Project #1 - MLP and Generalized RBF Network

Group gli ottimizza-TORI
Francesco Elvio Buti, Luca Bigi, Matteo Martinovich

November 2024

Table 1: Summary of Results

Model	N	σ	ρ_1	ρ_2	Initial Obj. Function	Final Obj. Function	Final Train Error	Final Val Error	Opt. Time
Full MLP	100	2	10^{-5}	-	15.807	6.55×10^{-4}	6.69×10^{-5}	1.05×10^{-4}	1.345s
Full RBF	50	0.88	10^{-5}	10^{-5}	1.929	1.2×10^{-3}	1.15×10^{-4}	1.64×10^{-4}	0.234s
Extreme MLP	100	2	10^{-5}	-	n.a.	1.59×10^{-3}	1.86×10^{-4}	2.09×10^{-4}	0.09s
Unsupervised RBF	50	0.88	10^{-5}	10^{-5}	n.a.	3.31×10^{-3}	1.09×10^{-3}	2.36×10^{-3}	0.001s
Block Decomposition	50	0.88	10^{-5}	10^{-5}	1.767	1.43×10^{-3}	9.45×10^{-5}	1.38×10^{-4}	1.62s

1 Full Minimization

1.1 Data shuffling and processing for model training

When loading the dataset, we applied a random shuffling procedure to the data, using a fixed seed within the data-loading function to ensure reproducibility. This practice is generally considered a good approach to prevent any potential ordering or structure in the dataset from inadvertently influencing the model's training process. In this specific case, shuffling was not strictly necessary, as the data points were already generated in a randomized manner and were not expected to exhibit dependencies based on their order. However, including this step contributes to robustness and aligns with standard preprocessing practices.

1.2 MLP

We developed a single-hidden layer FNN using a Multilayer Perceptron (MLP). The gradient of the objective function was computed using a custom-built function rather than relying on autograd. This approach allowed for more efficient gradient calculations, significantly improving the overall performance of the optimization process. The choice of hyperparameters, namely the number of neurons N of the hidden layer, the regularization parameter ρ, and the spread σ of the hyperbolic tangent used as activation function g, was carried out with the implementation of a grid search complemented by K-Fold cross-validation, with $k = 5$. The grid-search intervals were (50, 301, stepsize=10) for N, $\{10^{-5}, 10^{-4}, 10^{-3}\}$ for ρ, and (1, 9, stepsize=0.5) for σ, while for the seed we only explored the seeds $\{100, 200\}$. The hyperparameter-tuning procedure selects the combination of hyperparameters which minimizes the mean validation error across the K folds. This ensures optimal model performance on unseen data. Following the grid search, we realized that the model with the selected parameters was highly complex and computationally expensive. Consequently, we plotted the trends of the mean validation error and the mean training error as functions of each hyperparameter (ρ, σ, and N) individually, while keeping the other hyperparameters fixed at their optimal values as determined through grid search.
The resulting plots, presented in the graphs section, highlighted the potential to reduce the model's complexity without significantly compromising its performance.

The final performance obtained are reported in Table 1. The figure 11 and 12 shows the trend of training and validation error varying the number of neurons N, and does not report real evidence of overfitting; in fact, as N grows, the generalization capability measured by the validation error doesn't seem to worsen. This behavior was expected, as the data points used for validating were generated by the same unknown function that produced the samples used for training.
Once the hyperparameters were set, the full optimization of the network parameters W, b, and V was performed using `scipy.minimize` from the SciPy toolbox. A multistart procedure was implemented to select the best starting point for the minimization. In particular, we searched for the seed in the range (100, 10000, stepsize=76) which generated the lowest mean validation error following the minimization. The obtained seed was 9828. The tolerance in the minimize function was adjusted to a value of 10^{-7} aiming for a balance between minimizing the regularized error and ensuring computational efficiency. This choice allowed the optimization to converge within a reasonable number of iterations without excessive computational cost, while still achieving a satisfactory error reduction compared to the default tolerance value of 10^{-5}.
The key elements of the training procedure are reported below:

- Optimization routine used: L-BFGS-B
- Returned message: CONVERGENCE: REL REDUCTION OF F $\leq$ FACTR $\times$ EPSMCH
- Starting / Final value of the objective function: 15.807 / 6.55 $\times 10^{-4}$
- Starting / Final value of the norm of the gradient: 34.555 / 2.22$\times 10^{-3}$
- Iterations: 1462
- Number of function / gradient evaluations: 1586
- Optimization time: 1.345 s

The plot of the approximating function is reported in Fig. 2 and is compared with the true one.

