CS 486 Assignment 4

July 24, 2015

1. Proving sigmoid function properties

Where $\sigma(a) = \frac{1}{1+e^{-a}}$. Note alternative representations $\sigma(a) = \frac{e^a}{1+e^a} = 1 - \frac{1}{1+e^a}$.

Prove $\sigma(-a) = 1 - \sigma(a)$.

$$\begin{aligned}
\sigma(-a) &= 1 - \sigma(a) \\
&= 1 - \frac{1}{1+e^{-a}} \\
&= \frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}} \\
&= \frac{e^{-a}}{1+e^{-a}} \\
&= \frac{1}{1+e^{a}} \\
&= \frac{1}{1+e^{-(-a)}} \\
&= \sigma(-a)
\end{aligned}$$

Prove $\sigma^{-1}(a) = \ln(\frac{a}{1-a})$. Let $y = \sigma$, solve for a and switch the a and y back; the new y is the inverse σ^{-1} function.

$$\begin{aligned}
y &= \frac{1}{1+e^{-a}} \\
y + ye^{-a} &= 1 \\
e^{-a} &= \frac{1-y}{y} \\
\ln(e^{-a}) &= \ln(\frac{1-y}{y}) \\
a &= -\ln(\frac{1-y}{y}) \\
a &= \ln(\frac{y}{1-y}) \\
y &= \ln(\frac{a}{1-a}) \\
\sigma^{-1}(a) &= \ln(\frac{a}{1-a})
\end{aligned}$$

Prove $\frac{\partial\sigma}{\partial a} = \sigma(a)(1-\sigma(a))$.

$$\begin{aligned}
\frac{\partial \sigma}{\partial a} &= \sigma(a)(1-\sigma(a)) \\
&= \sigma(a) - \sigma^2(a) \\
&= \frac{e^a}{e^a+1} - (\frac{e^a}{e^a+1})^2 \\
&= \frac{e^a(e^a+1)}{(e^a+1)^2} - \frac{e^{2a}}{(e^a+1)^2} \\
&= \frac{e^{2a}+e^a-e^{2a}}{(e^a+1)^2} \\
&= \frac{e^a}{(e^a+1)^2} \\
&= \frac{\partial}{\partial a}(\frac{1}{1+e^{-a}}) \\
&= \frac{\partial \sigma}{\partial a}
\end{aligned}$$

2. Neural network using tanh instead of σ

For tanh and σ, their relation is a scaling plus a linear transformation. $\tanh(a) = 2\sigma(2a) - 1$. Subsituiting in definition of σ proves this.

$$\begin{aligned}
\tanh(a) &= 2\sigma(2a) - 1 \\
&= 2(\frac{1}{1+e^{-2a}}) - 1 \\
&= -\frac{e^{-2a}-1}{e^{-2a}+1} \\
&= \frac{1-e^{-2a}}{1+e^{-2a}}
\end{aligned}$$

This is (one of) the definition of *tanh*. Isolating for σ, $\sigma(a) = \frac{\tanh(a/2)+1}{2}$. So replacing the activation function $g(\cdot)$ from using σ to using *tanh* can be done by applying this transformation.

Originally, as given in assignment, $g = \sigma$:

$$y_i(x, W) = \sigma(\sum_j W_{ji}^{(2)} g(\sum_k W_{kj}^{(1)} x_k + W_{0j}^{(1)}) + W_{0i}^{(2)})$$

Note the replacement of $g(\sum_k W_{kj}^{(1)} x_k + W_{0j}^{(1)})$ when $g = \sigma$ with $\frac{g(\frac{\sum_k W_{kj}^{(1)} x_k + W_{0j}^{(1)}}{2})+1}{2}$ when $g = \tanh$.

