Machine Learning per il Calcolo Scientifico

First laboratory exercises

General guide

For each laboratory, an incomplete Python notebook will be provided with exercises (steps) that must be completed in the given order (some of the exercises will be needed in future laboratories). In step zero, all the Python packages that are needed in order to complete the notebook are listed. This PDF includes the text for the exercises and the expected outcomes for each step. While following the notebook is recommended, you are also welcome to attempt the exercises without using it.

Step Zero

Here are the required Python (`https://www.python.org/`) packages for this laboratory:

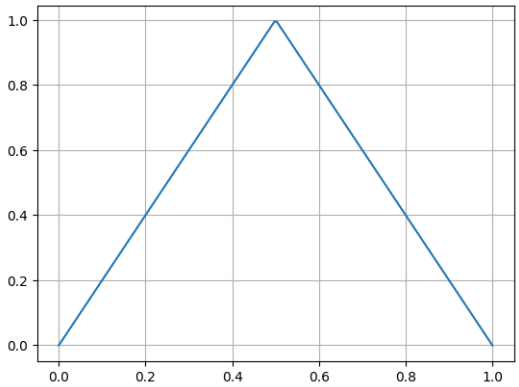
- PyTorch (`https://pytorch.org/`)
- Numpy (`https://numpy.org/`)
- Matplotlib (`https://matplotlib.org/`)

First Step: Define $\Phi^\wedge$ using torch Tensors

a. Using torch.tensor, define the weight and biases for the $\Phi^\wedge$ Neural Network (NN).
b. Using the definition seen during the lecture construct the $\Phi^\wedge$ NN with ReLu activation that emulates F_1.

$$F_1 : x \mapsto \begin{cases} 2x & x \in (0, 1/2] \\ 2 - 2x & x \in (1/2, 1) \\ 0 & \text{elsewhere.} \end{cases}$$

c. Plot the $\Phi^\wedge$ NN in [0,1].



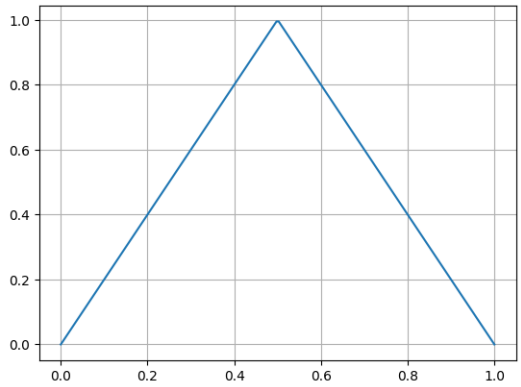
Second step: Define a general Feed-Forward Neural Network (FNN)

a. Complete the Python class DeepNet that defines a general FNN in Pytorch. (Will be used in other labs)

b. Using the DeepNet class, create a FNN with 1 layer 3 neurons with ReLu as an activation function.

```
DeepNet(
  (hidden): ModuleList(
    (0): Linear(in_features=1, out_features=3, bias=True)
  )
  (output): Linear(in_features=3, out_features=1, bias=False)
)
```

c. Redefine the $\Phi^\wedge$ NN by loading the weights and biases defined in the first step point a (set_weight set_bias set_output_weight) into the FNN. Then plot the output of the NN in [0,1].



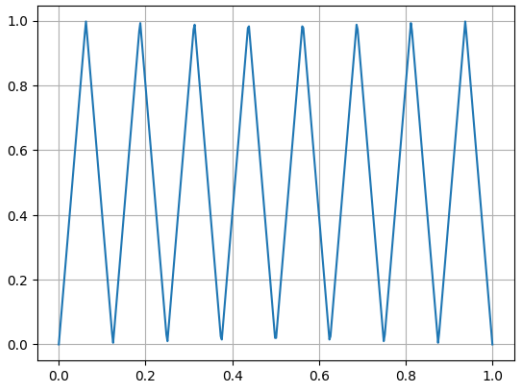
Third step: Concatenation of two Networks

a. Complete the Python function concatenate, which implements the non-sparse concatenation.

b. Concatenate 4 times the $\Phi^\wedge$ NN defined before (second step point c).

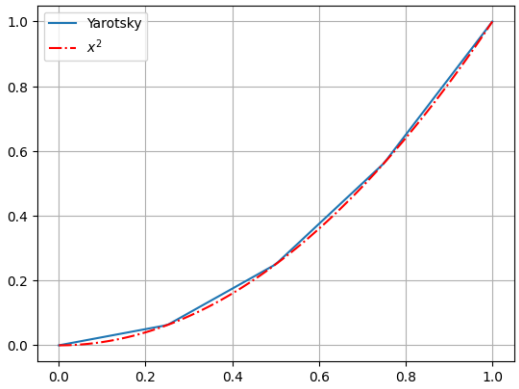
```
DeepNet(
  (hidden): ModuleList(
    (0): Linear(in_features=1, out_features=3, bias=True)
    (1-3): 3 x Linear(in_features=3, out_features=3, bias=True)
  )
  (output): Linear(in_features=3, out_features=1, bias=False)
)
```

c. Plot the NN obtained in the previous point in [0,1].

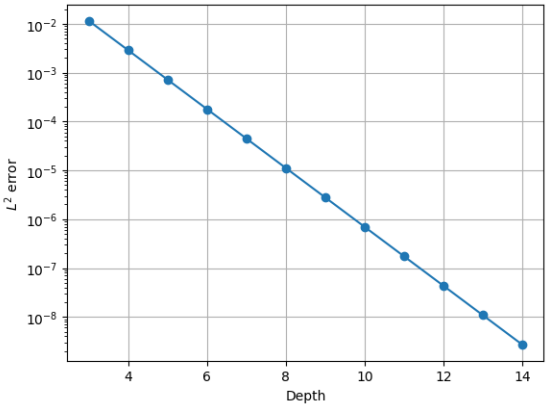


Fourth step: Define the Yarotsky Network

a. Complete the class YarotskyNet, then using this class define a Yarotsky NN with depth = 3.
b. Plot the output of the Yarotsky Network over [0,1] and compare it to x^2.



c. Evaluate the L^2 error of the Yarotsky NN with respect to x^2 ($\approx 1.13 \times 10^{-2}$).
d. Evalute the L^2 error of the Yarotsky Network with respect to x^2, as a function of the depth of the network.


10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
L^2 error
4
6
8
10
12
14
Depth