Project 3: Essentials of Deep Learning

In this project, we learn and play essential mathematical components of deep learning on a simple artificial neural network by using the MATLAB script provided by C. F. Higham and D. J. Higham along with their paper "Deep Learning: An Introduction for Applied Mathematicians", arXiv:1801.05894v1, Jan. 2018. Minor changes have been made to the original script.

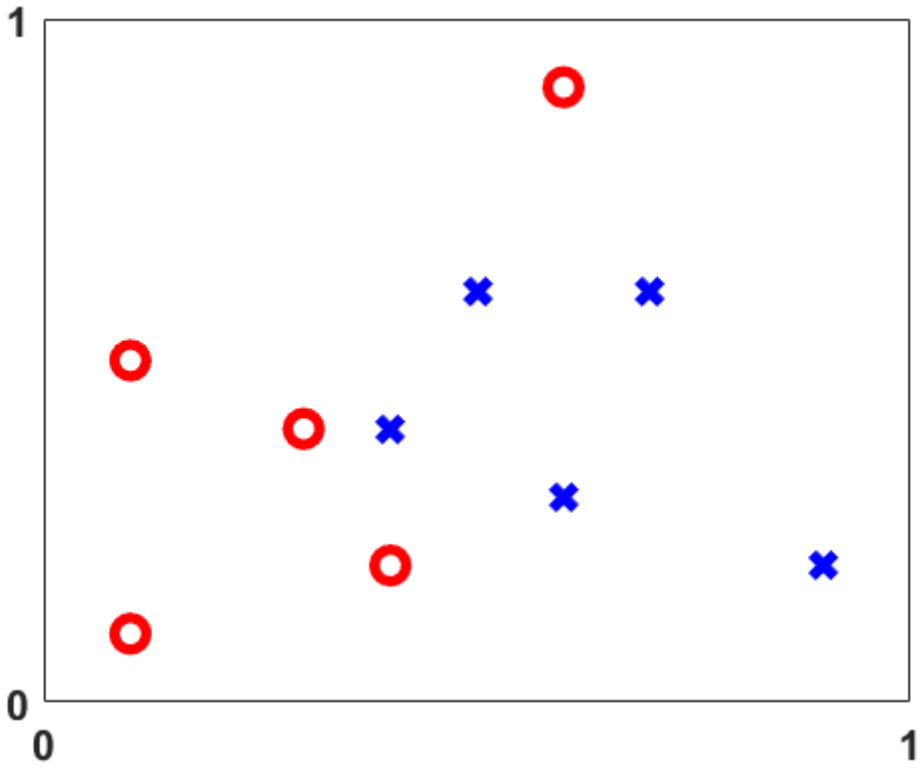
Contents

Step 1: set up the training data

```
function netbpsgd
```

```
% initial training data
m = 5;
n = 5;
x1 = [0.1,0.3,0.1,0.6,0.4,0.6,0.5,0.9,0.4,0.7];
x2 = [0.1,0.4,0.5,0.9,0.2,0.3,0.6,0.2,0.4,0.6];
y = [ones(1,m) zeros(1,n); zeros(1,m) ones(1,n)];

% you can upload the training data file, or use the following way to
% enter the training data on the screen as you like, where
%    m is the number of class A (red points)
%    n is the number of class B (blue points)
%    x1, x2 are the coordinates of the training data
%
%function [x1,x2] = netbpsgd(m,n)
%figure(1)
%set (gcf,'menubar','none');
%axes('position',[0 0 1 1]);
%[x1,x2] = ginput(m+n);
%y = [ones(1,m) zeros(1,n); zeros(1,m) ones(1,n)];

% plot the training data
figure(2)
clf
a1 = subplot(1,1,1);
plot(x1(1:m),x2(1:m),'ro','MarkerSize',12,'LineWidth',4)
hold on
plot(x1(m+1:m+n),x2(m+1:m+n),'bx','MarkerSize',12,'LineWidth',4)

a1.XTick = [0 1];
a1.YTick = [0 1];
a1.FontWeight = 'Bold';
a1.FontSize = 16;
xlim([0,1])
ylim([0,1])
```

```
print -dpng pic_xy.png

tic    % begin of clock count
```