Using $g = \tanh$, the equivalent network is:

$$y_i(x, W) = \sigma(\sum_j W_{ji}^{(2)} (\frac{\tanh(\frac{\sum_k W_{kj}^{(1)} x_k + W_{0j}^{(1)}}{2}) + 1}{2}) + W_{0i}^{(2)})$$

This can be rewritten in form given earlier. Let new vector V be a linear transformation of W.

$$y_i(x, W) = \sigma(\sum_j V_{ji}^{(2)} \tanh(\sum_k V_{kj}^{(1)} x_k + V_{0j}^{(1)}) + V_{0i}^{(2)})$$

Since W is the matrix of weights which were trained using $\sigma(a)$, then all of its elements should transformed to use $\tanh(a)$.

$$V = 2\sigma(2W) - 1$$

3. Threshold perceptron learning algorithm

A network with all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network.

The activation function g is typically either a hard threshold, in which case the unit is called a perceptron, or a logistic function, in which case the term sigmoid perceptron is used.

Is the dataset linearly separable?

The data is linearly seperable because training converged to weights that classified all of the training data correctly. If it *wasn't* linearly seperable then the perceptron cannot learn it.

Train and test accuracy

Trained correctly in 75 iterations Correct predictions: 93.9394% (341/363).

Final weights of the threshold perceptron

There are 65 total attributes: 64 image attributes plus 1 constant.

Final weights: -31 104 155 -242 -107 -56 -360 -119 80 158 -170 -115 24 -293 -505 -294 -87 60 -165 -137 -81 -166 -5 -188 -54 -220 -62 80 -184 -154 -276 16 -225 98 -113 90 18 191 50 -87 50 120 17 658 307 194 -87 401 -170 -321 -224 156 498 -148 192 238 49 -117 -327 -230 -180 183 393 53 90

Matlab source code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 3. j53sun (#20387090)
clear;
clc;
load trainData.csv;
load trainLabels.csv;
load testData.csv;
load testLabels.csv;
% Add bias nodes
data = [ones(length(trainLabels),1), trainData];
tdata = [ones(length(testLabels),1), testData];
% Scale the labels to [0,1]
label = trainLabels - min(trainLabels);
tlabel = testLabels - min(testLabels);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Threshold Perceptron: learn the weights
weights = zeros(1,size(data,2));
for i = 1:500 % limit set arbitrary at 500 (it stops earlier)
    weights = percept_threshold(weights,data,label);
    % Keep training until all training data is classified correctly
    if trained_correctly(weights, data, label);
        disp(['Trained correctly in ', num2str(i), ' iterations']);
```

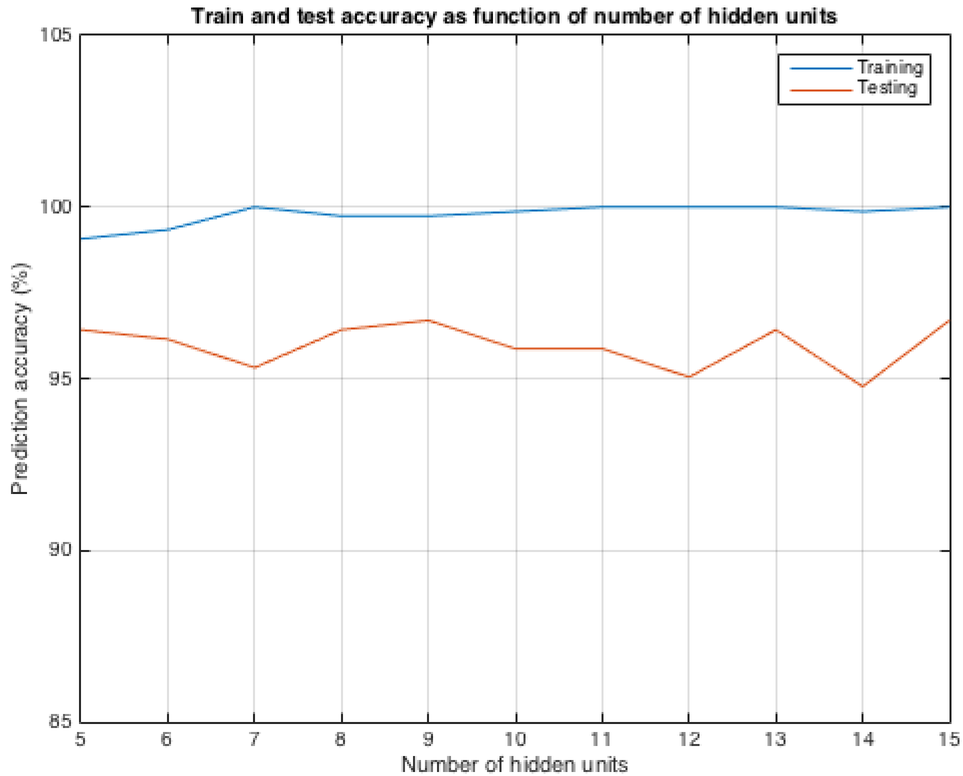
```
        break;
    end
end