1.3 Radial Basis Function

In this case, we constructed a Radial Basis Function network. As we did for the MLP model, we computed the gradient of the objective function with a custom function designed specifically for this purpose. This method greatly enhanced the performance by reducing computational time and optimizing the efficiency of the gradient evaluation, which played a crucial role in accelerating the optimization process. The hyperparameters to fix were the number of centers N, the spread σ of the Gaussian RBF and regularization terms ρ_1 and ρ_2. The setting procedure had the same structure as the MLP case: a grid search supported by K-Fold cross-validation. The grid-search intervals were (40, 70, stepsize=1) for N, $\{10^{-4.5}, 10^{-4.75}, 10^{-5}\}$ for $\rho 1$ and $\rho 2$, (0.82, 0.95, stepsize=0.01) for σ, and (100, 301, stepsize=100) for the seed. For simplicity we chose to use a single value ρ beacuse the grid search consistently selected both ρ values as 10^{-5} , indicating not significant differentiation in their optimization. As for the MLP model, we opted to reduce its complexity by analyzing the trends of the mean validation error with respect to the three hyperparameters. The final values are reported in table 1. Once the parameters were set we created a for cycle to determine the choice of the seed (in the range [100, 10000] with a stepsize=76), the obtained seed was 2532.
The trend of the error with respect to the number of centers N is, as expected, similar to the MLP case, highlighting that as the number of centers grows in our range there isn't evidence of overfitting.
The key elements of the training procedure are reported below:

- Optimization routine used: L-BFGS-B

- Tolerance set to: 10^{-7}
- Returned message: CONVERGENCE: REL REDUCTION OF F $\leq$ FACTR $\times$ EPSMCH
- Starting / Final value of the objective function: 1.929 / 1.2×10^{-3}
- Starting / Final value of the norm of the gradient: 1.554 / 5.75×10^{-4}
- Iterations: 374
- Number of function / gradient evaluations: 389
- Optimization time: 0.234 s

The plot of the approximating function is reported in Fig. 4 and is compared with the true one.

Eventually for both the MLP and the RBF model, the selection of hyperparameters, was made iteratively analyzing the behavior of the errors by fixing two hyperparameters at a time and leaving one variable, this is observable in the graphs section. In this case, the choice was guided towards options that provided a good trade-off between efficiency and accuracy.

2 Two Blocks Methods

2.1 Extreme Learning MLP

The hyperparameters were set to the values obtained in point 1.1. In this section, we implemented an Extreme Learning procedure, which involves fixing randomly the weights of the hidden layer W and the biases of each neuron b, and minimizing the regularized quadratic convex error with respect to the only variables v. The optimization problem simplifies into a Linear Least Square (LLSQ) problem with a linear gradient and was therefore solved efficiently with the `np.linalg.lstsq()` method from NumPy. Concerning the random generation of W and b, we implemented a multi-start procedure fixing a seed in the range (1, 100000, passo=1) and we select the seed that guaranteed the lowest value of the error, following the optimization. The seed selected by this procedure was 33410.
As expected, the final values of the training and test errors exceeded those of the fully optimized model, but this trade-off was necessary to enhance computational efficiency. The plot of the approximating function in this case is reported in Fig 6.

2.2 Unsupervised Center Selection RBF

The hyperparameters were fixed to those found in point 1.2. Similarly to point 2.1, the approach followed in this section requires fixing the centers c by randomly selecting N points out of the P training samples of the RBFs and training the network by minimizing the regularized quadratic convex error only with respect to the weights of the output layer v. Also in this case the training procedure turns into a LLSQ optimization problem and was solved efficently using np.linalg.lstsq(), drastically reducing computational time if compared to the fully optimized models.
Similarly to point 2.1, we searched for the seed in the range (1, 300000, stepsize=1) that guaranteed the best performance following the optimization.
The plot of the approximating function in this case is reported in Fig 8.

