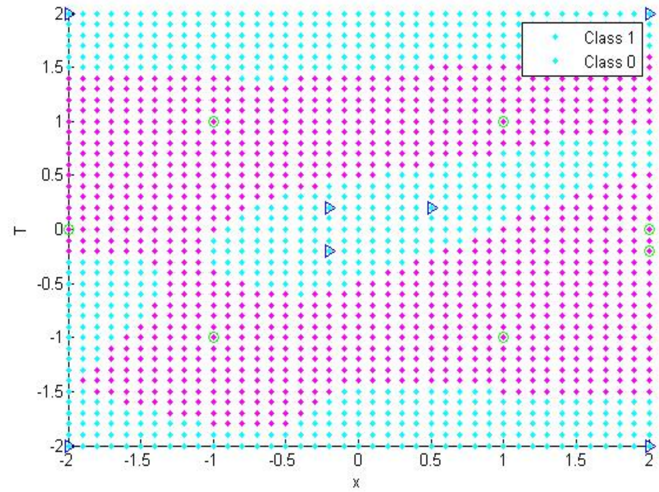

Figure 8: Decision boundary for 7-layer perceptron- after less than 10000 iterations not converged-cross entropy case

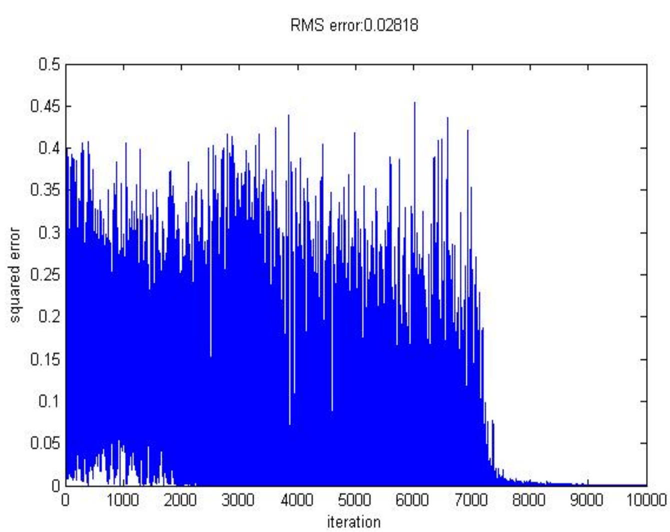

Figure 7: RMS error - 5 layer net, after less than 8000 iterations converged- MSE case

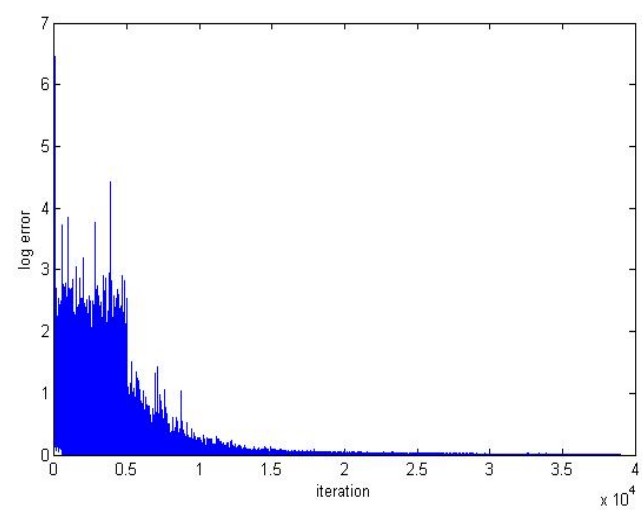

Figure 9: RMS error - 6 layer net, still fast converged- cross entropy case

than MSE in finding better local minimas. This fact is shown with the sum square error which is much less in the Cross Entropy case.

6. Final Codes

In the following I provide the MATLAB code that I used for 5-layer perceptron with MSE cost function.

