Project 4 - Essentials of Deep Learning

Linear Algebra and Learning from Data

Task 1: Derive expressions of backpropagation and sgd for a network of L layers

with one neuron per layer.

Answer: For a network with only one neuron per layer, the input and output of every layer is a scalar except for the first input which is the vector x corresponding to a sample in the training dataset.

Let's construct this network with the following notation:

X is the input, y is the output, and for every layer $i = 2, 3 \dots, L$

$$z_i = w_i a_{i-1} + b_i$$

$$a_i = \sigma(z_i)$$

With σ the activation function, $w_i \; and \; b_i$ the weight and bias of layer i. In our case, all $w_i \; and \; b_i$ are scalar. Note: $\hat{y} = z_L$

For layer 1:

$$z_1 = w_1 x + b_1 \; and \; a_1 = \sigma(z_1)$$

With $w_1 \; and \; b_i$ vectors.

Then the derived expression of backpropagation a scholastic gradient descent is given by:

$$\frac{\partial L}{\partial w_L} = \frac{\partial L}{\partial z_L}\frac{\partial z_L}{\partial w_L} = a_{L-1}(\hat{y} - y)$$

$$\frac{\partial L}{\partial b_L} = \frac{\partial L}{\partial z_L}\frac{\partial z_L}{\partial b_L} = \hat{y} - y$$

For all $i = 2, 3, \dots, L-1$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z_i}\frac{\partial z_i}{\partial w_i} = \delta_i a_i$$

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial z_i}\frac{\partial z_i}{\partial b_i} = \delta_i$$

$$\delta_i = \frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial z_{i+1}}\frac{\partial z_{i+1}}{\partial z_i} = \delta_{i+1}\frac{\partial(w_{i+1}\sigma(z_i) + b_i)}{\partial z_i} = \delta_{i+1} w_{i+1}\sigma'(z_i)$$

For layer 1 $x, w_1 \; and \; b_1$ are vectors:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \delta_1 x^T$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial b_1} = \delta_1$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial x} = \delta_1 \frac{\partial (w_1 x + b_1)}{\partial x} = \delta_1 w_1^T \sigma'(z_1)$$

