# Deep Learning Mini-Project 2 Report
# Implementing from Scratch a Mini Deep-Learning Framework.

Group 78*
Caterina Bigoni and Nicolò Ripamonti

18 May 2018

## 1 Introduction

The goal of this project consists in designing a mini deep learning network frameworks "by scratch". We have therefore built a network that combines fully connected layers, Tanh, the logistic sigmoid and ReLu. Moreover, we have implemented a forward and backward pass as well as the update of parameters using the stochastic gradient descent (SGD). For this part we followed the mathematical framework described in the lecture notes [1] and in [2].

This report is organised as follows: in Section 2 we describe the generalities of the implementation, explaining in particular the concept of dictionaries used to define the operators and their connectivity map. Then, in Section 3, we test the constructed network on a two-class classification problem both for a linearly separable dataset and a non-linearly separable one. Satisfactory results, i.e. from 0.5 to 2% error, are achieved in both cases. Conclusions and further remarks are presented in Section 4.

## 2 Code

The library handles the network structure by means of dictionaries, the ideal type to represent graphs in Python language . The generic dictionary considered for this project can be written as

$$\begin{aligned}\text{Dictionary} = \{\ &\text{key}_1\ : [\ \text{value}_{1,1},\ \text{value}_{1,2}, \dots],\\ &\text{key}_2\ : [\ \text{value}_{2,1},\ \text{value}_{2,2}, \dots],\\ &\dots\\ &\text{key}_N\ : [\ \text{value}_{N,1},\ \text{value}_{N,2}, \dots]\},\end{aligned}$$

where for each key we may have an arbitrary (and possibly different) number of values. We use the list constructor [...] to gather all the values related to a certain key because lists are containers easily iterable and we need this feature as explained later on.
The first dictionary that the user has to define is called `operators` in `test.py` and it records the modules used in the network. For example, let us consider a simple network, as the one represented in Figure 1. Its related dictionary is

$$\begin{aligned}\text{operators} = \{&1\ : [\ \text{Linear1}\ ],\\ &2\ : [\ \text{Nonlinear1}\ ],\\ &3\ : [\ \text{Linear2}\ ],\\ &4\ : [\ \text{Nonlinear2}\ ],\\ &5\ : [\ \text{Sum}\ ]\}.\end{aligned}$$

*As agreed with Dr. F. Fleuret, L. Pegolotti has collaborated with group 78 for Project 1 and with group 79, with M. Martin, for Project 2. C. Bigoni and N. Ripamonti have worked together on both projects.

The next step is the definition of a `connectivity` dictionary: from this variable, an object of the class `Network`, will generate two additional dictionaries, used to introduce and ordering in the forward pass. Since more than

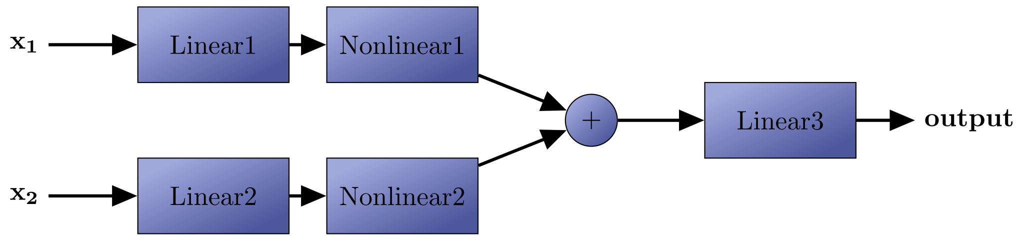

Figure 1: Example of network that can be represented using our library.

### 2.1 General Network

### 2.2 Modules

With the term *module* we mean each possible element of the network from linear or nonlinear layers to the loss function. Moreover, other operators such as the sum of two different inputs are treated as modules, and hence we obtain an easier representation for more complex networks 3 . The general structure of a module object is given in `ModuleBase.py` and in each derived object we have to define the following methods:

- `forward`: Implement the forward pass given the input of the module and store both the input and the output as attributes of the class.
- `backward`: Compute the gradient of the loss with respect to the input given the gradient of the loss with respect to the output. If the module contains parameters, it also computes the gradient of the loss with respect of them.
- `update_param`: Given a learning rate as input, updates the parameters of the module using a gradient descent based approach.

#### 2.2.1 Linear Layer

## 3 Numerical Experiments

Before running the model presented in the previous section, we run a simple "sanity check" test to verify if our setup behaves correctly. We first consider a simpler classification problem: a two-class problem where the discriminant line that separates points classified with label 0 or 1 is linear. In particular, we sample points from a standard Normal distribution and assign them a soft label 0.8 or 0.2 if they lie on the left or on the right side of a line, respectively. The line is fixed with intercept in the origin and slope equal 4. To train this problem, we use a very simple network where the operators and connectivity map are chosen as follows:

$$\text{operators} = \{1 \ : \ \text{Linear} \ \}, \quad \text{connectivity} = \{1 \ : [\,]\}$$

We also need to specify the input operator as [1] and the output operator equal to 1. In this model the first value  Linear  has dimensions number of classes × number of inputs. As shown in Figure 2, even a very simple network of this kind can provide satisfactory results on unseen data: 0.3% error on the training dataset and 0.5% error on the test set.

For this problem and all the following once we sampled 1000 points in the train set and 200 points in test set. The learning rate is set equal to 0.05 for all problems and we run it for 1000 epochs.

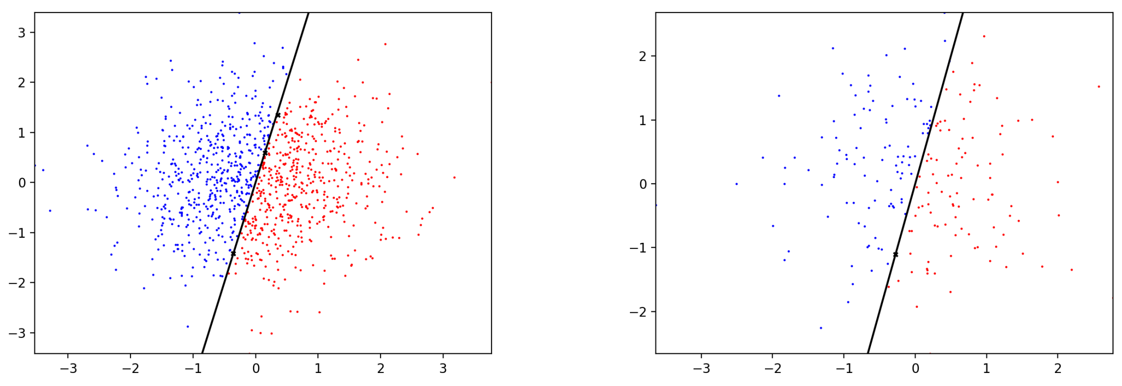

Figure 2: Training (left) and test (right) sets for the first simple model. The discriminant between the two classes is a line, points belonging to class 0 and 1 are marked in blue and red, respectively. Points misclassified are marked with a black x symbol.

As one could have expected, this very simple linear model does not generalise to more complex classification problems, where there discriminant plane is non linear, as for example the dataset required for this project. After randomly generating points in $[0,1]^2$ (sampled from a uniform distribution) with label 0 if they lie inside the circle of radius $\frac{1}{\sqrt{2\pi}}$ or 1 otherwise, we test the classifier using the simple linear model presented above. Figure 3 shows the results on an unseen test set. All the points close to circle, the discriminant border, are misclassified (black cross) leading to an error of 10.2% in the training set and 11% in the test set.

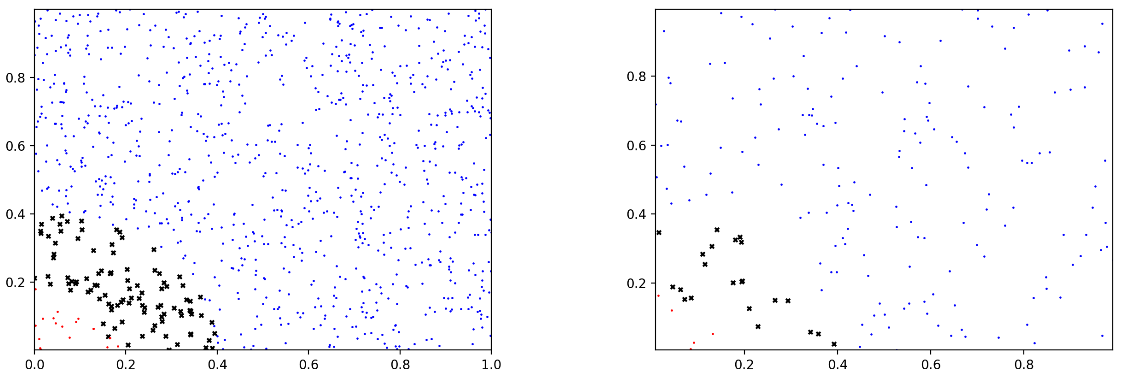

Figure 3: Training (left) and test (right) sets for the first simple model on the requested data. The discriminant between the two classes is a circle, points belonging to class 0 and 1 are marked in blue and red, respectively. Points misclassified are marked with a black x symbol.

It seems evident that for this setup a network with at least logistic sigmoids or Tanh is needed to learn the non-linearities. After this preliminary set, we are ready to test the performance of the model described in Section 2: Figure 4 shows the results on an unseen test set. Here we obtain 0.5% error on the training set and 0% error on the test set. These results are very satisfying and confirm the well behaviour of the model we implemented.

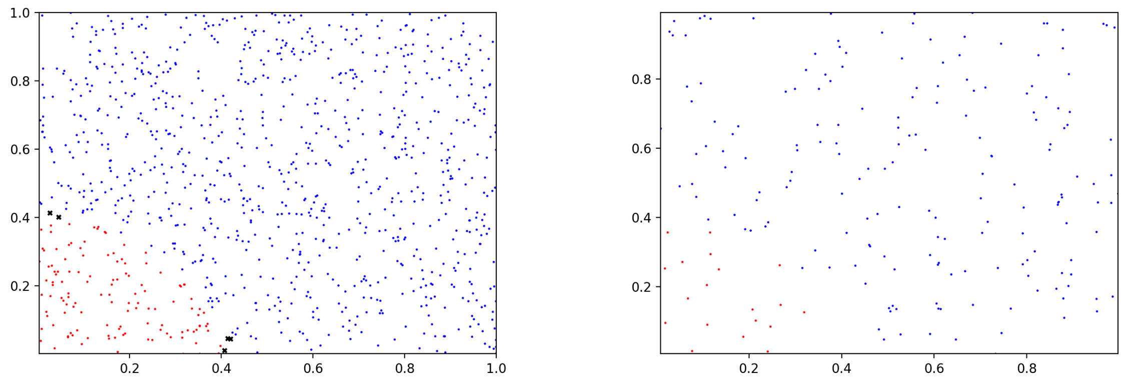

Figure 4: Training (left) and test (right) sets for the non-linear discriminant problem using the model described in Section 2. Points belonging to class 0 and 1 are marked in blue and red, respectively and points which are misclassified are marked with a black x symbol.

## 4 Conclusion

## References

[1] Deep Learning Lecture Notes by Franois Fleuret (Course EE-559). `https://documents.epfl.ch/users/f/fl/fleuret/www/dlc/`.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.