Neural Networks: Implementing a Neural Network from scratch Deep Learning in Computer Vision Lab 2

Awet Haileslassie Gebrehiwot, Santiago Herrero Melo

I. INTRODUCTION

In this second lab session of *Deep Learning in Computer Vision* we will see how to build a neural network from scratch in Python with numpy. We will use the MNIST dataset, one of the most used datasets in machine learning research. This dataset consists of 70000 grayscale images of handwritten digits (of size 28x28). The project will be developed in python language, using the tensorflow environment.
A neural network consists of the following components

- *An input layer, x*
- *An arbitrary amount of hidden layers*
- *An output layer, y*
- *A set of weights and biases between each layer, w & b*
- *A choice of activation function for each hidden layer, .*

A. Single Neuron:

As a first exercise, we will consider a single neuron with 28x28=784 inputs and a single sigmoid unit generating the output. Our neuron will simply learn to distinguish between the digit 0 and the other digits (in the MNIST dataset).

B. A Neural Network with one Hidden Layer:

In this exercise, we will add an intermediate hidden layer which consists of 64 units. This means that the 784 inputs will be connected to each of the 64 units of the hidden layer, which will also be connected to the output layer. The purpose of this hidden layer is to create system that allows us a faster and better learning for the neural network. It also provide the discrimination necessary to be able to separate your training data.

C. Multiclass Neural Network:

In this occasion, we will recognize each of the ten different handwritten digits (from 0 to 9) individually. This means that now, the output will have ten units, being each of them connected to each of the units of the hidden layer.

II. IMPLEMENTATION

Taking the artificial neural network approach, the computer is fed training examples of known handwritten digits(MNIST dataset), that have been previously labeled as to which number they correspond to, and via the algorithm the computer then learns to recognize each character. The Neural network via single Neuron (Perceptron) algorithm allows the computer to learn by incorporating new data.

A. Single Neuron:

We divide the operation in forward propagation and backward propagation. The first thing to do is to create a vector of randomly initialized weights.

When performing forward propagation, we first multiply the inputs with their corresponding weights, and add the bias, if there is any, to obtain matrix Z. Afterwards, we calculate the sigmoid of the obtained matrix (equation 1). The result of the sigmoid function (A) will be then used to calculate the loss function of equation **??**, where Y is the vector that contains the labels of the input.

$$\sigma = \frac{1}{1 + exp(-Z)} \tag{1}$$

$$L = -(Y * log(A) + (1 - Y) * log(1 - A)) \tag{2}$$

When performing backward propagation, we just have to update the weights and bias according to a previously defined learning rate (α). The updating also depends on the input matrix and labels, and the result of the sigmoid function.

The forward propagation will be performed both in the train and test sets. For calculation the loss function, which can be seen in Figure (2c) for different number of epochs, we will calculate the mean of the obtained value for each epoch.

B. A Neural Network with one Hidden Layer:

In this occasion, the main difference is the design of the weights. We will have two different matrices, one for the weights of the input layer respect to the hidden layer, and one for the hidden layer respect to the output. Therefore, we will also need two bias values instead of one.

The forward propagation will be calculated in a similar way than in the previous case, but in this time we have to calculate two sigmoid functions with equation 1, the one that corresponds to the weighting of the input layer and the one that corresponds to the hidden layer. The loss function is calculated applying this last matrix to equation 2.

The backward propagation works also on a similar way, but once more, we have to do the double of operations, as we now have to update two weights matrices and two bias values.

As in the previous case, the forward propagation will be performed in both train and test set simultaneously for each epoch.

C. *Multiclass Neural Network:*

In this exercise, the output layer changes from the previous one, modifying its number of nodes from one to ten, as shown in Figure (1).