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Classify the testing data with the weight vector
predictions = predict_using_weights(weights, tdata, tlabel);
num_correct = sum(predictions == tlabel);
total = length(predictions);
disp(['Correct predictions: ' num2str(100*num_correct/total) '% (' ...
      num2str(num_correct) '/', num2str(total) ')']);
disp(['Final weights: ' num2str(weights)]);

function [ weights ] = percept_threshold(weights, data, label)
%F_PERCEPT_THRESHOLD
for i = 1:length(label)
    x = data(i,:);
    y = label(i,:);
    a = f_step(weights * x'); % step-wise activation
    weights = weights + (y - a) * x;
end
end

function [ output ] = trained_correctly( weights, data, label )
%F_TRAINED_CORRECTLY Returns 1 if all correct, 0 else.
for i = 1 : length(data)
    if label(i) ~= f_step(weights * data(i,:)')
        output = 0;
        return;
    end
end
output = 1;
return;
end

function [predictions] = predict_using_weights(weights, tdata, tlabel)
%F_PREDICT_USING_WEIGHTS returns predictions on data given weight.
predictions = zeros(size(tlabel));
for i = 1 : length(tdata)
    predictions(i) = f_step(weights * tdata(i,:)');
end
end

function [ output ] = f_step( input )
%F_STEP Step-wise activation function
output = input >= 0;
end
```

4. Feed forward neural network

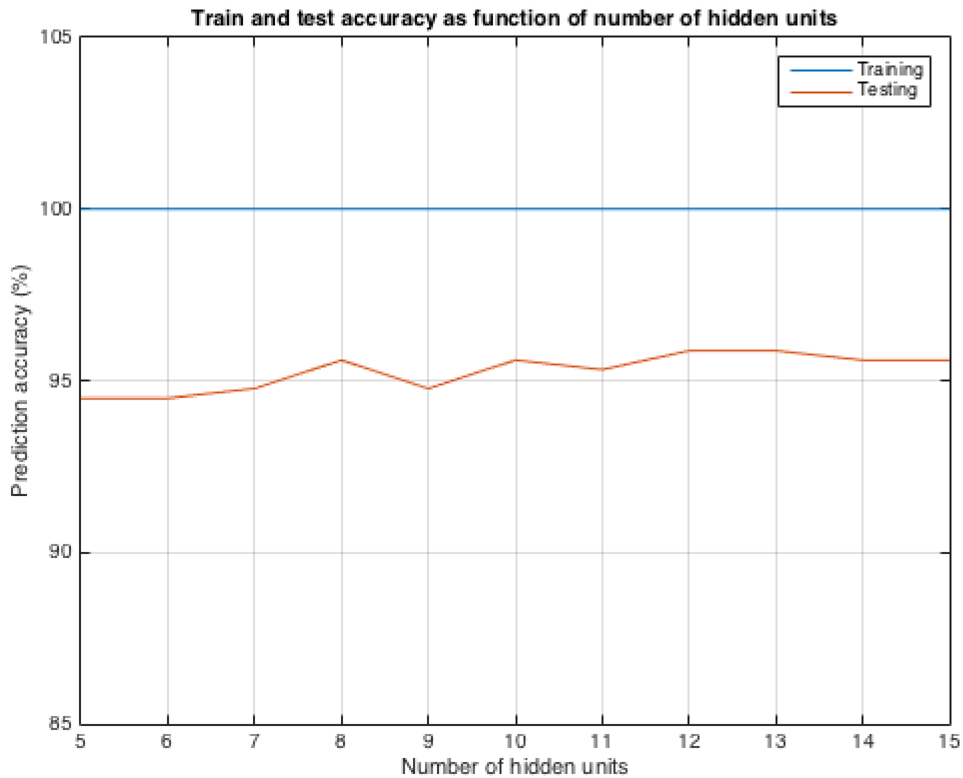
Graph of train and test accuracy as function of number of hidden nodes (5 to 15)

I produced two graphs, one with regularization of input and one without. The assignment didn't ask for regularization but I include it here because it should work better, but it doesn't.

No regularization



94.7 to 96.7.
With regularization



94.5 to 95.8.

Discussion of results in the graph

More hidden units seem better in general, but not necessary a non-decreasing increase in accuracy. Having a certain number of hidden nodes comparable to the number of input nodes seem to work better.

In the best case, 100% training data and 96.7% test data was categorized correct. In the worst case, 99.0% training data and 94.7% test data was correct.

Which algorithm (threshold perceptron vs neural network) performs best

Neural net works better as it was able to predict more accurately. In the worst case for the neural net, it was still better than the threshold perceptron. But this is expected, as neural net is more complicated and fits data better. Neural net is also able to seperate data that is not linearly seperatable, something a perceptron network cannot do.

A perceptron network with m outputs is really m separate networks, because each weight affects only one of the outputs. Thus, there will be m separate training processes.

Training a perceptron is much quicker than neural net. But if many more categories exists, such as 10 digits were to be trained, it would be quicker on the neural net as there are just additional output nodes. Whereas for the perceptron, it might not be able to seperate out the data.

Matlab source code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Question 4. j53sun (#20387090)
```