Step 2: Initialize weights and biases

```
%rng(5000);
rng('default')
W2 = 0.5*randn(2,2);
W3 = 0.5*randn(3,2);
W4 = 0.5*randn(2,3);
b2 = 0.5*randn(2,1);
b3 = 0.5*randn(3,1);
b4 = 0.5*randn(2,1);
```

Step 3: set the learning rate and number of sgd iterations

```
eta = 0.05;  % learning rate
Niter = 3e5; % total number of iters, adjustable to your computational budget
```

Step 4: backpropagation to train the network

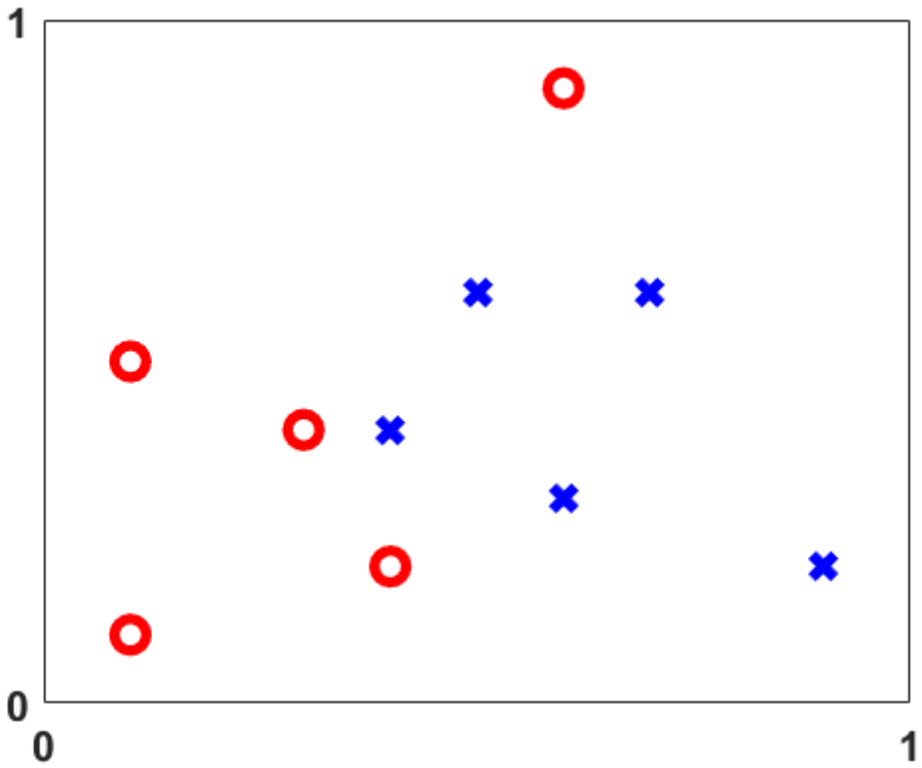
```
savecost = zeros(Niter,1);
for counter = 1:Niter
    k = randi(m+n);  %choose one point from the sample
    x = [x1(k); x2(k)];
    % Forward pass
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    % Backward pass
    delta4 = a4.*(1-a4).*(a4-y(:,k));
    delta3 = a3.*(1-a3).*(W4'*delta4);
```

