Optimization Methods for Machine Learning
MLP and Generalized RBF Network

Group Name : Optimized Noobs
Alessio Barboni - 2027647
Nina Kaploukhaya - 1998079
Mehrdad Hassanzadeh - 1961575

November 20, 2022

Method comparison table

Ex	Model	Settings			Final train error	Final test error	Opt. time
		N	ρ	σ			
Q1.1	Full MLP	35	1e-05	1.4	3.48e-05	1.76e-04	22.71 seconds
Q1.2	Full RBF	30	5e-05	1.2	0.0041	0.009	53.6 seconds
Q2.1	Extreme MLP	35	1e-05	1.4	0.0118	0.02016	0.6452 seconds
Q2.2	Unsupervised c RBF	30	5e-05	1.2	0.014	0.026	40.59 seconds
Q3.0	THE BEST MODEL	35	1e-05	1.4	7e-05	1.4e-4	59.59 seconds

Question 1

Question 1.1

The method and settings we used

The optimization routine chosen was *BFGS*, implemented using the *scipy.optimize* library. This algorithm exploits the gradient of the objective function, that we computed explicitly using the *Backpropagation* algorithm, in order to converge more quickly to the solution. With respect to its derivative-free version, in which the gradient is estimated automatically using *first-differences*, this optimization routine is quite faster, hence we decided not to bound the maximum number of iterations (that although has never exceeded 10.000), and to keep the tolerance of the solver to its default value.

The choice of model parameters

In order to fine-tune the hyper-parameters of the *MLP* (i.e., N, ρ, σ), we implemented *GridSearch* from scratch, and ran it after a further split of the train set into train and validation sets. The hyper-parameters grid was defined by hand and contained a total of 216 combinations of triplets. In the end, the values for which the objective function on the validation set was minimized were 35, 1e-05 and 1.4 for N, ρ, and σ respectively. In addition, we carried out an underfitting/overfitting analysis by varying the hyper-parameters one-by-one, see Figure 3, 4, 5 in the Appendix section. Both the number of nodes in the hidden layer N and the spread σ have not shown any evidence of overfitting at the values we tried, whereas the regularization shows a clear overfitting behavior at large values of ρ.

Regarding the optimization of the parameters w, b, and v, the routine has returned a "successful optimization" message and has improved the loss computed on the objective function from a value of about 5.11 (which depends on the random initialization of the w, b, v parameters) to a value of 6.18e-04, after exactly 5248 iterations and 5325 function/gradient evaluations, in a total of 22.71 seconds. The training error has dropped from 5.11 to 3.48e-05, whereas the test error from 4.31 to 1.76e-04. The plot of the approximating function obtained by the *MLP* is shown in Figure 2 of the Appendix.

Results

- **Starting value of the objective function** was **'5.11'**
- **Final value of the objective function** was **'6.18e-04'**
- **Number of iterations** performed was **'5248'**
- **Number of function evaluations** was **'5325'**
- **Number of gradient evaluations** was **'5325'**
- **Running time** of the model was **'22.71'** seconds
- **Training set error** has reduced from **'5.11'** to **'3.48e-05'**
- **Test set error** has reduced from **'4.31'** to **'1.76e-04'**

Question 1.2

The method and settings we used

For the optimization routine we decided to use *Gradient Descent*, implementing it by hand. We explicitly computed the gradient of the objective function and we used it in the back-propagation stage.

The choice of model parameters

As in question 1.1, in order to fine-tune the hyper-parameters, we split the train data into train and validation sets, and then we ran *GridSearch* on them. The optimal values of the hyper-parameters turned out to be: $N = 30$, $\sigma = 1.2$, $\rho = 5e - 05$. To analyze how the choice of the hyper-parameters can lead to over(under)fitting and thus affect performances overall, we plotted the train and test errors for different hyper-parameters values. See figures 9, 10, 11 in the Appendix. E.g., we can see that a small number of neurons can lead to underfitting, whereas there is no evidence of overfitting for the values of N we have tested.