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Load training data
% Training data contains 753 images 8x8 pixels, giving us 64 input layer
% units, not counting bias node
clear; close all; clc;
load trainData.csv;
load trainLabels.csv;
load testData.csv;
load testLabels.csv;

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Initialize
input_layer_size = size(trainData, 2); % 8x8 input image of digit
num_labels = 2;                        % 6 or 7
alpha = 0.001;                         % learning rate
max_iter = 1000;
options = optimset('MaxIter', max_iter);
data = trainData;
tdata = testData;

% Scale the labels
label = trainLabels - min(trainLabels) + 1;
tlabel = testLabels - min(testLabels) + 1;

minNodes = 5;
maxNodes = 15;

% Store results in matrix for graphing
results = zeros(maxNodes - minNodes, 2);

for hidden_layer_size = minNodes:maxNodes     % variable hidden units

% Weights unrolled and initalized to random numbers in [-0.5,0.5]
weightsInputLayer = rand(hidden_layer_size, input_layer_size + 1);
weightsOutputLayer = rand(num_labels, hidden_layer_size + 1);
weightsInital = [weightsInputLayer(:) ; weightsOutputLayer(:)] - 0.5;

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Train neural network

fprintf('Training neural network with %d nodes %d iterations\n', ...
    hidden_layer_size, max_iter);

% Create a simplified reference to the cost function to be minimized
cost_fn = @(p) f_nnCostFunction(p, ...
                                input_layer_size,  ...
                                hidden_layer_size, ...
                                num_labels, data, label, alpha);