3 Two Block Decomposition Method (RBF)

In this approach, we implemented an iterative procedure alternating between the minimization with respect to the centers c and the minimization with respect to the output weights v. At each iteration, the centers were updated by fixing the weights v obtained from the previous step and solving the optimization problem to minimize the regularized empirical error. Once the centers were updated, the weights v were recomputed by minimizing the error with the centers fixed to their updated values. This step transformed the problem into a linear least squares (LLSQ) optimization characterized by a linear gradient, which was effectively resolved using the np.linalg.lstsq() function from NumPy. The alternating minimization process continued iteratively until one of the following stopping criteria was met:

- The maximum number of iterations k=500 was reached.
- The reduction in the objective function value was less than 10^{-8} for more than 10 consecutive iterations.
- The norm of the gradient of the objective function dropped below 10^{-5}

This two-block decomposition strategy allowed for a gradual refinement of both the centers and the output weights, leveraging the simplicity of the LLSQ problem for the weight updates. The convergence behavior and final performance of this method are detailed in Table 1, while the approximating function is visualized in Fig. 10.

4 Graphs

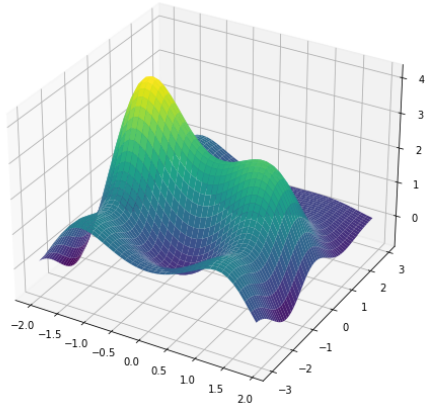

Figure 1: Original Function

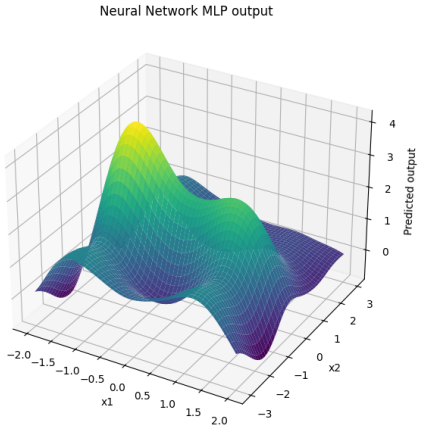

Figure 2: MLP Output