The choice of learning rate for gradient descent

We also analyzed how different learning rates impact the training error across 2000 iterations of our algorithm (see 8 in Appendix). Based on this we chose learning rate equal to 0.01 as the most optimal one, since an error rate that decreases gradually is preferable.

Results

Our results, upon completing successfully the optimization and using the hyper-parameters found above, where as follows:

- **Starting value of the objective function** was **'4.9'**
- **Final value of the objective function** was **'0.0033'**
- **Number of iterations** performed was **'2000'**
- **Number of function evaluations** was **'2000'**
- **Number of gradient evaluations** was **'4000'**
- **Running time** of the model was **' 53.6'** seconds
- **Training set error** was **0.0041**
- **Test set error** was **0.009**

Comparison RBF vs MLP

Plots of the function representing the approximating function obtained by the MLP (2) and by the RBF (6) networks presented in Appendix. From the results we have obtained, we can conclude that the approximation of the MLP seems more precise, although this was expectable since we set a maximum number of iterations equal to 2000 for the RBF (due to our machine limitations), and no limit for the MLP. The final train and test errors also reflect this.

Question 2

Question 2.1

The method and settings we used

We chose the same optimization routine used for Question 1.1, namely *BFGS* with the gradient computed explicitly, without setting a cap on the maximum number of iterations allowed and keeping the tolerance at its default value.

Results and plot

Compared to the *Full optimization* of Question 1.1, the *Extreme Learning* procedure terminated successfully the optimization in just 394 iterations, against the 5248 of the previous one. However, since we are fixing w_{ij} and b_j at random, and we are minimizing the regularized quadratic convex training error $E(v)$ only w.r.t v, we end up with a larger training and test error. We tested a bunch of seeds in order to have different initializations and at the end we selected the one that gave us the best result.

- **Training set error** was **'0.01184'**, against 3.48e-05 of Question 1.1
- **Test set error** was **'0.02016'** against 1.76e-04 of Question 1.1

Question 2.2

The method and settings we used

We used the same optimization method as for Question 2.1: The gradient method implemented by hand with the hyper-parameters defined above. We chose the centers in the following way:

1. We choose randomly from the training data a number of samples (equals to the number of centers), and assign centers to them.
2. Repeat the whole training procedure for the differently assigned initial centers several times (each time randomly assigned with random.seed equal to one of teammates matricola number) and then choose the best one.

Results and plot

Our results with fixed as described above hyper-parameters where as follows:

- **Training set error '0.014'**. Which is higher than for RBF with supervised selection of centers.
- **Test set error** was **'0.026'**. Also is higher than for RBF with supervised selection of centers.

The plot of the function representing the approximating function obtained by the RBF with unsupervised selection of the centers (7) in comparison with the true one (6) are in Appendix. As we see, the one with supervised selection of centers approximate the function much better.

Question 3

In two block decomposition method we try to split the process of optimization of the parameters into two separate blocks:

1. **Convex subproblem**: which is the problem of optimizing the output weights V(ω_2) where we have the hidden layer weights W (ω_1) and the bias(b) are fixed.
2. **Non convex subproblem**: which is the problem of optimizing the hidden layer weights W (ω_1) and the bias(b) when the output weights V(ω_2) are fixed.

Model Description

In in order to optimize the parameters of the model we will try to fix the parameters in K iterations (where K is one of the hyperparameters of the model) where in each iteration we will do the following:

1. Finding the **global minima** of the **convex subproblem** using the **Dual annealing** method in scipy.optimize by setting the maximum iterations to 500 to find the optimum output weights V(ω_2).
2. Finding the minima (not necessarily the global minima) of the **non convex subproblem** using the **BFGS** method in scipy.optimize by setting the maximum iterations to 2000 to find the optimum hidden layer weights W (ω_1) and the bias(b).

To avoid overfitting the model (and thus **early stopping**), we will use **20%** of the training data for validation. After each iteration, we will determine whether or not the error on the validation set has decreased since the previous iteration. If the model is unable to reduce the error on the validation set in the current iteration, we will reset the parameters to those computed in the previous iteration and terminate the process.

Training results

In our model we choose the hyperparameters of the best model that we resulted in the Question 1 and setting the parameter **K**(the number of the iterations to solve the subproblmes) to **'10'**. Our results where as follows:

- **Initial error** of the model on the training data before starting the optimization process was **'5.037'**
- **Number of outer iterations** performed came to **5** due to the early stopping criteria that we used
- **Total number of function evaluations** came to **'227234'**
- **Total number of gradient evaluations** came to **'11489'**
- **Running time** of the model was **'59.59'** seconds
- **Training set error** reduced to **'7e-05'**
- **Test set error** was **'1.4e-4'**

Moreover, comparing the **true function** in the figure 1 with **the approximated function using the MLP with two block decomposition method** in the figure 12, we can verify that the two plots seem quite the same which means that we were able to approximate the true function very well.

Comparing with randomized method

In this section we want to compare the MLP network using the two block decomposition method with the MLP network we build in the question (Extreme MLP).

Method	Test Error	Opt. Time	# Function Evaluations	# Gradient Evaluations
Extreme MLP	0.02016	0.6452 seconds	398	398
MLP two block decomposition method	1.4e-4	59.59 seconds	227234	11489

The table above clearly shows that the Extreme MLP is much faster and has less computation demand because it evaluates fewer functions/gradients at the expense of having higher test error. In terms of model accuracy, MLP with two block decomposition method outperforms Extreme MLP, but it requires more computing power.

Appendix

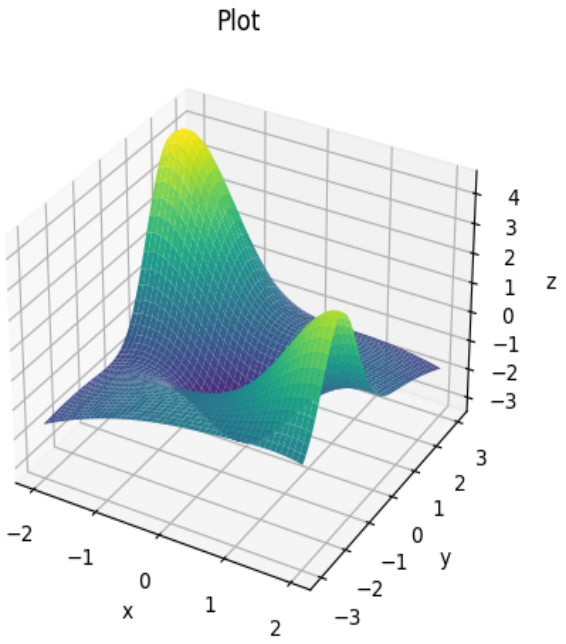

Figure 1: The true function

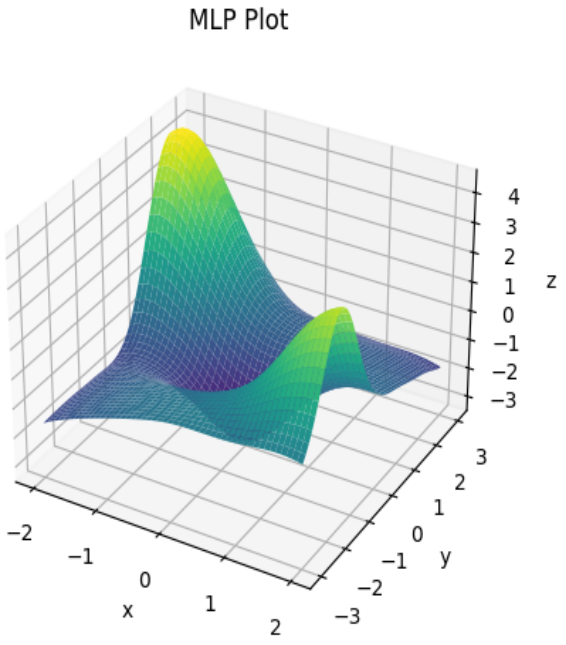

Figure 2: MLP approximation function

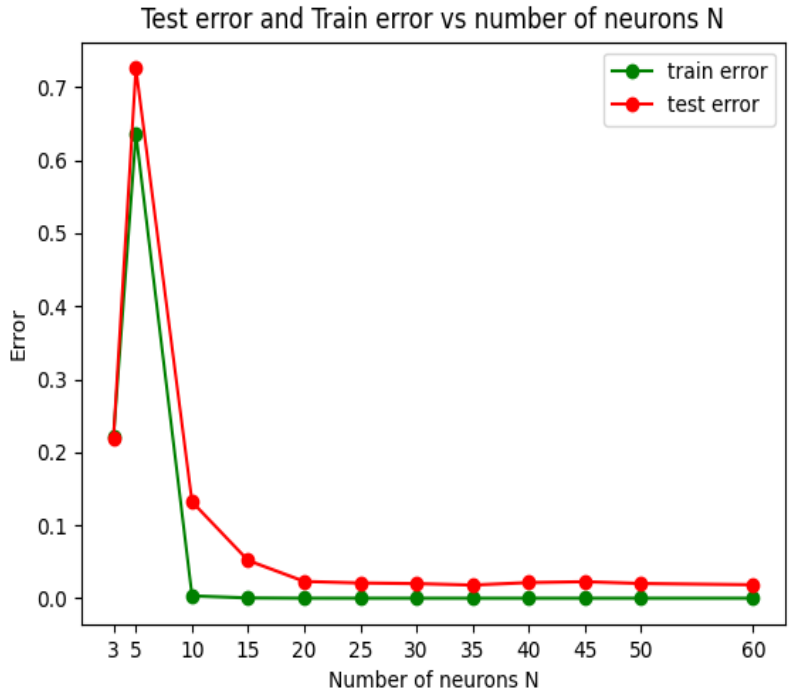

Figure 3: MLP Over/Underfitting Analysis w.r.t. N

Figure 4: MLP Over/Underfitting Analysis w.r.t. σ

Figure 5: MLP Over/Underfitting Analysis w.r.t. ρ

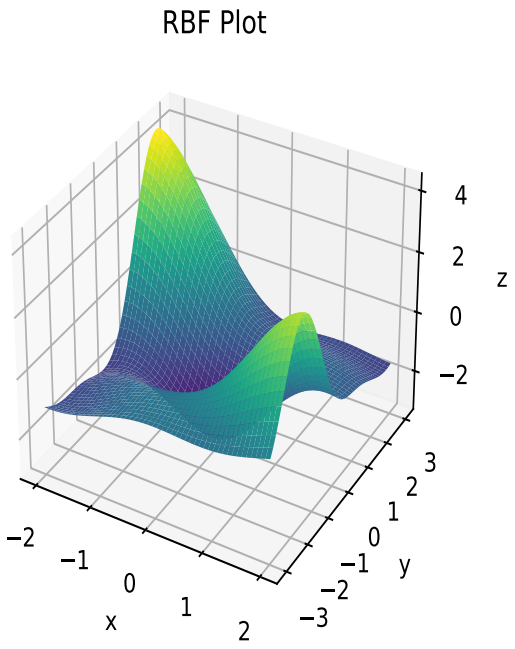

Figure 6: RBF with supervised choice of centers

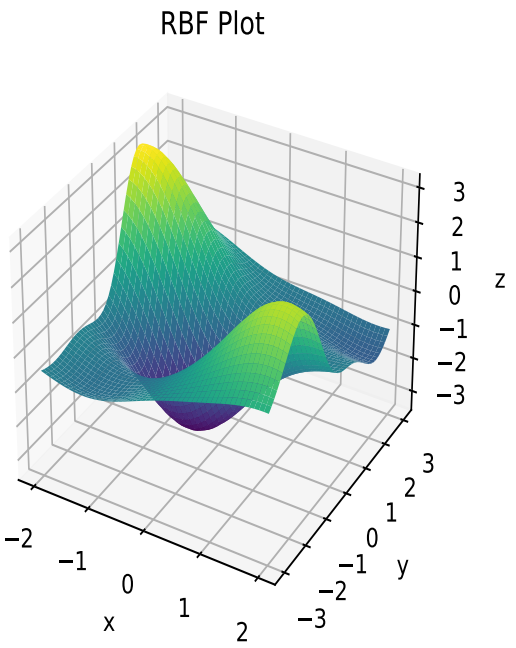

Figure 7: RBF with unsupervised choice of centers

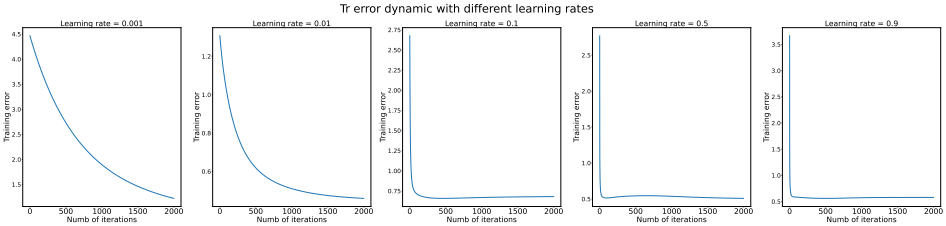

Figure 8: RBF Train error vs Learning rate

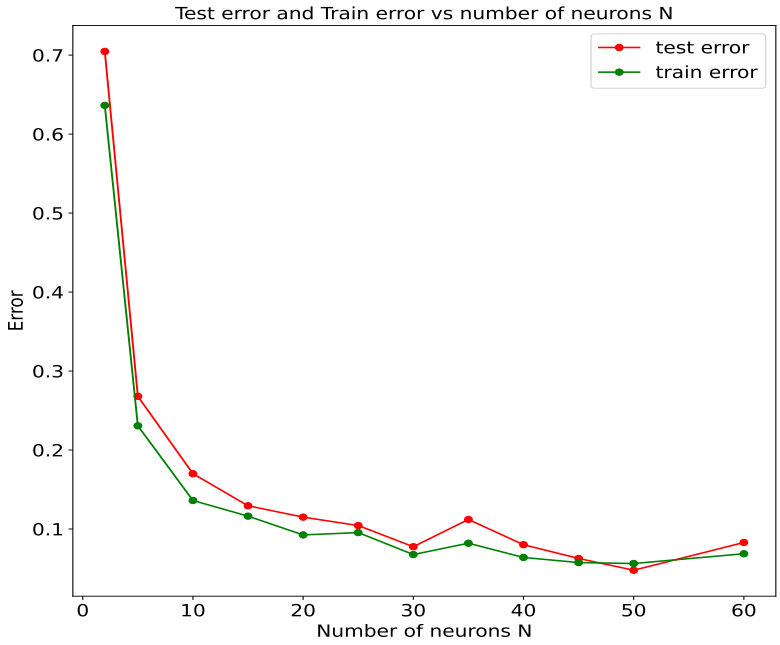

Figure 9: RBF Train/Test error vs Number of neurons

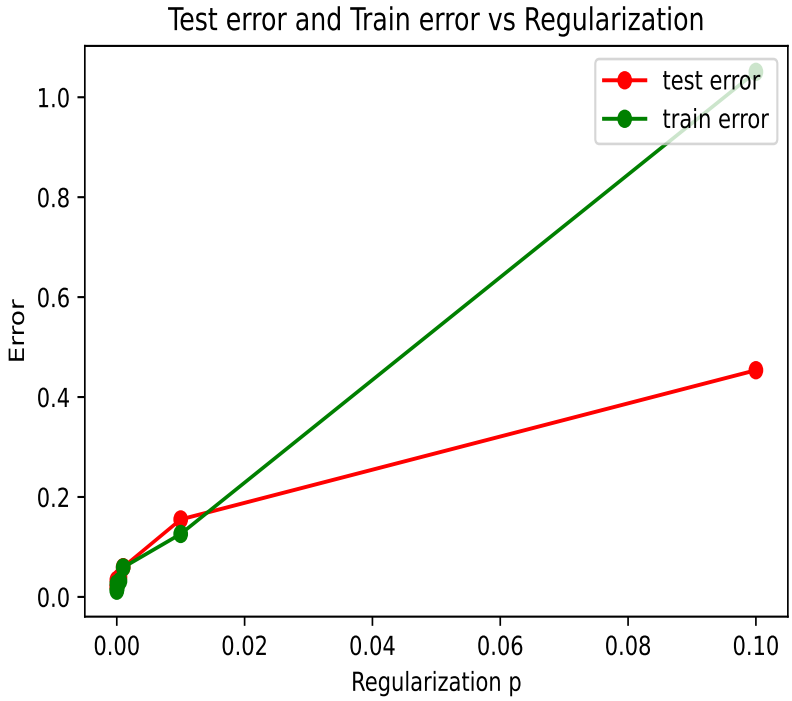

Figure 10: RBF Over/Underfitting Analysis w.r.t. ρ

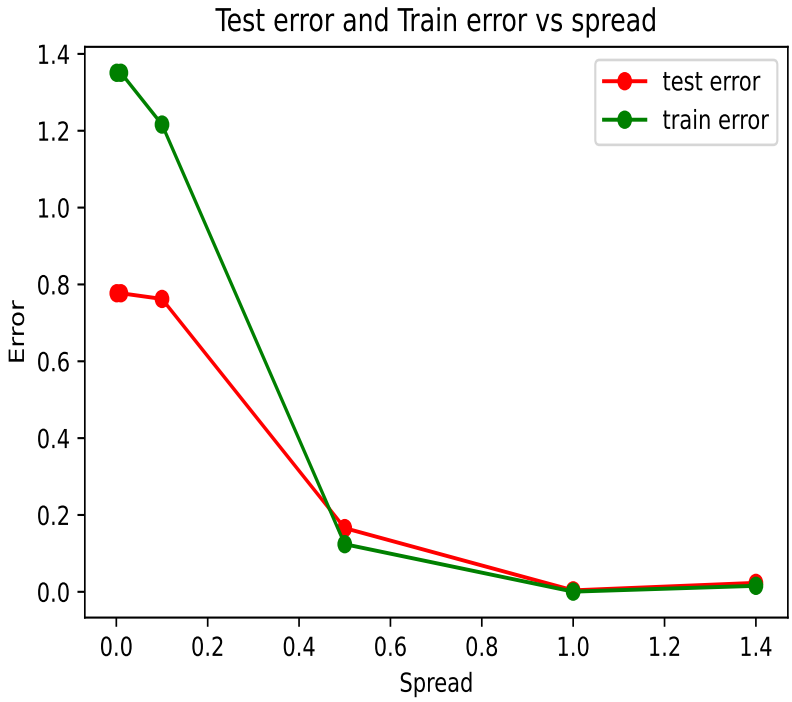

Figure 11: Over/Underfitting Analysis w.r.t. σ

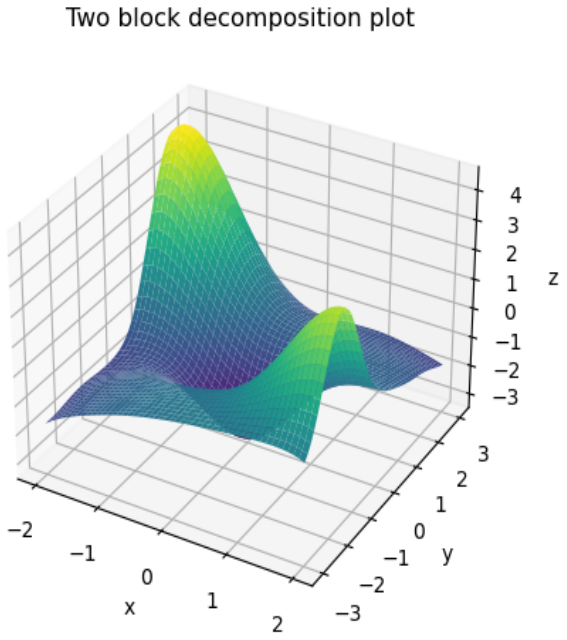

Figure 12: MLP with the two block decomposition method approximated function