% Call an optimization library to do descent
[weights, cost] = fmincg(cost_fn, weightsInital, options);
```

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Unroll weights
weightsInput = reshape( ...
         weights(1:hidden_layer_size * (input_layer_size + 1)), ...
         hidden_layer_size, (input_layer_size + 1));

weightsHidden = reshape( ...
         weights(1 + hidden_layer_size * (input_layer_size + 1):end), ...
         num_labels, (hidden_layer_size + 1));


% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
%  Predict: compute the training set and test set accuracy

predtrain = f_predict(weightsInput, weightsHidden, data);
predtest = f_predict(weightsInput, weightsHidden, tdata);

acc_train = mean(double(predtrain == label)) * 100;
acc_test = mean(double(predtest == tlabel)) * 100;

fprintf('Accuracy (%d hidden nodes) training %f%%, test %f%%\n', ...
    hidden_layer_size, acc_train, acc_test);

results(hidden_layer_size - minNodes + 1, :) = [acc_train, acc_test];
end

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Plot the graph

creategraph(results);
disp(results);


function [J, grad] = f_nnCostFunction(nn_params, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, ...
                                   X, y, lambda)
%F_NNCOSTFUNCTION returns the cost and gradient of the neural network

% Unroll weights
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));
Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));
m = size(X, 1);

% Feedforward the neural network and return the cost in J
a1 = [ones(m,1) X];
z2 = Theta1*a1';
a2 = [ones(1, m); f_sigmoid(z2)];
```

```
z3 = Theta2*a2;
a3 = f_sigmoid(z3);

% Predictions
K = num_labels;
Y = zeros(K, m);
for i = 1:m
    Y(y(i), i) = 1;
end

% Cost function can be replaced to use mean sequared error later
costPos = -Y .* log(a3) ; % result in [1,753]
costNeg = -(1-Y) .* log(1-a3); % result in [1, 753]
cost = costPos + costNeg;
J = (1/m) * sum(cost(:)); % result in [1,1]

% Regularization
Theta1Filtered = Theta1(:,2:end); % result [5, 64]
Theta2Filtered = Theta2(:,2:end); % result [1, 5]
reg = lambda / (2*m) * ( sum( Theta1Filtered(:).^2 ) + sum( Theta2Filtered(:).^2) );
J = J + reg;


% Backpropagation, computing the gradients
Delta1 = 0;
Delta2 = 0;

for t = 1:m
    % Step 1 - Forward propagation: z_i and a_i
    a1 = [1 X(t,:)]';
    z2 = Theta1 * a1;
    a2 = [1; f_sigmoid(z2)];
    z3 = Theta2 * a2;
    a3 = f_sigmoid(z3);

    % Step 2a - Error calculations: output layer
    yt = Y(:,t);
    d3 = a3 - yt;

    % Step 2b - Error calculations: hidden layers
    d2 = Theta2Filtered' * d3 .* f_sigmoidGradient(z2);

    % Step 3a - Gradient calculation using a_i and the error d_i
    Delta2 = Delta2 + d3 * a2';
    Delta1 = Delta1 + d2 * a1';
end
Theta1_grad = (1/m) * Delta1;
Theta2_grad = (1/m) * Delta2;


% Regularization with the cost function and gradients
Theta1_grad(:,2:end) = Theta1_grad(:,2:end) + ((lambda/m) * Theta1Filtered);
```

```
Theta2_grad(:,2:end) = Theta2_grad(:,2:end) + ((lambda/m) * Theta2Filtered);

% Roll gradients
grad = [Theta1_grad(:) ; Theta2_grad(:)];

end

function p = f_predict(Theta1, Theta2, X)
%F_PREDICT Predict the label of an input given a trained neural network
m = size(X, 1);
h1 = f_sigmoid([ones(m, 1) X] * Theta1');
h2 = f_sigmoid([ones(m, 1) h1] * Theta2');
[~, p] = max(h2, [], 2);
end

function [ output ] = f_sigmoid( input )
%F_SIGMOID Sigmoid activation function
output = 1./(1 + exp(-input));
end

function g = f_sigmoidGradient(z)
%F_SIGMOIDGRADIENT returns the gradient of sigmoid function at z
g = f_sigmoid(z) .* (1 - f_sigmoid(z));
end

function [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
% Minimize a continuous differentialble multivariate function. Starting point
% is given by "X" (D by 1), and the function named in the string "f", must
% return a function value and a vector of partial derivatives. The Polack-
% Ribiere flavour of conjugate gradients is used to compute search directions,
% and a line search using quadratic and cubic polynomial approximations and the
% Wolfe-Powell stopping criteria is used together with the slope ratio method
% for guessing initial step sizes. Additionally a bunch of checks are made to
% make sure that exploration is taking place and that extrapolation will not
% be unboundedly large. The "length" gives the length of the run: if it is
