Project 1 OMML

Group **AWS**:
Buzzichini Davide 1895533
Mazza Manuel 1819451

Question 1.1: MLP

The selection of the hyperparameters (N, rho, sigma) has been done through the grid search method, implementing a 5-fold cross validation that tested values of N in the range [1 ; 120], rho in the range [10-5 ; 10-3] and sigma in [0.1 ; 2.5].
Its execution returned the combination of N, rho and sigma that minimizes the mean squared error on the validation set. In particular:

- *N (number of neurons)*: The errors' evolution with respect to N described in **[FIG 1.1A]** show how after N=43, the validation error varies negligibly. Knowing that, we ran a further "grid-search" on a smaller range of N (starting from N=40) and identified in N=50 as the best result. Even though increasing the number of neurons may result in an even lower error, the improvement would be so small that it wouldn't be worth the increased risk of overfitting. Underfitting is clearly absent, as it is evident only for smaller values of N (less than 20) and both errors result to be firmly low for our chosen N.
- *ρ (regularization constant)*: **[FIG 1.1B]** shows how increasing rho leads to higher values of errors, almost in an exponential way. As both errors diverge, we can say that higher values of ρ introduce underfitting in the model, whereas smaller values can increase the risk of overfitting (even though it's not the case in our range of interest, as both training and validation errors are significantly small). In fact, the grid search gave ρ=1e-05 as its optimal value.
- *σ (spread in the activation function)*: From the plot shown at **[FIG 1.1C]**, we can see how values of σ closer to 0 introduce underfitting inside the model. We get better results approaching σ=1, where both errors are minimized. Further increasing the value of sigma could result in overfitting, as the validation error starts to diverge from the training error. Accordingly, the grid search appointed σ=1 as the optimal value.

To minimize the regularization error, we exploited the *minimize()* function from the *scipy.optimize* python library, together with an ad-hoc gradient function to enhance the function's optimization performances.
In such minimization, we also defined other key parameters for our optimization: function and gradient tolerance. We found 1e-7 to be our value for both, as it gives us our preferred tradeoff between execution time (sub 5 seconds) and error values (in the order of 1e-5). Increasing tolerances results in lower time but higher errors, whereas decreasing them achieves the exact opposite.

The plot is shown in **[FIG 1.1D]** while the output table can be seen at **[TAB 1.1]**.

NOTE: Our network has been trained on the whole dataset, so it's expected to have the validation error (computed though Kfold cross validation) to be exactly equal to the training error due to the fact that we only minimize ones outside the cross validation procedure, as requested.
We acknowledged that our program actually behaved correctly in our grid search function(visible in the hyperparameter's graphs), as in it we minimized the function using a part of the set for each fold, and only then computing the validation error on the other part.

Question 1.2: RBF

As in Q1.1, we got the must suitable hyperparameters implementing a grid search function with 5-fold cross validation.
We tested up to 200 number of neurons(N), while for the range of rho and sigma we kept the same of Q1.1.
Once again, the best triple has been selected not only according to the mere output of the grid search, but also by looking at the attached graphs.

[FIG 1.2A] shows how errors fluctuate following N's variation. It's evident how up to N=30, the model would strongly suffer from underfitting, while it doesn't show significant signs of overfitting even up to N=200.
However, we were still cautious with the selection, choosing N=63 to balance the graph and the grid search output evidences.
[FIG1.2B] and **[FIG 1.2C]** reported below and highlight that rho and sigma have a similar influence on the error as in Q1.1, therefore we selected them as before: ρ=1e-5 and σ=1. The only difference to point out is the introduction of underfitting for values of sigma greater than 1, in contrast with the presence of overfitting that the same values introduced in Q1.1.

To be consistent with the results of Q1.1 and to be able to make a plausible comparison with MLP, we kept the tolerances unchanged, set to 1e-7.

The plot is shown in **[FIG 1.2D]** while the output table can be seen at **[TAB 1.2]**.

Comparison between MLP (Q1.1) and RBF(Q1.2)

We conducted a brief comparison concerning the performance of the two just presented methods, analyzing both the quality of approximation and the efficiency of optimization:

- **QUALITY:** Both methods approximate pretty well, returning an error which is at least in the order of 1e-4. The MLP slightly outperforms the RBF in terms of error, as also highlighted by the N graph of both, where also validation error is smaller in the MLP. Nonetheless both errors are optimally low.
- **EFFICIENCY:** We can say the same about efficiency, with both methods optimizing in about one second. This time is the RBF method to be slightly better, mostly due to its lower number of iterations and gradient/function evaluations.

Question 2.1: Extreme MLP

Extreme learning requires to initialize the input weights *W* and the biases *b* randomly, with the error function becoming a quadratic function in *v*, for which the minimization can simply be achieved by putting its derivative equal to 0.
To do so, we used the "*np.linalg.lstsq*" solver, as once we moved from *W* and *b* to *Q* and *c*, the output weights that minimize the error can simply be found by solving *Qv+c=0.*

With respect to the full MLP, in extreme learning the selection of the starting point is much more crucial due to the fact that we only perform one iteration to minimize the error. To solve this issue, we wrote and ran a very simple multistart algorithm to select the best possible seed in the range [0; 1e5].
The plot is shown in **[FIG 2.1A]** while the output table can be seen at **[TAB 2.1]**.

Comparison between FULL MLP (Q1.1) and EL MLP(Q2.1)

Extreme Learning proposes a much quicker execution at the expense of a less precise optimization. As the results indicate in **[TAB 1.1]** and **[TAB 2.1]**, extreme learning presents an optimization close to 0, which comes at the cost of having an error that is 2 orders of magnitude higher when compared with its Full MLP counterpart.

Such outcome highlights how extreme learning is more suitable for larger datasets due to its fast execution, while full MLP outperforms it precision wise when dealing with a smaller number of samples.

Question 2.2: Unsupervised RBF

Unsupervised RBF is a variation of full RBF in which centers are chosen randomly among the samples in the dataset.
As in Q2.1, we are dealing with a quadratic convex problem, whose minimization is achieved by solving a linear least squares problem in *v* through the numpy "*lstsq*" function.
The single iteration nature of the discussed method makes the starting point crucial in the optimization, so we ran a multistart algorithm also for this problem, dealing with seeds ranging from 0 to 1e5, and applying only the one who gave the lowest error values.

The plot is shown in **[FIG 2.2A]** while the output table can be seen at **[TAB 2.2]**.

Comparison between FULL RBF (Q1.2) and unsupervised RBF(Q2.2)

[TAB 1.2] and **[TAB 2.2]** show clearly the differences among full and unsupervised RBF, which are not too far off those we encountered when discussing about Q1.1 and Q2.1.
As a matter of fact, unsupervised RBF outclasses full RBF in terms of execution time, optimizing the error function in instants.

Faster convergence also results in a slightly higher error, with full RBF edging unsupervised RBF in terms of error, due to the fact that it optimizes on all parameters iteratively, even though this requires numerous function and gradient evaluations.

Question 3: Two block decomposition

The two block method solves the optimization problem by splitting it into two subproblems, which are optimized alternatively and iteratively until some conditions are met.
These subproblems are:

- Non-convex minimization, conducted using the output weights v, for which we used the already cited *lstsq* function from the *numpy.linalg* python library.
- Convex optimization, conducted through input weights *W* and biases *b.* Again, we handled such problem with the *minimize* function from the *scipy* library.

We choose to implement this method using **MLP**, so we also kept the same hyperparameter selection as requested.

We implemented the alternative optimization of the two subproblems into a cycle whose execution stops whenever the value of the gradient's norm falls under 1e-6 (indicating that we are close enough to a minimum point) or if the number of maximum cycle iterations is exceeded (that we set to 150).
Other than those, the cycle establishes the following early stopping procedures:

1. *Time limit*: set to 10 seconds
2. *Patience*: if the two block method doesn't provide a better error for *maxPatience* (set equal to 20) iterations, then stop the execution.

The plot is shown in **[FIG 3A]** while the output table can be seen at **[TAB 3]**.

Comparison between two block decomposition method (Q3) and previous techniques

This method is clearly the most advanced and complex of those met for this homework. Such complexity is evident especially in the running time, as it is the slowest of the analyzed method. However, this time the tradeoff between time and precision that we saw when comparing full and EL MLP is less evident, as being the slowest doesn't translate to being the most precise error wise.
We can still highlight how it performs better than EL MLP, but the cost in terms of time and complexity may not be worth it; let alone when noticing that full MLP performs better in both quality and efficiency.

The two block decomposition is still a reliable and efficient method, whose performances might have been held back by the small dimension of the dataset in exam. In fact, this technique's main advantage is that separating a complex problem into two subproblems makes the optimization easier to solve, which comes in very handy when dealing with larger datas. So This method is particularly useful when solving the full optimization problem directly is too computationally demanding, whereas it isn't really a viable option for smaller and simpler problems.

Figures and tables

Q1.1 Full MLP

```
Neurons = 50
Rho = 0.000010
Sigma = 1.000000
Function and gradient tolerance = 1e-7

Optimization method = L-BFGS-B
Optimization success = True
Function value in starting point = 88.694159
Function value in optimum point = 0.001025
Gradient norm at starting point = 77.852103
Gradient norm at optimum point = 0.003941

Number of iterations = 815
Number of function evaluations = 882
Number of gradient evaluations = 882
Time spent to optimize the function = 0.329001

Training error = 0.000050
Validation error = 0.000050
```

TAB 1.1

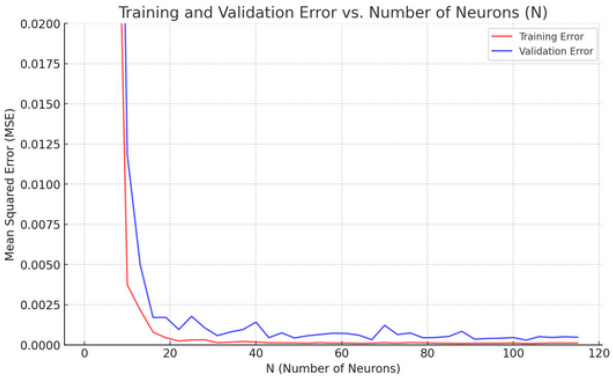


FIG 1.1A: Errors on number of neurons N

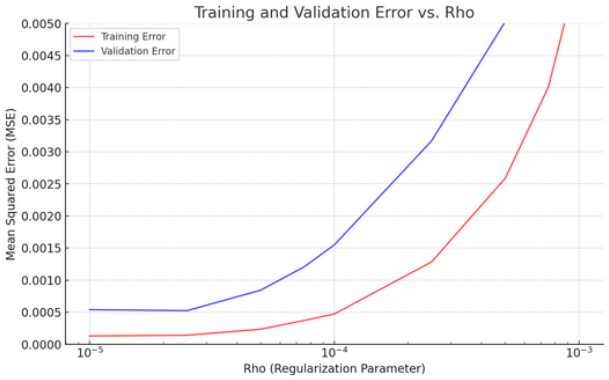


FIG 1.1B: Errors on regularization parameter ρ

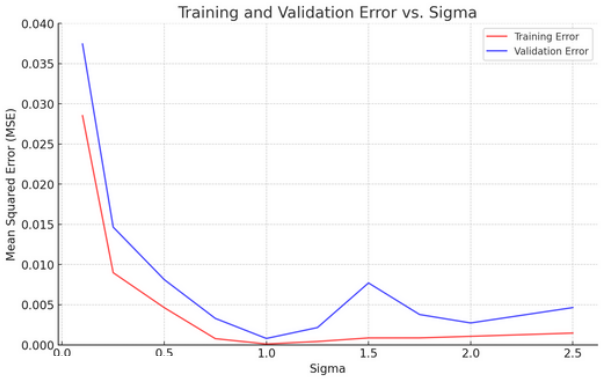


FIG 1.1C: Errors on g function's spread σ

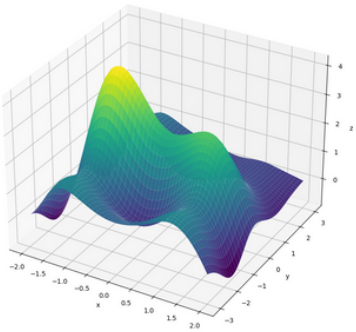
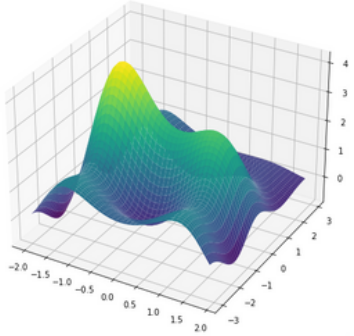


FIG 1.1D: Plot

vs



Original function's plot

Q1.2 Full RBF

```
Neurons = 63
Rho = 0.000010
Sigma = 1.000000
Function and gradient tolerance = 1e-7

Optimization method = L-BFGS-B
Optimization success = True
Function value at starting point = 1.682655
Function value in optimum point = 0.001436
Gradient norm at starting point = 2.013699
Gradient norm at optimum point = 0.000254

Number of iterations = 423
Number of obj. function evaluations = 434
Number of gradient evaluations = 434
Time spent to optimize the function = 0.327878

Training error = 0.000259
Validation error = 0.000259
```

TAB 1.2

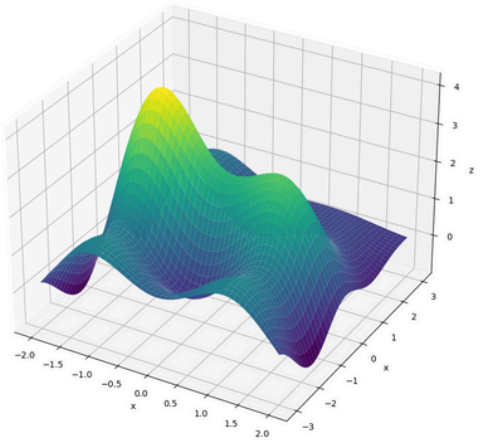


FIG 1.2D: Plot

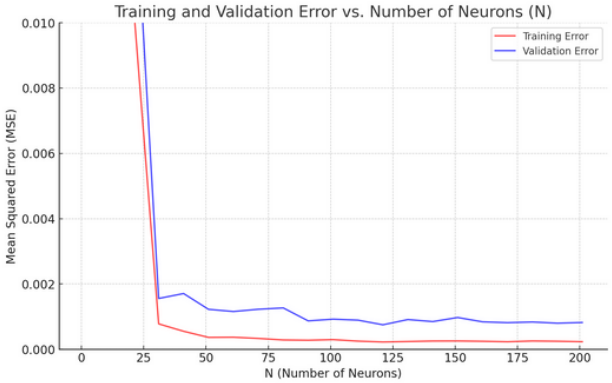


FIG 1.2A: Errors on number of neurons N

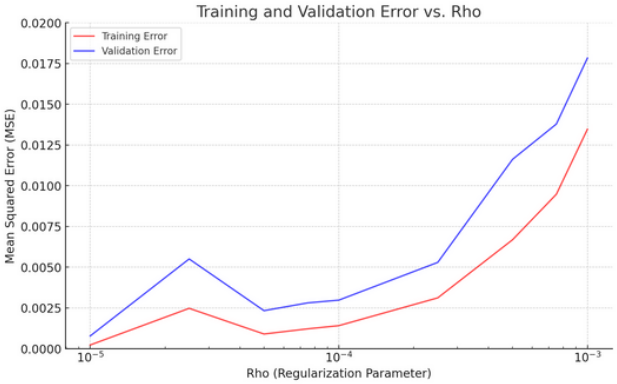


FIG 1.2B: Errors on regularization parameter ρ

FIG 1.1C: Errors on g function's spread σ

Q2.1: Extreme Learning MLP

```
Neurons = 50
Rho = 0.000010
Sigma = 1.000000
Optimum seed = 65363

Optimization method = Linear least squares
Optimization succes = True
Function value in starting point = 9.538326
Function value in optimum point = 0.002119

Number of iterations = 1
Number of function evaluations = 1
Time spent to optimize the function = 0.000000

Training error = 0.001226
Validation error = 0.001226
```