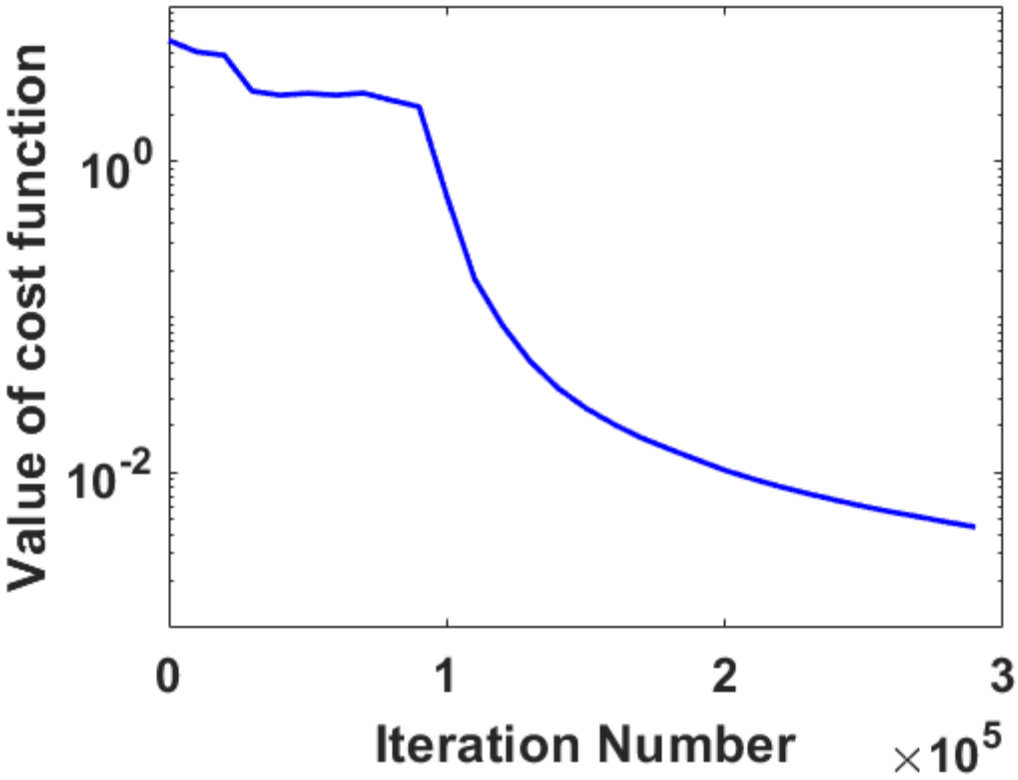
```
    delta2 = a2.*(1-a2).*(W3'*delta3);
    % Gradient step
    W2 = W2 - eta*delta2*x';
    W3 = W3 - eta*delta3*a2';
    W4 = W4 - eta*delta4*a3';
    b2 = b2 - eta*delta2;
    b3 = b3 - eta*delta3;
    b4 = b4 - eta*delta4;
    % Monitor progress
    newcost = cost(W2,W3,W4,b2,b3,b4);   % display cost to screen if you like
    savecost(counter) = newcost;
end

total_training_time = toc   % end of clock
```

```
total_training_time =

   15.0405
```

Step 5: show the ``convergence" of the cost function

```
figure(3)
clf
semilogy([1:1e4:Niter],savecost(1:1e4:Niter),'b-','LineWidth',2)
xlabel('Iteration Number')
ylabel('Value of cost function')
set(gca,'FontWeight','Bold','FontSize',18)

print -dpng pic_cost.png
```

