# Project 3: Essentials of Deep Learning

## 1 Task1: Backpropagation and sgd L layer network with one neuron per layer

Given a $L$ layered network with one neuron per layer, we want to derive expressions of backpropagation and sgd for this specific case.
Since we know the network has $L$ layers and 1 neuron per layer we know the network has the structure "$\underbrace{n-1-1-\ldots-1}_{L\text{-times}}$", where $n$ is the dimension of the input vector $x$. To get this structure, we have to change the dimensions of the weights $W^i$ and biases $b^i$ for $i \in \{2, \ldots, L\}$. Further we have to code $L-1$ forward passes and gradient steps for $W^i$ and $b^i$. Since there is only one neuron per layer, we know $W^i, b^i \in \mathbb{R}$. So we initialize the weights and biases as follows:

```
rng('default')
W2 = 0.5*randn(1,2); %starting vector x is of dimension 2
W3 = 0.5*randn(1,1);
...
WL = 0.5*randn(1,1);
b2 = 0.5*randn(1,1);
b3 = 0.5*randn(1,1);
...
bL = 0.5*randn(1,1)
```

In the backpropagation step we only have to add $a^i$ and $\delta^i$ for $i \in \{2, \ldots, L\}$ analogously.

## 2 Task2: Modifying script netbpsd.m

### 2.1 Adding additional hidden layers

In this subsection we will change the given "2-2-3-2" network *netbpsgd* to a "2-5-5-5-2" network. This means we have to add one more weight $W^5$ and bias $b^5$ and change the dimensions of the the given weights and biases in the script. After changing it we get following matrices:

```
rng('default')
W2 = 0.5*randn(5,2);
W3 = 0.5*randn(5,5);
W4 = 0.5*randn(5,5);
W5 = 0.5*randn(2,5);
b2 = 0.5*randn(5,1);
b3 = 0.5*randn(5,1);
b4 = 0.5*randn(5,1);
b5 = 0.5*randn(2,1);
```

We will keep the learning rate and number of sgd iterations unchanged at $\eta = 0.05$ and $Niter = 3e5$.

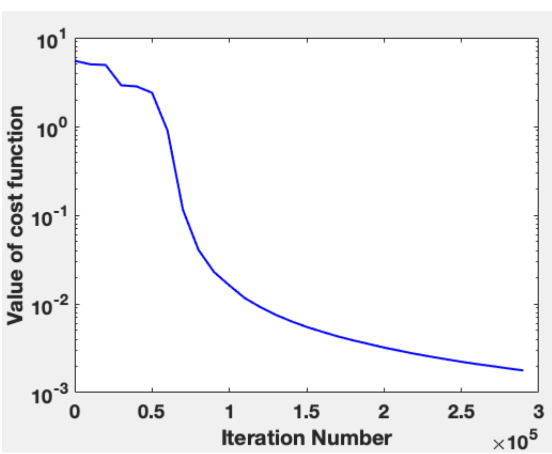
In the backpropagation step to train the network we will add one additional forward and backwards passing layer and gradient step for $a5$, $\delta 5, W^5$ and $b^5$.

After making small changes to the classification by displaying shaded and unshaded areas in the script and adding $a5$ in the auxiliary function *costval* we get following total training time for the "2-5-5-5-2" network:

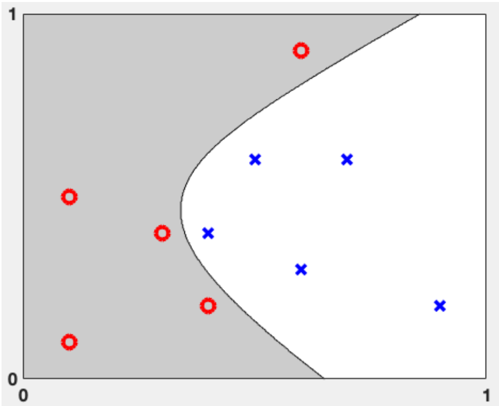
```
total_training_time =

    8.8144
```

This gives us following "convergence" diagram of the cost function:



and following classification:




## 2.2 ReLu function as the activation function

In this part we will use the function $ReLu(x) = \max(0, x)$ as the activation function. We will rewrite this function as follows:

$$f(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The derivate of $f$ is:

$$f('x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

We will code this function in matlab as follows:

```
function f = dactivate (x,W,b)
    a = W*x+b;
    f = 0;
    n = size(a,1);
    f = zeros(n,1);
  for i= 1:n

      if a(i,1) > 0
        f(i,1) = 1;
      end
```

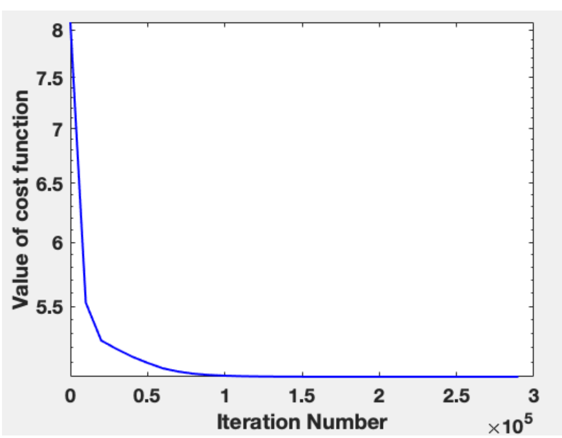
```

  end
end
```

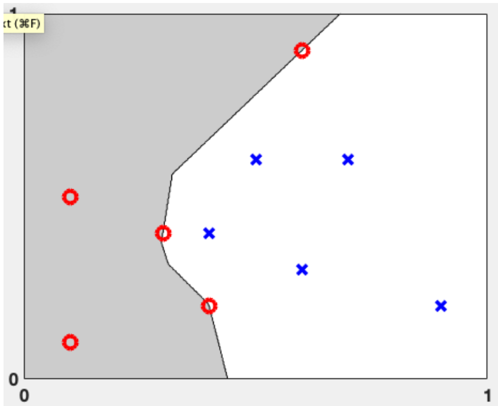
This derivative is needed for computing $\delta^i$ for $i \in \{2, \ldots 5\}$. We get following $\delta^i$'s by using the *dactivate* function:

```
    % Forward pass
    a2 = activate(x,W2,b2);
    a3 = activate(a2,W3,b3);
    a4 = activate(a3,W4,b4);
    a5 = activate(a4,W5,b5);
    % Backward pass
    delta5 = dactivate(a4,W5,b5) .* (a5-y(:,k));
    delta4 = dactivate(a3,W4,b4) .* (W5'*delta5);
    delta3 = dactivate(a2,W3,b3) .* (W4'*delta4);
    delta2 = dactivate(x,W2,b2) .* (W3'*delta3);
```

We get following "convergence" diagram for $\eta = 0.0025$ :



and following separation:

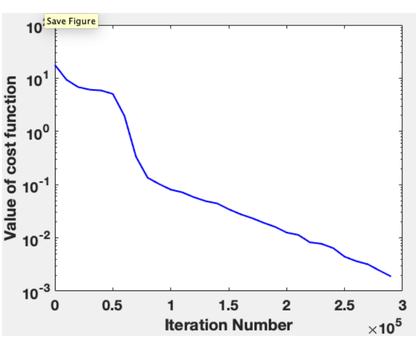



## 2.3 New testing set

We will now use following training data for our classification:

```
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)];
```

With the same learning rate as before we get following "convergence" diagram:



and following classification: