

UNIVERSITÉ DE GENÈVE

DEEPLARNING

---

## Project 2

---

*Student:*

Fardin HOSSAIN

Salma ENNAJ

Yacine M'RAD

*Teacher:*

Pr. François FLEURET

*Teaching Assistants:*

Bálint MÁTÉ

Atul SINHA

December 18, 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Program Architecture</b>	<b>2</b>
2.1	Implemented modules . . . . .	2
2.1.1	Module Linear . . . . .	2
2.1.2	Module Tanh . . . . .	2
2.1.3	Module Sigmoid . . . . .	3
2.1.4	Module ReLU . . . . .	3
2.2	Main Module . . . . .	3
2.3	Sequential . . . . .	3
<b>3</b>	<b>Results</b>	<b>4</b>
<b>4</b>	<b>Conclusion</b>	<b>4</b>

# 1 Introduction

The goal of this project is to implement a complete set of tools from scratch and perform a gradient descent in order to classify data on a small set of test data. The generated data consists of points of class 1 or 0 separated by a circle centered in 0.5,0.5 and radius  $1/\sqrt{2 \cdot \pi}$ . The class is 0 for points outside and 1 otherwise. This project is heavily inspired from Practical Work n3 where we had to manually implement a single fully connected layer. The difference here is that the code has been made more modular allowing to change the activation functions, or add layers by simply manipulating "boxes".

## 2 Program Architecture

### 2.1 Implemented modules

The different modules needed in order to perform are relatively small, and consists of a Linear layer, ReLU and TanH.

Each of the different modules had the following methods implemented, no matter if it was needed for something or not.

---

```
1 class MODULE_NAME():
2     def __init__(self) -> None: """when module is created"""
3     def __call__(self, x: torch.Tensor) -> torch.Tensor: """forward pass"""
4     def backward(self, dl_ds: torch.Tensor) -> torch.Tensor: """backward pass"""
5     def param(self) -> List: """get the parameters"""
6     def zero_grad(self) -> None: """re-initialize gradients"""
7     def update(self, *args): """update parameters from gradient"""
```

---

#### 2.1.1 Module Linear

This module allows to compute the forward and backward pass over a single example.

The forward pass computes the output  $s^{(l)}$  from the input  $x^{(l-1)}$  with the equation.

$$s^{(l)} = w^{(l)}x^{(l-1)} + b^{(l)}$$

The backward pass computes the derivative of weights and bias from the output with:

$$\left[ \frac{\partial l}{\partial w^{(l)}} \right] = \left[ \frac{\partial l}{\partial s^{(l)}} \right] (x^{(l-1)})^T, \quad \left[ \frac{\partial l}{\partial b^{(l)}} \right] = \left[ \frac{\partial l}{\partial s^{(l)}} \right] \quad \text{and} \quad \left[ \frac{\partial l}{\partial s^{(l+1)}} \right] = \left[ \frac{\partial l}{\partial s^{(l)}} \right] \cdot w^{(l)}$$

#### 2.1.2 Module Tanh

The forward pass is implemented with the following:  $s^{(l)} = \frac{\exp(x^{(l)}) - \exp(-x^{(l)})}{\exp(x^{(l)}) + \exp(-x^{(l)})}$

And the backward pass:  $\left[ \frac{\partial l}{\partial s^{(l+1)}} \right] = (1 - \tanh(s)^2) \cdot \left[ \frac{\partial l}{\partial s^{(l)}} \right]$

### 2.1.3 Module Sigmoid

The forward pass is implemented with the following:  $s^{(l)} = \frac{1}{1 + \exp(-x^{(l)})}$

And the backward pass:  $\left[ \frac{\partial l}{\partial s^{(l+1)}} \right] = \text{sigmoid}(s) \cdot (1 - \text{sigmoid}(s)) \cdot \left[ \frac{\partial l}{\partial s^{(l)}} \right]$

### 2.1.4 Module ReLU

The forward pass is implemented with the following:  $s^{(l)} = \begin{cases} 0 & \text{if } x^{(l)} < 0 \\ x^{(l)} & \text{otherwise} \end{cases}$

And the backward pass is:  $\left[ \frac{\partial l}{\partial s^{(l+1)}} \right] = \left[ \frac{\partial l}{\partial s^{(l)}} \right]$  if  $x^{(l)} > 0$  0 otherwise

## 2.2 Main Module

## 2.3 Sequential

The sequential method allows to set all the network's layers, and store them in a list:

`model.Sequential(Linear(2,25), ReLU(), Linear(25,25), ReLU(), Linear(25,1), Tanh())` means that the figure 1 is created.

Then each module can be called in the class in the following way:

---

```

1 for cur_module in self.operations_l:
2     x = cur_module(x) #forward pass
3 for cur_module in self.operations_l[::-1]:
4     x = cur_module.backward(x) #backward pass

```

---

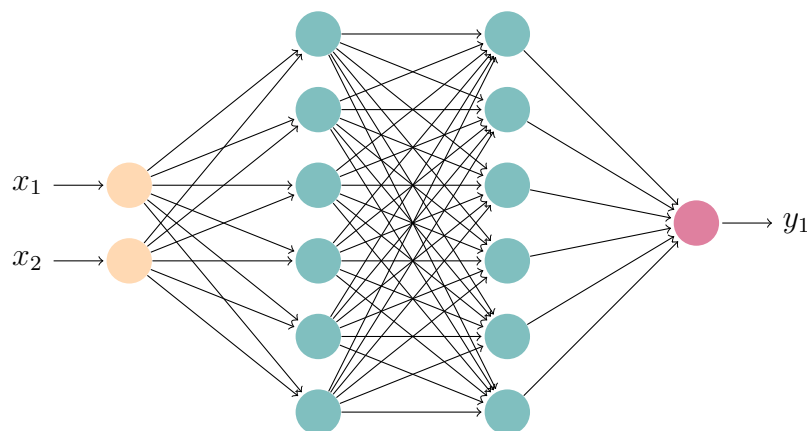


Figure 1: Created layer by sequential with two hidden layers and 25 units

### 3 Results

Figures 2 and 3 show that the error rate and loss decrease as we want and the figure 4 shows that target class is correctly predicted after 10 training epochs.

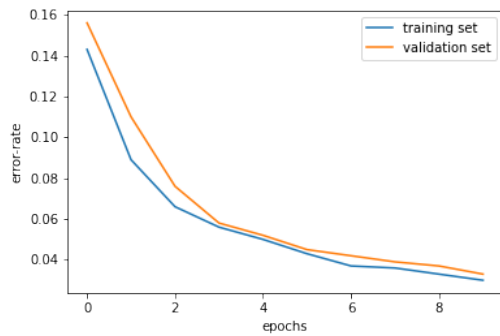


Figure 2: Error-rate through the epochs

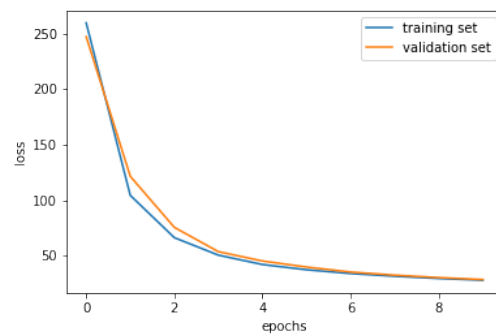


Figure 3: Loss through epochs

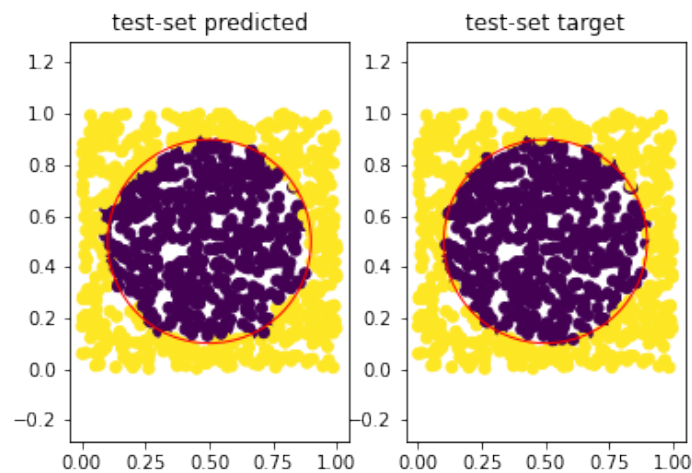


Figure 4: Visual verification of Predicted vs Target class

### 4 Conclusion

This working project allowed us to learn how PyTorch works and implement a framework on our own. This allowed to deepen our understanding of the Deep-Learning course.