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)];
```

Answer: We work on a constructed dataset of ten points.

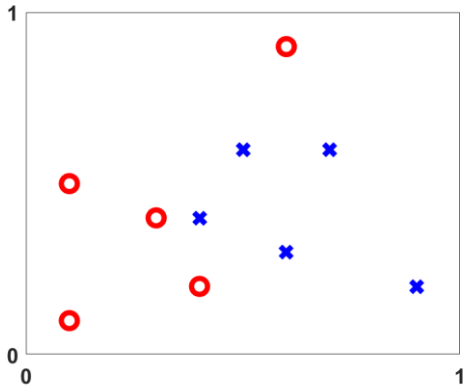

Figure 1. Training dataset for part 1

First, we construct a network of 5 layers with respectively 2, 5, 5, 5 and 2 neurons each. The parameters are kept as follows:

Learning rate = 0.05

Number of iterations: 3e5

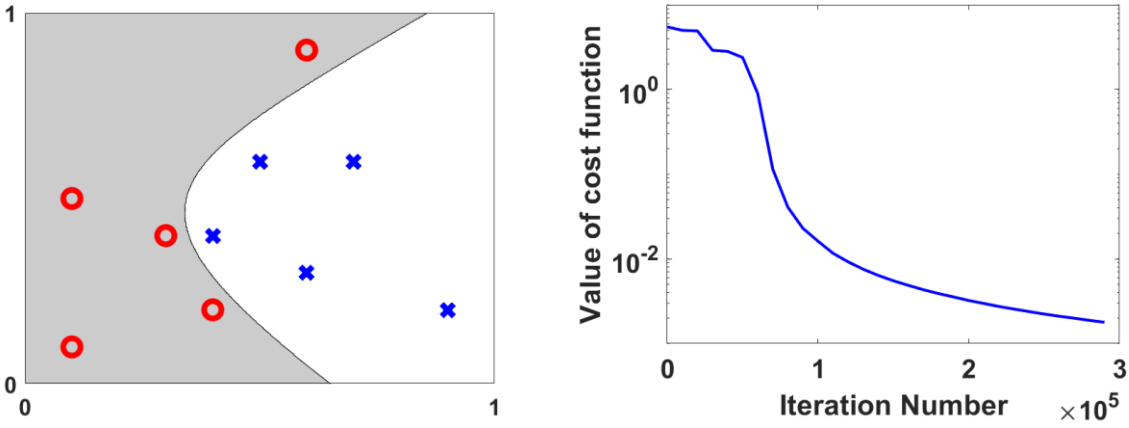

Figure 2. Results of the first model

The results of the first model are good on this first dataset. The points are well separated, and the cost function converges as expected. However, its convergence could probably be improved as it is quite slow for iteration number higher than 10^5.

Then we modify the code to have a network of 4 layers with respectively 2, 3, 3 and 2 neurons each. We use the Relu function as the activation function for every layer and change the learning rate to 0.0025. This is our second model.

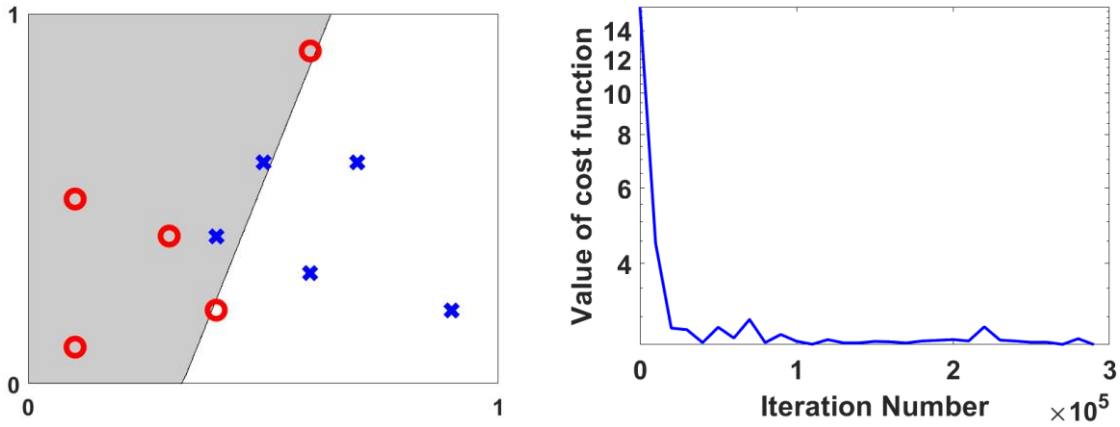

Figure 3. Results of the second model

The second model fails to correctly classify the data. The appearance of the straight separation line makes me think I made mistakes in the code. The objective of the activation function is to introduce non-linearity, but the non-linearity is not clearly visible in the results plotted here.

The third model is a combination of the two previous ones. It is a network of 5 layers with respectively 2, 5, 5, 5 and 2 neurons each. The relu activation function is used on every layer and the parameters are:

Learning rate = 0.0025

Number of iterations = 3e5

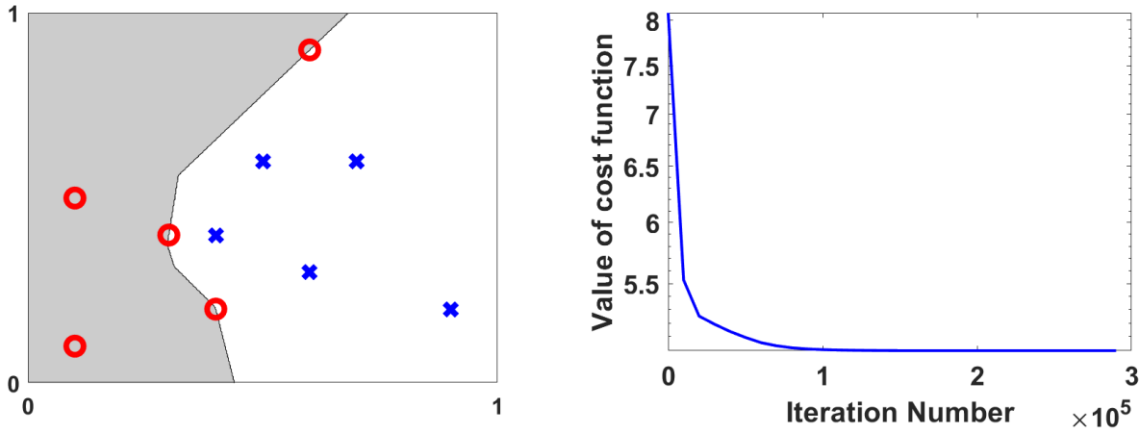

Figure 4. Results of the third model

This last model succeeds in classifying the data despite some points that are at the limits of their class. In terms of convergence, this model is better than the first one. Indeed, the cost function plunges in the first iterations and can be considered as optimal for a number of iterations equals to 10^5.

To test further the models, we use another dataset of points:

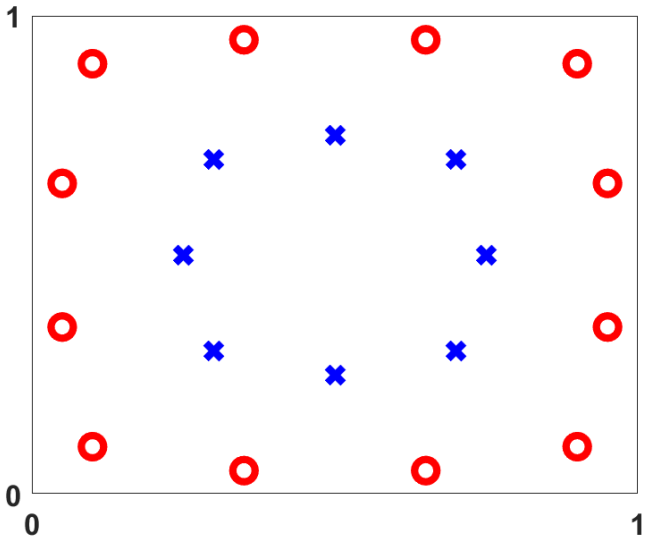

Figure 5. New dataset for the second part

Whereas the two first models fail to return a valid separation line, the last network works pretty well on this data. All the points are well classified, and the appearance of the classes is coherent with the data. The cost function converges as well but may need more iterations.

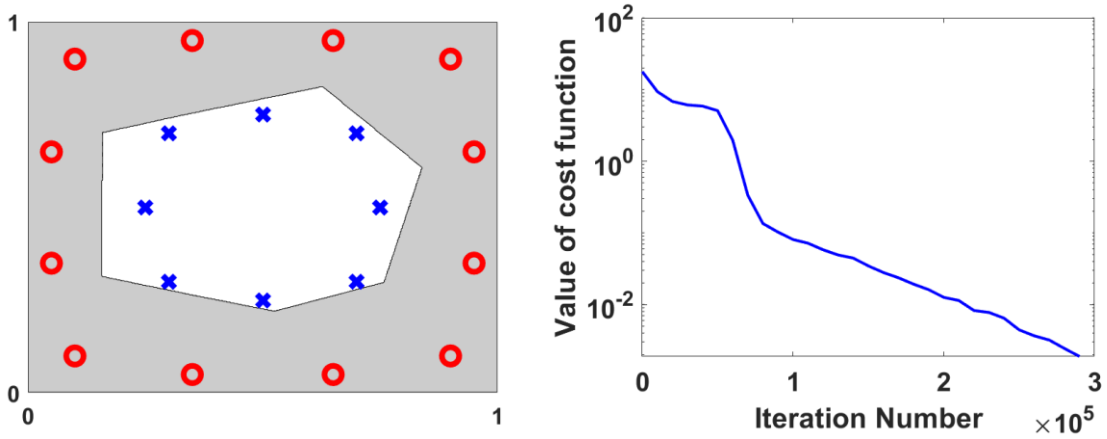

Figure 6. Results of the third model on the second dataset