Assignment Description

In this assignment you will implement neural networks for regression. We want to reconstruct in the region $[-2;2] \times [-3;3]$ a two dimensional function $F : \mathbb{R}^2 \to \mathbb{R}$ which picture is sent in a separate file. You do not have the analytic expression of the function but only a data set obtained by randomly sampling 250 points x^p and adding to the value function $F(x^p)$ a random noise in the range $[-10^{-5}, 10^{-5}]$ so that the data set is

$$\{(x^p; y^p) : x^p \in \mathbb{R}^2; y^p \in \mathbb{R}; p = 1, \ldots, 250\}$$

You will need to provide a training error and a validation error. Please note that the test error will be evaluated on the BLIND TEST SET that will be provided by the teacher AFTER the submission of projects by all teams.

In the exercise where you have to define a starting point for the optimisation procedure, we ask you to chose it randomly by fixing a seed, so that running the code multiple times will always produce the same result. You can test different seeds, which must be described in the report, but you must fix the seed in the code to the value that gave you the best results.

If you are using other functions which use random seed, this seed must be fixed and reported in the code clearly.

Question 1. (Full minimization)

You must construct a shallow Feedforward Neural Network (FNN) (one only hidden layer), both a MLP and a RBF network, that provides the model $f(x)$ which approximates the true function F. We denote by π the hyper-parameters of the network to be settled by means of an heuristic procedure and ω the parameters to be settled by minimizing the regularized training error

$$E(\omega; \pi) = \frac{1}{2P} \sum_{p=1}^{P} \left(f(x^p) - y^p \right)^2 + \frac{\rho}{2} \|\omega\|^2 \tag{1}$$

where the hyper parameter ρ stays in the range $[10^{-5} \div 10^{-3}]$.

1. (max score up to "20" (mandatory)) Construct a shallow MLP with a linear output unit, namely

   $$f(x) = \sum_{j=1}^{N} v_j g\left(\sum_{i=1}^{n} w_{ji} x_i + b_j \right)$$

   where $\omega = (v, w, b)$. The activation function $g(\cdot)$ is the *hyperbolic tangent*

   $$g(t; \sigma) := \tanh(t; \sigma) = \frac{e^{2\sigma t} - 1}{e^{2\sigma t} + 1} \tag{2}$$

   (with σ hyperparameter; g is available in Python with $\sigma = 1$: Numpy.tanh)

   The hyper-parameters are

   - the number of neurons N of the hidden layer

- the spread σ in the activation function g
- the regularization parameter ρ

It must be possible to specify N, σ and ρ as inputs parameters of your training code.

Write a program which implements the regularized training error function $E(v,w,b)$ (as obtained by (1) using $f(x)$ of the MLP network) and uses a Python routine of the optimization toolbox (scipy.optimize) to determine the parameters v_j, w_{ji}, b_j which minimize it. The code must produce a plot of the approximating function found.

2. (max score "+5" points) Construct a RBF network

$$f(x) = \sum_{j=1}^{N} v_j \phi(\|x^i - c_j\|)$$

where $\omega = (v,c)$ $v \in \mathbb{R}^N$ and $c_j \in \mathbb{R}^2$ $j = 1,\dots,N$.

You must choose as RBF function $\phi(\cdot)$ the Gaussian function

$$\phi(\|x - c_j\|) = e^{-(\|x-c_j\|/\sigma)^2} \quad \sigma > 0 \tag{3}$$

The hyper-parameters are

- the number of neurons N of the hidden layer
- the spread $\sigma > 0$ in the RBF function ϕ
- the regularization parameter ρ

It must be possible to specify N, σ and ρ as inputs parameters of your training code.

Write a program which implements the regularized training error function of the RBF network $E(v,c)$ (as obtained by (1) using $f(x)$ of the RBF network) and uses a Python routine of the optimization toolbox for its minimization with respect to both (v,c). The code must produce a plot of the approximating function found.

Please note that in Q1.1. and Q1.2 it is not required to evaluate the gradient so derivative-free optimization routine may be applied or finite difference evaluation of the gradient can be used. Nevertheless using gradient-based method will improve performance and boost up your code. Explicit evaluation of the exact gradient will allow to obtain a greater score.

Question 2. (Two blocks methods)

1. (max score up to "+2" points) Consider again the shallow MLP with linear output unit as defined at Ex. 1 of Question 1. Use the values of N, ρ, σ as you have fixed at Ex. 1. of Question 1.

Write a program which implements an *Extreme Learning* procedure, namely one that fix randomly the values of w_{ij}, b_j for all i, j and uses the "best" Python routine or any other optimization library to minimize the regularized quadratic convex training error $E(v)$ of the only variables $v \in \mathbb{R}^N$.

As for the random generation of w, b you have no restrictions on the way of proceeding. We suggest to identify a range and repeat the random choice more than once. In the run file, the w, b still need to be defined in a random-like way, definitions "by hand" will not be accepted.

2. (max score up to "+2" points) Consider again an RBF network as in Ex. 2 of Question 1.

   The number of RBF units N of the hidden layer, the positive parameter σ in the RBF function and ρ are those chosen at Ex 2. Question 1.

   Write a program which implements a method with unsupervised selection of the centers. You can select the centers by randomly picking N of the P points of the training set. We suggest to repeat the random choice more than once.

   Find the optimal weights by minimizing the regularized error $E(v)$ using a suitable Python routine or any other optimization library to minimize the quadratic convex function.

Question 3. (two-block decomposition method) (up to "+2" points)

Consider either the shallow MLP or the shallow RBF network. Set the hyper-parameters parameter at the values you selected in Question 1.

Write a program which implements a two block decomposition method, which alternates the convex minimization with respect to the output weights v and the non convex minimization with respect to the other parameters ((w, b) for the MLP and the centers c for the RBF network, respectively).

You must use appropriate Python routines of the optimization toolbox for solving the two block minimization problems. In this case, it is required that you calculate the gradient of the non-convex block. Consider adjusting the tolerances of the optimizers as the number of iterations increases.

1 Instruction for the report

In the report, for both Q_{11} and Q_{12} it is MANDATORY to report:

- the grid-search intervals, as well as the K-Fold procedure settings to tune the hyperparameters;
- analyse the occurrence of overfitting/underfitting varying the number of neurons N and the parameters ρ and σ;
- the final setting for N, ρ and σ; how did you choose them and if you can put in evidence over/underfitting and if you can explain why;

- **1.** starting/final values of the objective function (1), **2.** which optimization routine did you use for solving the minimization problem, **3.** the setting of its parameters (tolerance, max number of iterations etc) **4.** the returned message in output (successful optimization or others),**5.** number of iterations, **6.** number of function/gradient evaluations, **7.** computational time, **8.** whether you have implemented the gradient or use finite difference routine.

- the final values of the training error and the validation error. (NB: the training and validation errors are computed using the formula (1) with $\rho = 0$ (namely without the regularization term). This holds for all the exercises.)

- the plot of the function representing the approximating function obtained by the MLP and by the RBF networks;

- a comparison of performance between MLP and RBF networks both in terms of quality of approximation (training and validation error) and in efficiency of the optimization (number of function/gradient evaluation and computational time needed to get the solution). Please put these values in a table as explained at the end.

For $Q_{2.i}$ it is MANDATORY to report for each part $i = 1, 2,$:

- which optimization routine you use for solving the quadratic minimization problem and the setting of its parameters. Compare the performance of the optimization process w.r.t Full optimization of Question 1 (Ex. i);

- the values of the training error and the validation error; Compare these values with the ones obtained by the Full optimization of Question 1 (Ex. i);

- starting/final values of the objective function (1);

- the plot of the function representing the approximating function obtained;

for Q_3 it is MANDATORY to report:

- which optimization routines of the Optimization toolbox you used to solve the two block sub-problems and the setting of the parameters;

- the stopping criteria of the decomposition procedure; consider the use of early stopping rules;

- starting/final values of the objective function (1);

- the values of the final training error and validation error;

- the plot of the function representing the approximating function obtained by the neural model (either RBF or MLP) with block decomposition in comparison with the true one;

- comparison with the corresponding network obtained by random selection at the corresponding Exercise of Question 2. Comparison are both in the quality of the network (validation error), and in the efficiency of the optimization procedure (number of function/gradient evaluation and computational time needed to get the solution). Please put these values in a table.

- compare the results with those obtained at the corresponding exercise of Question 1.

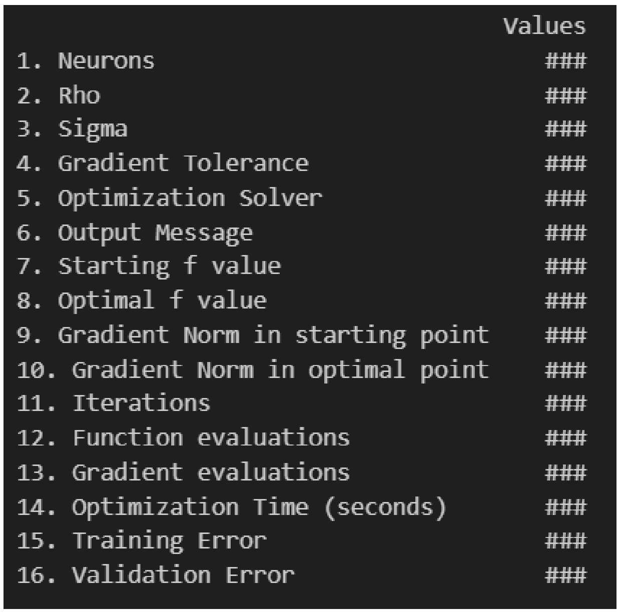
In the final report (pdf) it is also MANDATORY to gather into a final table (an example below) the comparison among all the implemented methods - in term of accuracy in learning and computational effort in training.