6.1. MSE efficient code

I provided the whole source code, the main part and the functions. In this code I implemented Nestrov's accelerated gradient which responds well in this case.

```
function simpleBP2_hw
% ENTER YOUR NAME HERE:
% for 5 hidden layer
%%
clear all
clc
nX = 3;          % input dimension (do not change)
nV = 5;          % TUNE  (number of hidden units)
eta =.87;      % TUNE  (learning rate)

x = zeros(1,nX);  % input layer
V = zeros(1,nV);  % hidden layer
O = 0;            % output layer
%%
W = (rand(1,nV)-0.5)*0.05;   % hidden->output weights
w = (rand(nV,nX)-0.5)*0.01;  % input->hidden weights

% Do not change the training data
tr_data = [ -2    -2  1  1;
             2    -2  1  1;
            -2     2  1  1;
             2     2  1  1;
            -1    -1  1  0;
             1    -1  1  0;
            -1     1  1  0;
             1     1  1  0;
            -0.2  -0.2  1  1;
               2  -0.2  1  1;
            -0.2   0.2  1  1;
             0.5   0.2  1  1
               2   0.0  1  0
              -2   0.0  1  0];
tr_x = tr_data(:,1:nX);
tr_T = tr_data(:,end);

[tr_m, tr_n] = size(tr_data);
figure(1); clf;

MAXIT = 1000000;     % TUNE
% MAXIT = 50000;     % TUNE
err = zeros(MAXIT,1);
%%
dw_old = (rand(nV,3)-0.5)
mu = .087
%%
for it = 1:MAXIT,

    k = floor(rand*tr_m+1);
    x = tr_x(k,:);
    T = tr_T(k);
    [O, Oh, V, Vh] = forwardPass(x,w,W);

    E = 0.5*(T-O)^2;         % SUM SQUARED ERROR
%    E = -T.*log10(O) - (1-T).*log10(1-O); % SUM CROSS ENTROPY
    err(it+1)=E;

    if (mod(it,1000)==0)   % show learnig progress
        cla;
        plot(0:it,err(1:it+1));
        xlabel('iteration');
        ylabel('squared error');
        drawnow;
    end;
    % ------MSE
    dW = -(T-O)*sigder(Oh)*V;    % SUM SQUARED ERROR BASED dW
    %----------Cross Ent
%    dW = -(T*sigder(Oh)/O).*V +((1-T)*sigder(Oh)/(1-O)).*V;  % my Tuning - SUM CROSS ENTROPY
    % ---------- MSE
    dw = -(T-O)*sigder(Oh)*(W.*sigder(Vh))'*x; % SUM SQUARED ERROR BASED dw
    % ----------Cross Ent
%    dw = -(T*sigder(Oh)/O).*(W.*sigder(Vh))'*x + ((1-T)*sigder(Oh)/(1-O)).*(W.*sigder(Vh))'*x; % My tuning
    W = W - eta*dW; % SUM CROSS ENTROPY

    W = W - eta*dW;
%    w = w - eta* dw+0.005*(rand(nV,3)-0.5);   % TUNE
%    grw = beta*dw_old + dw
%    dw1 = alpha*dw - mtm*dw
%
%%
    if it==1
        vect_p = -eta*dw;
    end

    %w - sqmul*grw; %- 0.01*dw_old; %0.0005*(rand(nV,3)-0.5);   % TUNE

    % Accelerated Gradient
    vect = mu*vect_p - eta*dw ; %
    w = w - mu * vect_p + (1+mu)*vect; %
    vect_p = vect;

%    dw_old = dw;
    if (mod(it,5000) == 0)
     if E >= 1e-3
%         vect_p = -eta*dw;
         eta = eta + .001;
         mu =  mu + .001;
         W = (rand(1,nV)-0.5);    % hidden->output weights
         w = (rand(nV,nX)-0.5);  % input->hidden weights
     elseif (eta > .1 && mu >.1)
         eta = eta - 0.01;
         mu = mu - .01;
     end
    end


end
%%
plot((0:length(err)-1),err);
xlabel('iteration');
ylabel('squared error');
drawnow;

ms_err = 0;
```

```
for k=1:tr_m,
    x = tr_x(k,:);
    T = tr_T(k);
    [O, Oh, V, Vh] = forwardPass(x,w,W);
    ms_err = ms_err + (T-O)^2;

    fprintf('Desired Output %2.2f  == %2.2f (Network Ouput)\n',T,O);
end
ms_err = ms_err/tr_m;
rms_err = ms_err^0.5;
fprintf('RMS error over data points:%3.5f\n',rms_err);
title(sprintf('RMS error:%3.5f\n',rms_err));

%% ENTER YOUR code to show decision boundary here
% ---------------------------------------------
xmin = -2; %min(tr_x(:,1));
xmax = 2;%max(tr_x(:,1));
ymin = -2; %min(tr_x(:,2));
ymax = 2; %max(tr_x(:,2));
figure;
hold on
dx=0.1;
dy=0.1;

for x1=xmin:dx:xmax
  for y1=ymin:dy:ymax
        activation = forwardPass([x1 y1 1],w,W);
           if activation > 0.5
      plot(x1,y1,'*c', 'markersize', 9)
            else
      plot(x1,y1,'>m', 'markersize', 9)
           end

    hold on;
    end
end
legend('Class 1', 'Class 0');

 nn_outpt = zeros(size(tr_x,1),1);
 for i=1:size(tr_x,1)
     nn_outpt(i) = forwardPass(tr_x(i,:),w,W);
 end

%class = zeros(n_pairs, size(x,2)+size(O,2))
class1 = tr_x(find(nn_outpt>=.5),1:2);%(1:7,1:2)
class0 = tr_x(find(nn_outpt<.5),1:2);%(8:14,1:2)
plot(class1(:,1),class1(:,2), 'b>');
plot(class0(:,1),class0(:,2), 'go');
% include legend
legend('Class 1', 'Class 0');
%legend();
% label the axes.
xlabel('x');
ylabel('T');
% ---------------------------------------------

end  % main
%%


function [O,Oh,V,Vh] = forwardPass(x,w,W)
Vh = x*w';
V = sigmoid(Vh);
    V(end) = 1;
    %V(end-1) = 0;
Oh = V*W';
O = sigmoid(Oh);
end

function ret = sigmoid(x)
B = 2;
ret = 1./(1+exp(-B*x));
end

function ret = sigder(x)
B = 2;
ret = B*sigmoid(x).*(1-sigmoid(x));
end
```