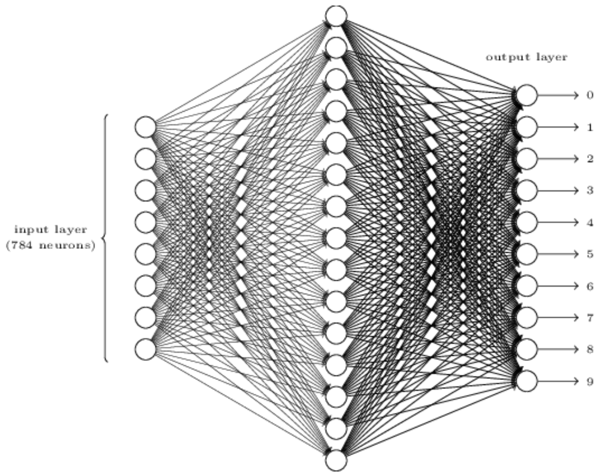


Fig. 1: Multi-class Neural Network with 10 outputs

The main modification respect to the previous exercise is this extension of the possible outputs, in which instead of differentiating between images with a ”0” and images that are not ”0”, we differentiate each of the images individually from each other (every digit from 0 to 9). Therefore, the weights matrix between the hidden layer and the output layer has been modified appropriately, having 10 rows (as this is the number of outputs) instead of only one.

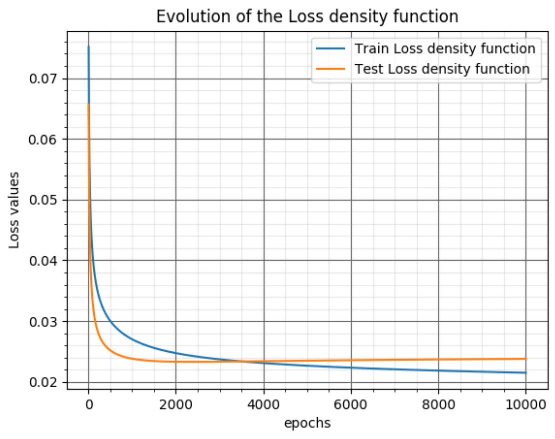
D. *Running the code*

For running the code we first have to activate the Tensorflow environment. Once that is done, we access the folder where the skeleton of the code is saved, and type the following line in the command window:
python lab2_skeleton.py for the single neuron script,
python lab2_2_skeleton.py for the single class script with hidden layer,
python lab2_3_skeleton.py for the multiple class with hidden layer script,

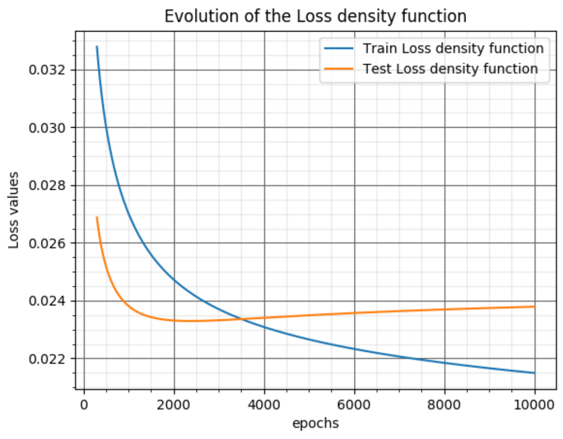
III. RESULTS AND ANALYSIS

A. *Single Neuron:*

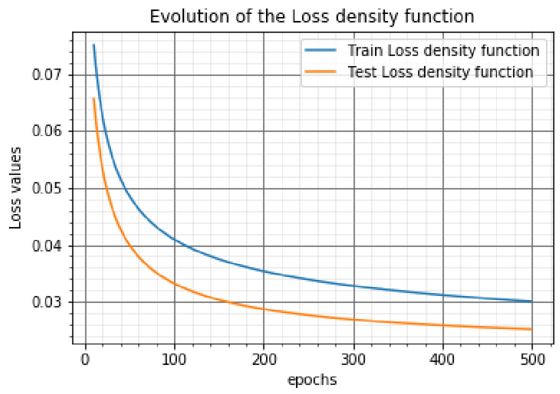
As we can observe form (Figure 2a and Figure 2b) the Test loss starts to increase after 2380 epoch. As it get closer to 4000 epoch, we can witness that the Test loss is getting higher than the Training loss which indicates the model is over-fitting to the training data-set.
The program has also run for only 500 epochs (Figure 2c), obtaining a loss value in the last epoch:
Test_Loss = 0.025196727549756617 and
Training_Loss = 0.030048830759622978.