		settings N σ ρ	Initial objective function	Final objective function	Final train error	Final validation error	opt. time
Q1.1	Full MLP						
Q1.2	Full RBF						
Q2.1	Extreme MLP	as in Q1					
Q2.2	Unsupervised c RBF	as in Q1					
Q3	Block Decomp	as in Q1					

2 Instructions for Python code

You are allowed to organize the code as you prefer **BUT** for each question $ij = \{11,12,21,22,3\}$ you have to create a different folder named **Question_ij_GroupName.py** where **you must provide the files according to the following instructions.**

- In the folder you submit, you must include the dataset.csv file, which will be used by your scripts to complete the tasks.
- For each question you have to provide a file called **functions_ij_GroupName.py**. This file should only include all the functions and libraries you used for solving the specific question ij.
- A file called **run_ij_GroupName.py** (e.g. run_11_Example.py, run_12_Example.py). This file will be the only one executed during the verification phase of the work. Additionally, the code must save the plot of the function in the specified folder with the name **img_ij_GroupName.png**, but **it should not display it on the screen**. It should include the import of all the necessary functions, but it has to print ONLY:
  1. Number of neurons N chosen
  2. Value of ρ chosen
  3. Value of σ chosen
  4. Values of other hyperparameters (if any)
  5. Optimization solver chosen (e.g. L-BFGS, CG,NTC, Nelder-Mead....)
  6. The returned message in output (successful optimization or others)
  7. Function value in the starting point
  8. Function value in the optimal point
  9. Gradient norm value in the starting point
  10. Gradient norm value in the optimal point
  11. Number of iterations
  12. Number of function evaluations
  13. Number of gradient evaluations
  14. Time for optimizing the network (from when the solver is called, until it stops)
  15. Training Error
  16. Validation Error



```
                                          Values
1. Neurons                                   ###
2. Rho                                       ###
3. Sigma                                     ###
4. Gradient Tolerance                        ###
5. Optimization Solver                       ###
6. Output Message                            ###
7. Starting f value                          ###
8. Optimal f value                           ###
9. Gradient Norm in starting point           ###
10. Gradient Norm in optimal point           ###
11. Iterations                               ###
12. Function evaluations                     ###
13. Gradient evaluations                     ###
14. Optimization Time (seconds)              ###
15. Training Error                           ###
16. Validation Error                         ###
```

Figure 1: Example of the expected output of file run_ij_GroupName.py

Note that:

1. Each question will be graded averaging the Performance score (25%), the Efficiency score (25%) and the Mathematical Correctness score (50%).

2. There is no competition with other students. Your grade does not depend on other teams' performances. However, the team that will obtain the best absolute value on the **blind** test error will receive an "award" from the teacher (not points in the grade).

3. You are warmly recommended to write clean and easily readable code. Not respecting the general assignment (for example, printing different values or not printing the requested outputs) will result in a penalization. Printing the progress of the algorithm or similar mistakes will slow down the code correction and result in a score decrease. Additionally, please make sure to comment your code sufficiently, so that an external reader can understand what each part of the code does.

4. Your code will not be debugged and will not be modified **in ANY case** after the submission. **If the code does not run, the resulting score will be zero**. You can schedule a meeting to get some help in debugging with teacher and/or teaching assistant **BEFORE the submission, not after**. You are warmly recommended to check multiple times and on different computers that your code is executable.

5. It is welcomed to take inspiration from other students, as well as from open-source contents online, but **"copy and paste" strategies will invalidate your project**. Be aware that **your code will be cross-checked with online materials, as well as with other students' code and with AI-generated code**.

6. The use of any library automatically building neural models (e.g, Pytorch, TensorFlow, sklearn.neuralnetwork etc.) is **FORBIDDEN** and automatically invalidates the project.

7. Inside the functions_ij_GroupName.py file, the grid search implementation should be included but commented out when submitting the project. The grid search must not execute, but its implementation should be visible for review.

8. In case some of the required values to print are not available (for instance you are using a particular library or you are using a non-Python source code or any other valid motivation), just print e.g. ”function evaluations = Not Available”; ”output message = Not Available”.