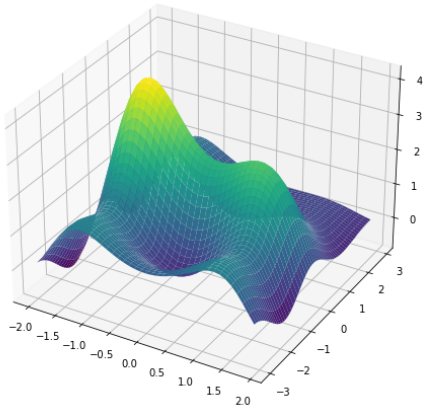

Figure 3: Original Function

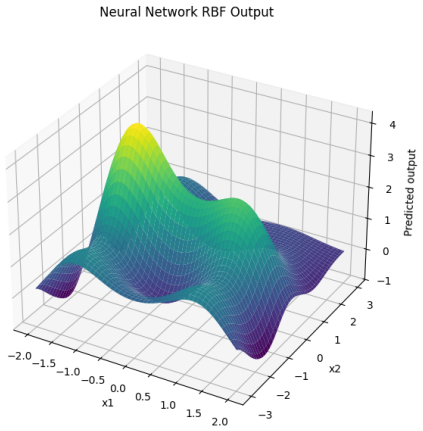

Figure 4: RBF Output

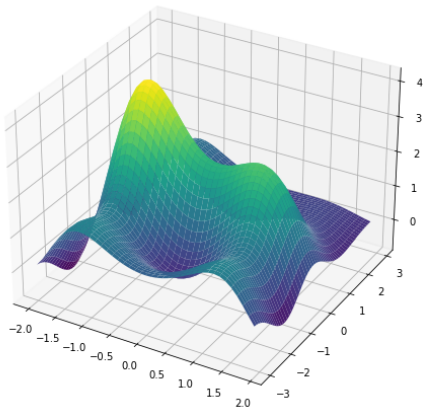

Figure 5: Original Function

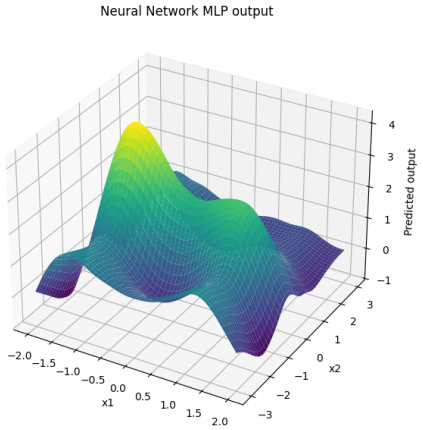

Figure 6: Extreme MLP Output

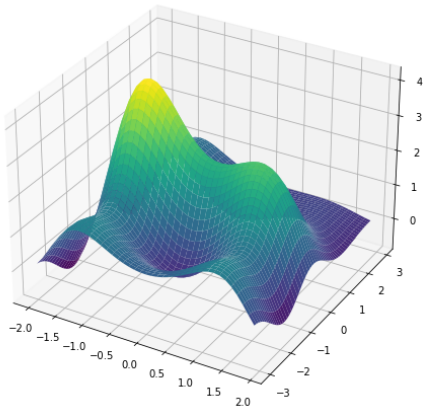

Figure 7: Original Function

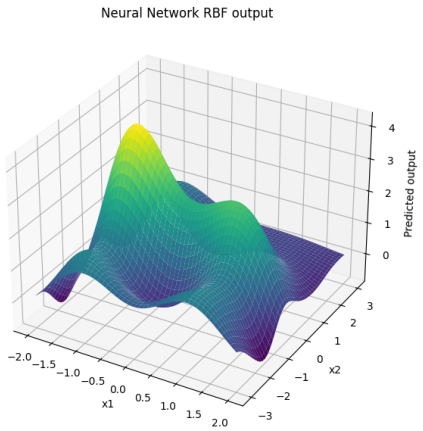

Figure 8: Unsupervised RBF Output

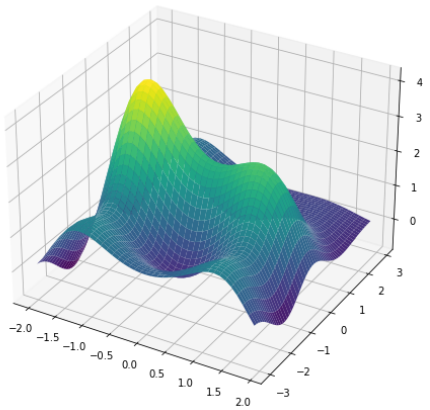

Figure 9: Original Function

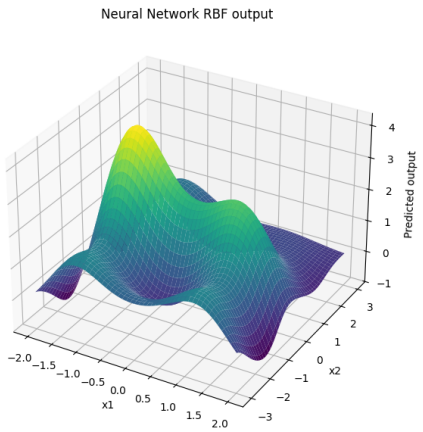

Figure 10: Two Block decomp Output

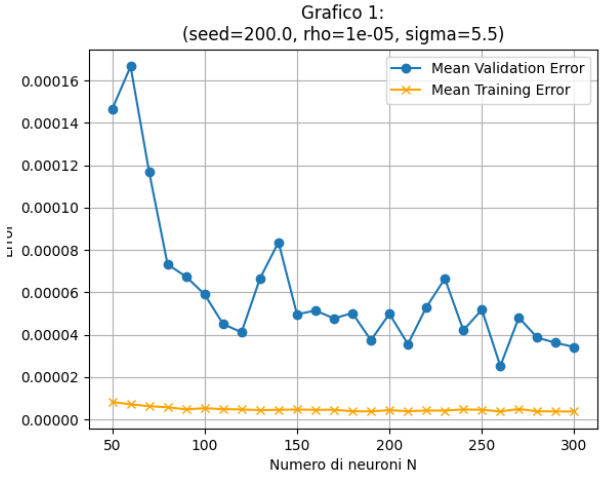

Figure 11: Grid Search N (MLP)

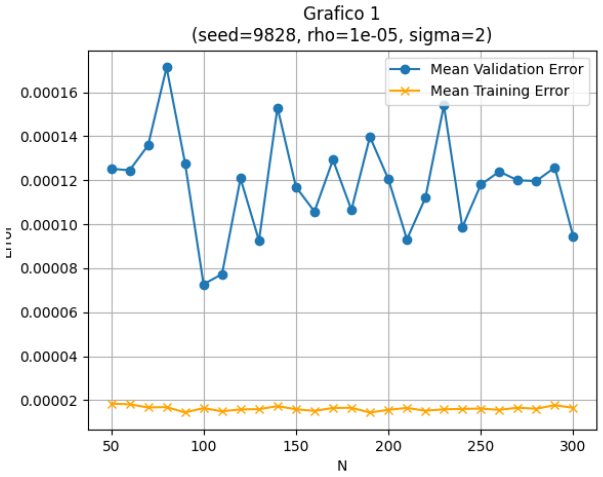

Figure 12: Graph for N final choice

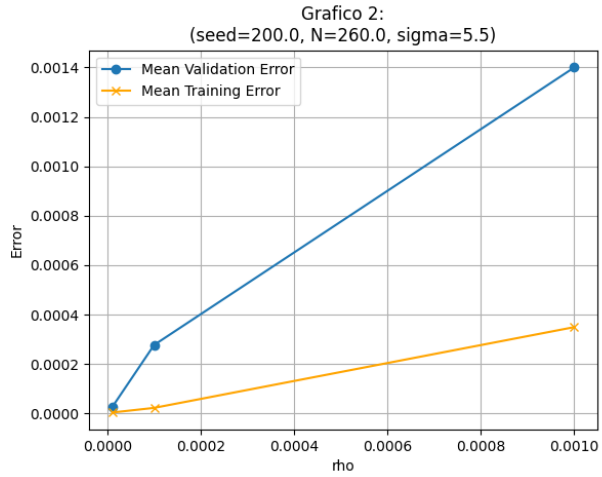

Figure 13: Grid Search rho (MLP)