TAB 2.1

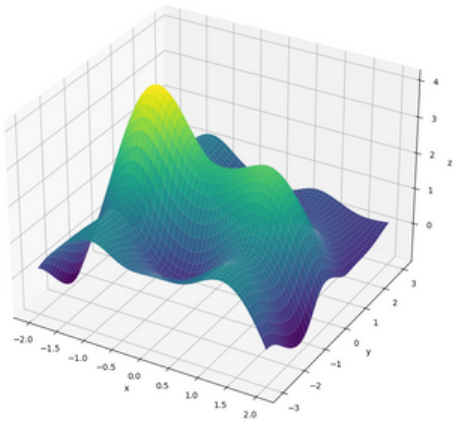


FIG2.1A: Plot

Q2.2: Unsupervised RBF

```
Neurons = 63
Rho = 0.000010
Sigma = 1
Best seed: 60236

Optimization method =Linear least squares
Optimization success = True
Function value at starting point = 2.142881
Function value in optimum point = 0.002730

Number of iterations = 1
Number of obj. function evaluations = 1
Time spent to optimize the function = 0.004027

Training error = 0.000821
Validation error = 0.000821
```

TAB 2.2

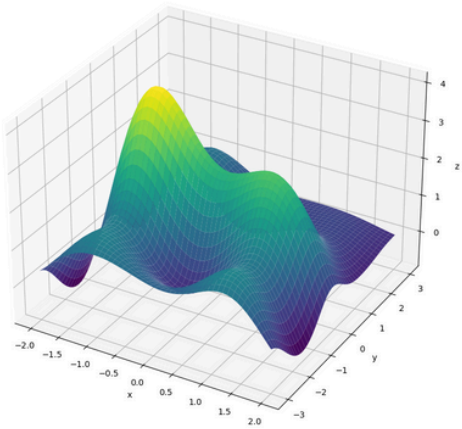


FIG2.2A: Plot

Q3: Two block decomposition method

```
-----------------------------------------------------------------
Neurons = 50
Rho = 0.000010
Sigma = 1.000000
Maximum number of cycle iterations = 150
Maximum patience parameter = 20
Initial gradient tolerance = 0.00000100
Tolerance reduced by 10 % each 10 iterations
Last gradient tolerance = 0.00000023

Optimization method (for input weights) = L-BFGS-B
Optimization method (for output weights) = LSTSQ (Numpy least square method)
Function value in starting point = 88.694159
Function value in optimum point = 0.003053

Number of cycle iterations = 150
Number of function evaluations = 1875
Number of gradient evaluations = 1575
Time spent to optimize the function = 1.254740

Training error = 0.000598
Validation error = 0.000598
-----------------------------------------------------------------
```

TAB 3

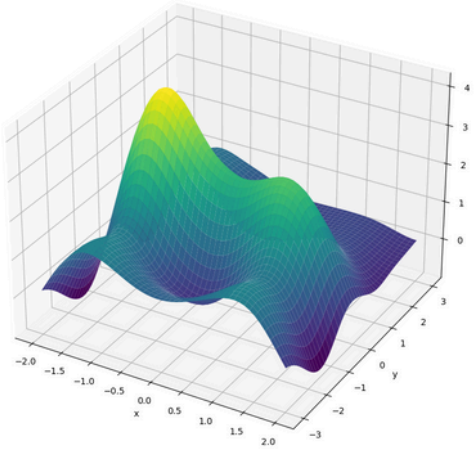


FIG3A: Plot

Summary table

		settings			Initial	Final	Final	Final	opt.
		N	σ	ρ	objective function	objective function	train error	validation error	time
Q1.1	Full MLP	50	1e-5	1	88,694159	0,001025	0,000050	0,000050	0,329
Q1.2	Full RBF	63	1e-5	1	1,682655	0,001436	0,000259	0,000259	0,328
Q2.1	Extreme MLP	as in Q1.1			9,538326	0,002119	0,001226	0,001226	0,0009
Q2.2	Unsupervised c RBF	as in Q1.2			2,142881	0,002730	0,000821	0,000821	0,004
Q3	Block Decomp	as in Q1.1			88,694159	0,003053	0,000598	0,000598	1,2547

NOTE: for the 2.n questions, the starting training error is computed by selecting randomly also the initial v value. Even though it wouldn't be expected when using such methods, we wanted to give an idea of the optimization achieved.