6.2. Cross entropy efficient code

In the following dode works well for the cross entropy cost function. All parts are same unless the main part.

```
nX = 3;          % input dimension (do not change)
nV = 6;          % TUNE  (number of hidden units)
eta =.271;       % TUNE  (learning rate)

x = zeros(1,nX);  % input layer
V = zeros(1,nV);  % hidden layer
O = 0;            % output layer

W = (rand(1,nV)-0.5)*0.05;   % hidden->output weights
w = (rand(nV,nX)-0.5)*0.01;  % input->hidden weights
%%
% Do not change the training data
tr_data = [ -2    -2  1  1;
             2    -2  1  1;
            -2     2  1  1;
             2     2  1  1;
            -1    -1  1  0;
             1    -1  1  0;
            -1     1  1  0;
             1     1  1  0;
            -0.2  -0.2  1  1;
               2  -0.2  1  1;
            -0.2   0.2  1  1;
             0.5   0.2  1  1
               2   0.0  1  0
              -2   0.0  1  0];
tr_x = tr_data(:,1:nX);
tr_T = tr_data(:,end);

[tr_m, tr_n] = size(tr_data);
figure(1); clf;

MAXIT = 500000;      % TUNE
% MAXIT = 50000;      % TUNE
err = zeros(MAXIT,1);
%%
dw_old = (rand(nV,3)-0.5);
MeanE = [];
beta = 0.0005;
mu = .00217;

% armijo line search parameters
delta = 0.5;
gamma = 1e-4;
alpha = eta;
%%

for it = 1:MAXIT

     k = floor(rand*tr_m+1);
     x = tr_x(k,:);
     T = tr_T(k);
     [O, Oh, V, Vh] = forwardPass(x,w,W);

     E = 0.5*(T-O)^2;           % SUM SQUARED ERROR

     err(it+1)=E;

     if (mod(it,1000)==0)   % show learnig progress
         cla;
         plot(0:it,err(1:it+1));
         xlabel('iteration');
         ylabel('squared error');
         drawnow;
     end;

     dW = -(T-O)*sigder(Oh)*V;    % SUM SQUARED ERROR BASED dW


    dw = -(T-O)*sigder(Oh)*(W.*sigder(Vh))'*x; % SUM SQUARED ERROR BASED dw

     W = W - eta*dW; % SUM CROSS ENTROPY

     W = W - eta*dW;
%%
     if it==1
        vect_p = -eta*dw;
     end
     %w - sqmul*grw; %- 0.01*dw_old; %0.0005*(rand(nV,3)-0.5);    % TUNE
     % Accelerated Gradient
     vect = mu*vect_p - eta*dw ; %

     w = w - mu * vect_p + (1+mu)*vect + beta *(rand(nV,3)-0.5) ; % + beta*randn(nV,3)

     vect_p = vect;
     % For analysing the state of convergence I take samples and based on
     % tne average of the samples
     if  (mod(it,500) == 0)
           MeanE = [MeanE E];
 %            MeanE = MeanE_old + .9*(E-MeanE_old);%
 %            MeanE_old = MeanE
     end

%     dw_old = dw;
      if (mod(it,5000) == 0)
       if mean(MeanE) >= 1e-1
%
     if max(MeanE) < 1e-5
         beta = beta*.0001;
     elseif min(MeanE) > 1e-4
         beta = beta*1.2;
     end
       end
       MeanE=[];
      end
```