(a) Loss density function for 10,000 epochs



(b) Zoom version of Loss density function for 10,000 epochs

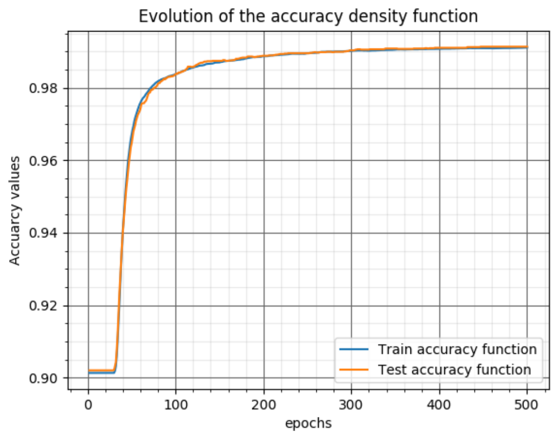


(c) Loss density function for 500 epoch

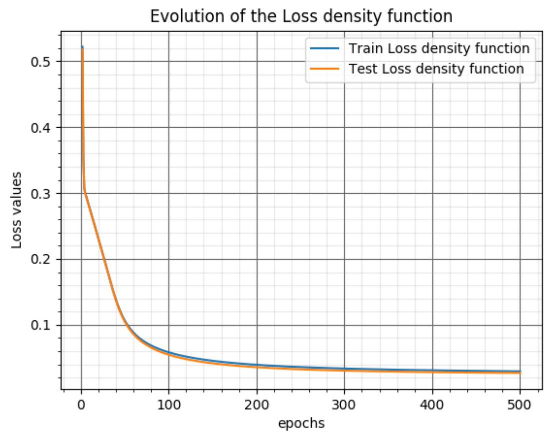
Fig. 2: Loss density function

B. *A Neural Network with one Hidden Layer:*

When we add a hidden layer, the accuracy (Figure (3a)) reaches a higher value, up to 0.9913, and the loss density function reaches a lower value (Figure (3b)), being this last one 0.02696. We can conclude that this hidden layer allows a better learning for the neural network, as the final values for the accuracy and the loss density function are better than in the case in which we didn't use any hidden layer.



(a) Accuracy for 500 epochs

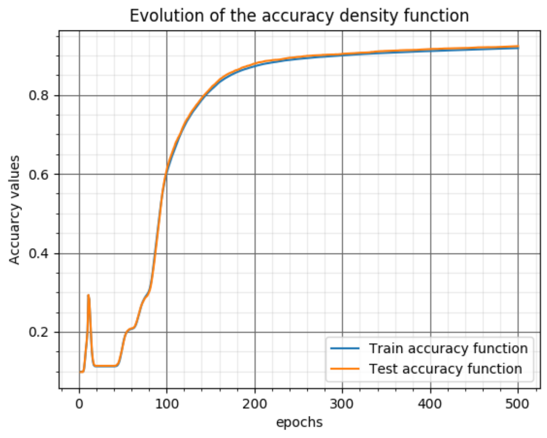


(b) Loss density function for 500 epochs

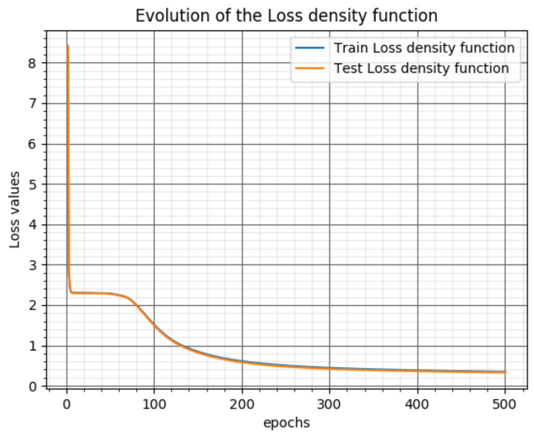
Fig. 3: Loss density function and accuracy for single neural network with a hidden layer

C. *Multi-class Neural Network:*

In this case, the accuracy is also incremented with the number of epochs, but as the output is more complex, the accuracy is a bit inferior to the one obtained in the previous cases, being its final value 0.9238, as shown in Figure (4a). The loss function also has a higher final value, being it 0.3298, as shown in Figure (4b) for only 500 epochs.



(a) Accuracy for 500 epochs



(b) Loss density function for 500 epochs

Fig. 4: Loss density function and accuracy for multiclass neural network