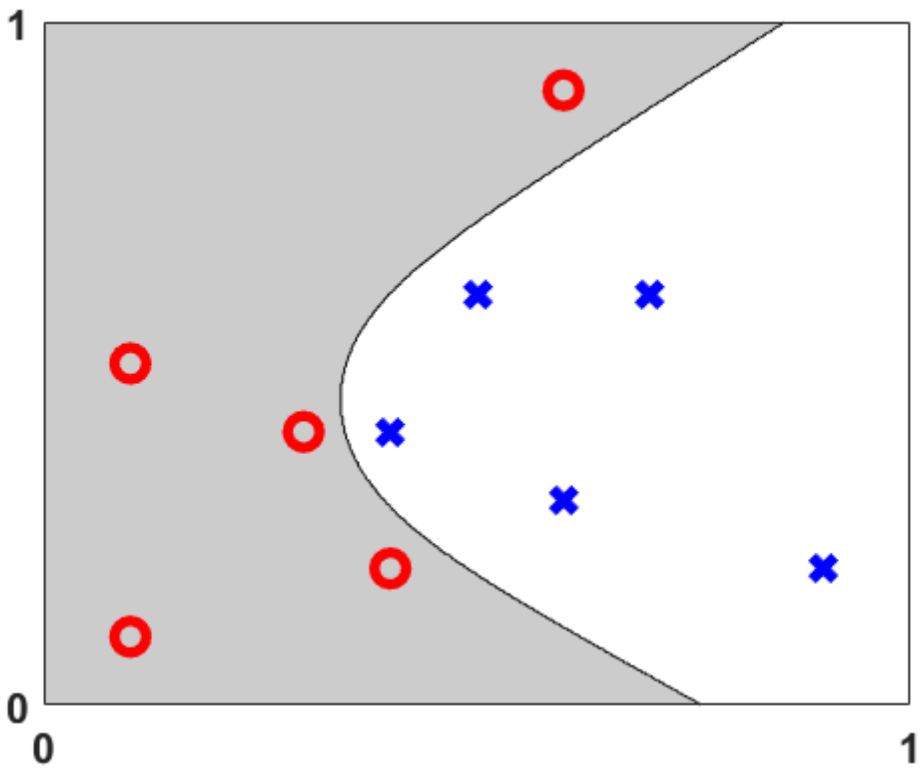




Step 6: show the classification by displaying shaded and unshaded regions

```
N = 500;
Dx = 1/N;
Dy = 1/N;
xvals = [0:Dx:1];
yvals = [0:Dy:1];
for k1 = 1:N+1
    xk = xvals(k1);
```

```
    for k2 = 1:N+1
        yk = yvals(k2);
        xy = [xk;yk];
        a2 = activate(xy,W2,b2);
        a3 = activate(a2,W3,b3);
        a4 = activate(a3,W4,b4);
        Aval(k2,k1) = a4(1);%a4 is a 2 by 1 vector to indicate class
        Bval(k2,k1) = a4(2);
        %Aval(k2,k1) = norm([1,0]'-a4,2);
        %Bval(k2,k1) = norm([0,1]'-a4,2);
     end
end
[X,Y] = meshgrid(xvals,yvals);

figure(4)
clf
a2 = subplot(1,1,1);
Mval = Aval>=Bval;
%Mval = Aval<=Bval;
contourf(X,Y,Mval,[0.5 0.5]) %receiving warning!!!

hold on
colormap([1 1 1; 0.8 0.8 0.8])
plot(x1(1:m),x2(1:m),'ro','MarkerSize',12,'LineWidth',4)
plot(x1(m+1:m+n),x2(m+1:m+n),'bx','MarkerSize',12,'LineWidth',4)
a2.XTick = [0 1];
a2.YTick = [0 1];
a2.FontWeight = 'Bold';
a2.FontSize = 16;
xlim([0,1])
ylim([0,1])

print -dpng pic_bdy_bp.png
```



Auxiliary functions

```
function costval = cost(W2,W3,W4,b2,b3,b4)
costvec = zeros(m+n,1);
for i = 1:m+n
    x =[x1(i);x2(i)];
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    costvec(i) = norm(y(:,i) - a4,2);
end
costval = norm(costvec,2)^2;

end % of nested function

function y = activate(x,W,b)
% ACTIVATE Evaluates sigmoid function.
% x is the input vector, y is the output vector
% W contains the weights, b contains the shifts
% The ith component of y is activate((Wx+b)_i)
% where activate(z) = 1/(1+exp(-z))
y = 1./(1+exp(-(W*x+b)));
end % of nested function
```

```
end
```

Task 1:

derive expressions of backpropagation and sgd for a network of L layers
with one neuron per layer.

Task 2:

Modify the script netbpsgd.m and report your results as follows:
Add
rng('default')
To the beginning of you code for every task!

(1) Add additional hidden layers and neurons to the existing network.
    Specifically, netbpsgd is a ``2-2-3-2'' network, change it to
    a ``2-5-5-5-2'' network. Using a learning rate of 0.05 and 3e5 number
    of sgd iterations to show the convergence behavior of the cost function
    and the classification with the training data set in netbpsgd.

(2) Replace the sigmoid activation function of netpbsgd to ReLU
    activation function activitation. Using a learning rate of 0.0025 and
    3e5 number of sgd iterations to show the convergence behavior of
    the cost function and the classification with the training data set
    in netbpsgd.

(3) Combine the tasks 2(1) and 2(2) to a network of five layers
    (``2-5-5-5-2'') and ReLU activation function, with the
    learning rate 0.0025 and 3e5 number of iterations. Show the
    convergence behavior of the cost function and the classification
    with the following training data set:

```
m = 12;
n = 8;
x1 = [0.1,0.05,0.05,0.1,0.35,0.65,0.9,0.95,0.95,0.9,0.65,0.35,0.7,0.3,0.3,0.7,0.25,0.75,0.5,0.5];
x2 = [0.1,0.65,0.35,0.9,0.95,0.95,0.9,0.65,0.35,0.1,0.05,0.05,0.7,0.7,0.3,0.3,0.5,0.5,0.75,0.25];
y = [ones(1,m) zeros(1,n); zeros(1,m) ones(1,n)];
y = [ones(1,m) zeros(1,n); zeros(1,m) ones(1,n)];
```

Published with MATLAB® R2019b