% positive, it gives the maximum number of line searches, if negative its
% absolute gives the maximum allowed number of function evaluations. You can
% (optionally) give "length" a second component, which will indicate the
% reduction in function value to be expected in the first line-search (defaults
% to 1.0). The function returns when either its length is up, or if no further
% progress can be made (ie, we are at a minimum, or so close that due to
% numerical problems, we cannot get any closer). If the function terminates
% within a few iterations, it could be an indication that the function value
% and derivatives are not consistent (ie, there may be a bug in the
% implementation of your "f" function). The function returns the found
% solution "X", a vector of function values "fX" indicating the progress made
% and "i" the number of iterations (line searches or function evaluations,
% depending on the sign of "length") used.
%
% Usage: [X, fX, i] = fmincg(f, X, options, P1, P2, P3, P4, P5)
%
% See also: checkgrad
%
```

```
% Copyright (C) 2001 and 2002 by Carl Edward Rasmussen. Date 2002-02-13
%
%
% (C) Copyright 1999, 2000 & 2001, Carl Edward Rasmussen
%
% Permission is granted for anyone to copy, use, or modify these
% programs and accompanying documents for purposes of research or
% education, provided this copyright notice is retained, and note is
% made of any changes that have been made.
%
% These programs and documents are distributed without any warranty,
% express or implied.  As the programs were written for research
% purposes only, they have not been tested to the degree that would be
% advisable in any important application.  All use of these programs is
% entirely at the user's own risk.
%
% [ml-class] Changes Made:
% 1) Function name and argument specifications
% 2) Output display
%

% Read options
if exist('options', 'var') && ~isempty(options) && isfield(options, 'MaxIter')
    length = options.MaxIter;
else
    length = 100;
end


RHO = 0.01;                            % a bunch of constants for line searches
SIG = 0.5;       % RHO and SIG are the constants in the Wolfe-Powell conditions
INT = 0.1;    % don't reevaluate within 0.1 of the limit of the current bracket
EXT = 3.0;                    % extrapolate maximum 3 times the current bracket
MAX = 20;                         % max 20 function evaluations per line search
RATIO = 100;                                      % maximum allowed slope ratio

argstr = ['feval(f, X'];                      % compose string used to call function
for i = 1:(nargin - 3)
  argstr = [argstr, ',P', int2str(i)];
end
argstr = [argstr, ')'];

if max(size(length)) == 2, red=length(2); length=length(1); else red=1; end
S=['Iteration '];

i = 0;                                            % zero the run length counter
ls_failed = 0;                             % no previous line search has failed
fX = [];
[f1 df1] = eval(argstr);                      % get function value and gradient
i = i + (length<0);                                            % count epochs?!
s = -df1;                                        % search direction is steepest
d1 = -s'*s;                                                 % this is the slope
```

```
z1 = red/(1-d1);                                  % initial step is red/(|s|+1)

while i < abs(length)                                      % while not finished
  i = i + (length>0);                                      % count iterations?!

  X0 = X; f0 = f1; df0 = df1;                     % make a copy of current values
  X = X + z1*s;                                             % begin line search
  [f2 df2] = eval(argstr);
  i = i + (length<0);                                          % count epochs?!
  d2 = df2'*s;
  f3 = f1; d3 = d1; z3 = -z1;             % initialize point 3 equal to point 1
  if length>0, M = MAX; else M = min(MAX, -length-i); end
  success = 0; limit = -1;                     % initialize quanteties
  while 1
    while ((f2 > f1+z1*RHO*d1) | (d2 > -SIG*d1)) & (M > 0)
      limit = z1;                                         % tighten the bracket
      if f2 > f1
        z2 = z3 - (0.5*d3*z3*z3)/(d3*z3+f2-f3);                % quadratic fit
      else
        A = 6*(f2-f3)/z3+3*(d2+d3);                                 % cubic fit
        B = 3*(f3-f2)-z3*(d3+2*d2);
        z2 = (sqrt(B*B-A*d2*z3*z3)-B)/A;       % numerical error possible - ok!