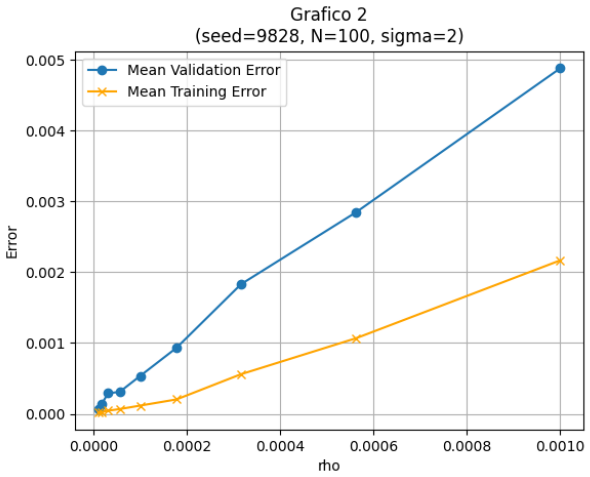

Figure 14: Graph for rho final choice

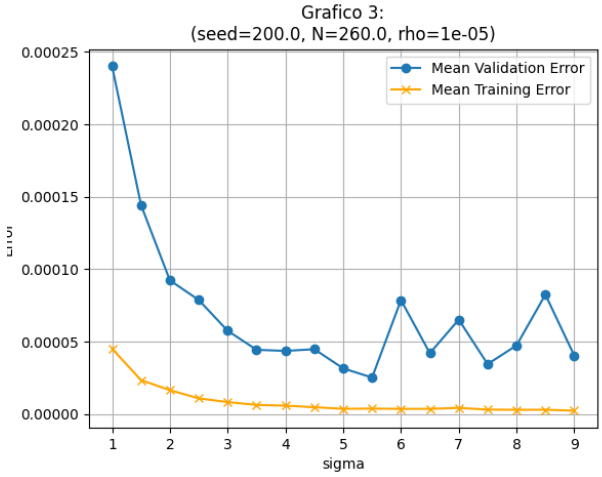

Figure 15: Grid Search sigma (MLP)

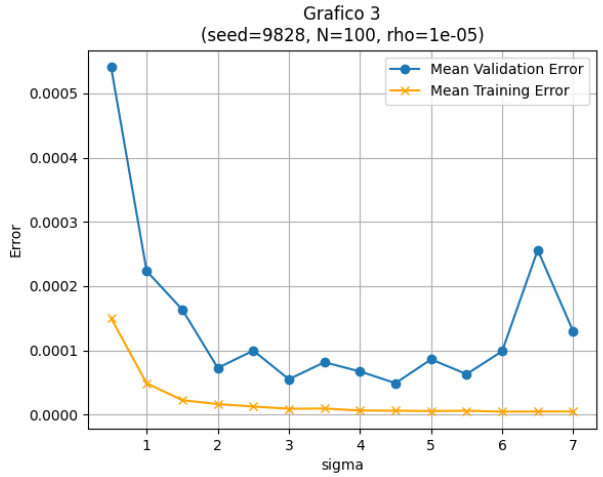

Figure 16: Graph for sigma final choice

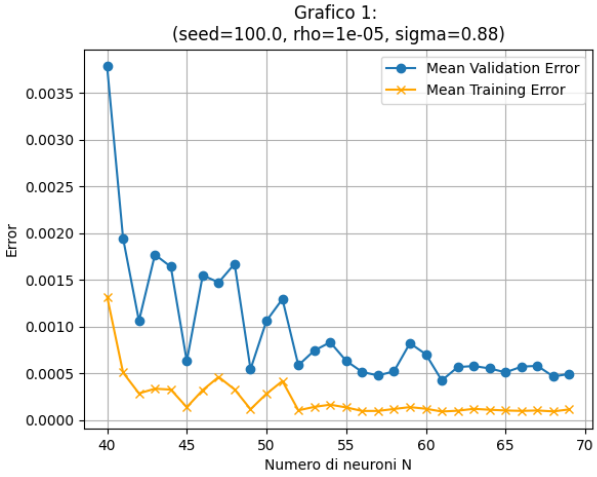

Figure 17: Grid Search N (RBF)