      end
      if isnan(z2) | isinf(z2)
        z2 = z3/2;                  % if we had a numerical problem then bisect
      end
      z2 = max(min(z2, INT*z3),(1-INT)*z3);  % don't accept too close to limits
      z1 = z1 + z2;                                           % update the step
      X = X + z2*s;
      [f2 df2] = eval(argstr);
      M = M - 1; i = i + (length<0);                           % count epochs?!
      d2 = df2'*s;
      z3 = z3-z2;                    % z3 is now relative to the location of z2
    end
    if f2 > f1+z1*RHO*d1 | d2 > -SIG*d1
      break;                                                % this is a failure
    elseif d2 > SIG*d1
      success = 1; break;                                             % success
    elseif M == 0
      break;                                                          % failure
    end
    A = 6*(f2-f3)/z3+3*(d2+d3);                      % make cubic extrapolation
    B = 3*(f3-f2)-z3*(d3+2*d2);
    z2 = -d2*z3*z3/(B+sqrt(B*B-A*d2*z3*z3));        % num. error possible - ok!
    if ~isreal(z2) | isnan(z2) | isinf(z2) | z2 < 0   % num prob or wrong sign?
      if limit < -0.5                               % if we have no upper limit
        z2 = z1 * (EXT-1);                 % the extrapolate the maximum amount
      else
        z2 = (limit-z1)/2;                                   % otherwise bisect
      end
    elseif (limit > -0.5) & (z2+z1 > limit)          % extraplation beyond max?
      z2 = (limit-z1)/2;                                               % bisect
```

```
    elseif (limit < -0.5) & (z2+z1 > z1*EXT)       % extrapolation beyond limit
      z2 = z1*(EXT-1.0);                           % set to extrapolation limit
    elseif z2 < -z3*INT
      z2 = -z3*INT;
    elseif (limit > -0.5) & (z2 < (limit-z1)*(1.0-INT))   % too close to limit?
      z2 = (limit-z1)*(1.0-INT);
    end
    f3 = f2; d3 = d2; z3 = -z2;                   % set point 3 equal to point 2
    z1 = z1 + z2; X = X + z2*s;                      % update current estimates
    [f2 df2] = eval(argstr);
    M = M - 1; i = i + (length<0);                             % count epochs?!
    d2 = df2'*s;
  end                                                       % end of line search

  if success                                         % if line search succeeded
    f1 = f2; fX = [fX' f1]';
%     fprintf('%s %4i | Cost: %4.6e\r', S, i, f1);
    s = (df2'*df2-df1'*df2)/(df1'*df1)*s - df2;      % Polack-Ribiere direction
    tmp = df1; df1 = df2; df2 = tmp;                         % swap derivatives
    d2 = df1'*s;
    if d2 > 0                                      % new slope must be negative
      s = -df1;                              % otherwise use steepest direction
      d2 = -s'*s;
    end
    z1 = z1 * min(RATIO, d1/(d2-realmin));          % slope ratio but max RATIO
    d1 = d2;
    ls_failed = 0;                              % this line search did not fail
  else
    X = X0; f1 = f0; df1 = df0;  % restore point from before failed line search
    if ls_failed | i > abs(length)          % line search failed twice in a row
      break;                             % or we ran out of time, so we give up
    end
    tmp = df1; df1 = df2; df2 = tmp;                         % swap derivatives
    s = -df1;                                                    % try steepest
    d1 = -s'*s;
    z1 = 1/(1-d1);
    ls_failed = 1;                                    % this line search failed
  end
  if exist('OCTAVE_VERSION')
    fflush(stdout);
  end
end
% fprintf('\n');

function creategraph(YMatrix1)
%CREATEFIGURE1(YMATRIX1)
%  YMATRIX1:  matrix of y data

%  Auto-generated by MATLAB on 24-Jul-2015 02:30:30

% Create figure
figure1 = figure;
```

```
% Create axes
axes1 = axes('Parent',figure1,'YGrid','on','XGrid','on',...
    'XTickLabel',{'5','6','7','8','9','10','11','12','13','14','15'});

% To preserve the Y-limits of the axes
ylim(axes1,[85 105]);

box(axes1,'on');
hold(axes1,'on');

% Create multiple lines using matrix input to plot
plot1 = plot(YMatrix1);
set(plot1(1),'DisplayName','Training');
set(plot1(2),'DisplayName','Testing');

% Create xlabel
xlabel('Number of hidden units');

% Create ylabel
ylabel('Prediction accuracy (%)');

% Create title
title('Train and test accuracy as function of number of hidden units');

% Create legend
legend(axes1,'show');
```