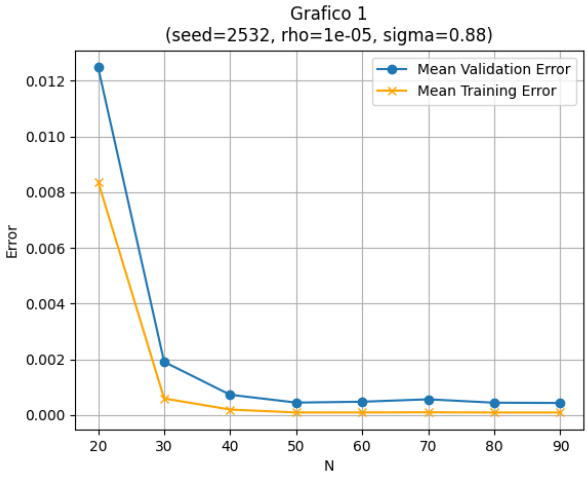

Figure 18: Graph for N final choice

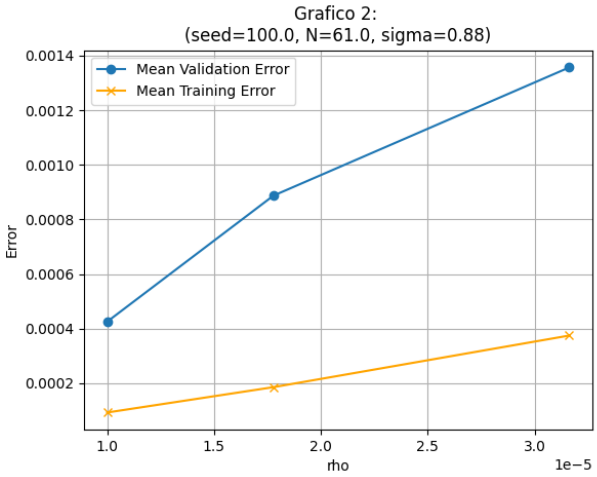

Figure 19: Grid Search rho (RBF)

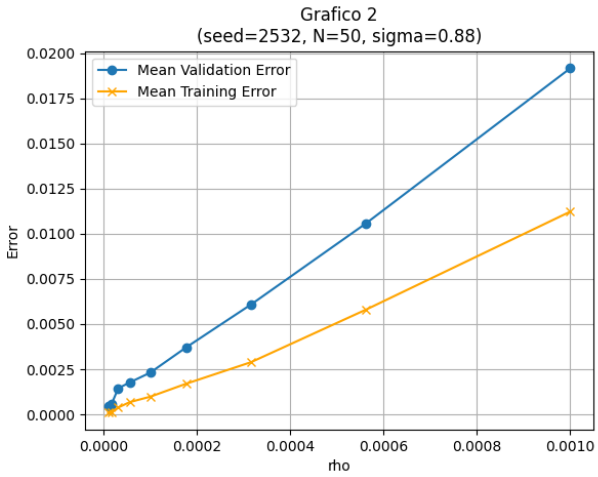

Figure 20: Graph for rho final choice

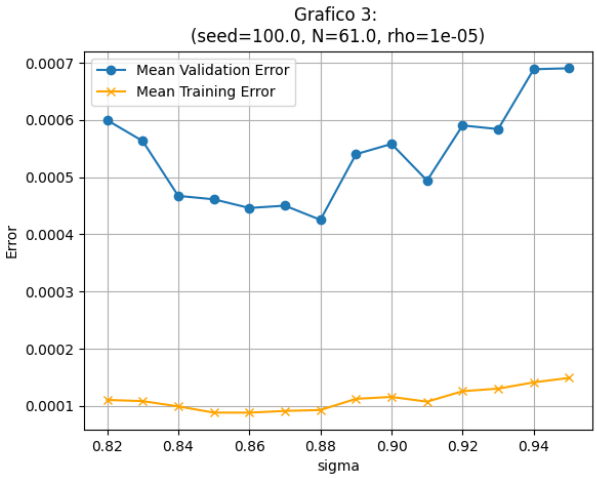

Figure 21: Grid Search sigma (RBF)

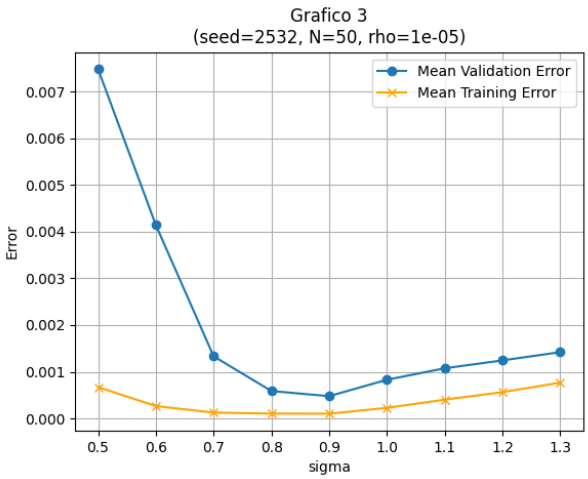

Figure 22: Graph for sigma final choice