Project 1 - MLP and Generalized RBF Network

Livia Lilli (1629888), Joanna Broniarek (1868264), Davide Facchinelli (1843896)

Optimization Methods for Machine Learning
Prof. Laura Palagi

January 19, 2020

SAPIENZA
UNIVERSITÀ DI ROMA

Contents

The main goal of this project was to implement neural networks to solve a regression problem. It was required to reconstruct a function $F : \mathbb{R}^2 \to \mathbb{R}$ in the region [-2,2] $\times$ [-1, 1]. Together with the task, the data set of 300 points were provided $\{(x^P, y^P) : x^P \in \mathbb{R}^2, y^P \in \mathbb{R}^2, p = 1, ..., 300\}$.

1 Question 1 (Full minimization)

In this part, two shallow Feedforward Neural Networks (FNN with one hidden layer) were implemented: a MLP and RBF networks. The hyper-parameters π of the network were settled by means of an heuristic procedure and the parameters ω were settled by optimizing the regularized training error:

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

1.1 A shallow MultiLayer Perceptron

A shallow MultiLayer Perceptron with a linear output unit was implemented. The formula is

$$f(x) = \sum_{j=1}^{N} v_j g\Big(\sum_{i=1}^{n} w_{ji} x_i - b_j \Big) \tag{2}$$

where $\omega = (v, w, b)$. The activation function $g(t)$ is the *hyperbolic tangent*. The hyper-parameters are: the number of neurons N of the hidden layer, the spread σ in the activation function **tanh()** and the regularization parameter ρ .

In order to have the clean and readable code, we decided to use the object-oriented programming paradigm. The implementation of the class MLP with all needed functions is located in a file **MLP.py**. The code execution with initialization, training and predicting is implemented inside the file **run_11_DaJoLi.py**. After running this file, the plot of the approximating function is saved in the working directory as **Predicted_Function_Plot11.png**.

The first step was preparation of the data by randomly splitting into target and test sets, and converting into *numpy* objects. The percentages used for this split are 85 % for target set (further additionally splitted into a training and validation sets) and 15 % for test set.

The next step was preparation to the training part. In order to find the best hyperparameters N, ρ, σ, the Grid-Search method was used. Hence, a grid for hyperparameters was specified, together with the optimization method for solving the minimization problem (used method: **BFGS**).

Regarding the activation function, the static method $tanh(t, sigma)$ inside MLP class was defined by ourselves. In order to avoid very big values both at numerator and denominator (which are read by *numpy* as infinite values) the function was rewrote in such a way:

$$tanh(t, \sigma) = \frac{1}{1 + 1/exp(2\sigma t)} - \frac{1}{exp(2\sigma t) + 1}$$

With this formula the infinite values are only as a denominator and *numpy* treats them as zeros.

Proceeding to the training part, the full minimization and the technique of K-fold cross validation was used. It is possible to specify number of splits for CV (n_{CV}) - our choice was 5 in order to have a validation set with a size of approx. 17 % of the whole dataset. For each combination of hyperparameters $\pi = (N, \rho, \sigma)$, the minimization of training error $E(\omega; \pi)$ is performed (with the use of *scipy.minimize* function). The optimization is done with respect to the ω vector. The initial ω_0 is selected randomly. In order to measure the performance of our model, the validation error (Mean Square Error) is calculated inside each loop based on a validation set, specific set of hyperparameters and the optimal parameter ω_{opt}. Because of the k-fold cross validation the validation error is an

average error among all k splits. The best hyperparameters π correspond to the lowest obtained averaged validation error. The optimal parameter ω is the result from the minimization of training error with the fixed best hyperparameters.

The last but not less important step is to compute the MSE on both target and test set in order to check the performance of model on new data. (More in the part Analysis & Results).

Analysis & Results

Initially, to perform efficiently our analysis, a small grid of hyperparameters with bigger range for a number of neurons and few picks for the rest was run. Based on obtained results the best number of neurons was selected. As shown in the plot (see Figure 1) after 12 neurons the error is not decreasing significantly, so the analysis was continued with 12 as the number of neurons. Then, a second Grid-Search was run but this time with fixed number of neurons and possibly various values for the other hyperparameters. The top 5 results are shown in the table (see Table 2). Taking into account the number of evaluation needed to train our network and validation error the best selected hyperparameters are $N = 12, \sigma = 0.34, \rho = 8e-08$. The third line of the mentioned table was another good candidate as it gives slightly worse error and better performance, but the improvement in the computing time was not enough convincing for us to use it. The final choice for the hyperparameter ρ is lower than the suggested boundary. However, running many trials confirmed that lowering it greatly improves the performance of the model on both train and test set. The hypothesis and possible explanation for this behaviour is that the data have a very low quantity of noise, and a low value of ρ is enough to prevent the function to interpolate a train set. In order to decrease the complexity of the model and therefore to avoid overfitting a possibly lower number of neurons was kept. The BFGS optimization routine was used as it was the best method in terms of performance. The final predicted function is plotted to confirm also graphically that the final model obtained good results (see Figure 2).

1.2 Radial Basis Function network

Radial basis function (RBF) network is built from three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer. The output of the network is a scalar function of the input vector and is given by:

$$f(x) = \sum_{j=1}^{N} \nu_j \phi(\parallel x^i - c_j \parallel) \tag{3}$$

where $\omega = (\nu, c), \nu \in \mathbb{R}^N and c_j \in \mathbb{R}^2, j = 1, ..., N$. The RBF activation function is defined as the Gaussian function:

$$\phi(\parallel x^i - c_j \parallel) = e^{-((\|x^i - c_j\|)/\sigma)^2}, \sigma > 0.$$

The hyper-parameters are: the number of neurons N of the hidden layer, the spread $\sigma > 0$ in the RBF function ϕ and the regularization parameter ρ.

The implementation of the class RBF is located in a file **RBF.py**. The code execution with initialization, training and predicting is implemented inside the file **run_12_DaJoLi.py**. After running this file, the plot of the approximating function is saved in the working directory as **Predicted_Function_Plot12.png**.

Data splitting is done in the same way as it was done for MLP. The procedure of initializing and training a model is similar to previously described MLP. The differences are: a new implemented function for predictions based on a formula (3) , a new activation function and a new structure of initializing the parameter $\omega = (\nu, c)$. In this case, to solve the minimization problem with respect

to both (ν, c) the **CG - Conjugate Gradient** method was used.

Analysis & Results

Grid search was performed exactly as in the MLP network case. The results are shown in Figure 3. The best hyperparameters are $N = 14, \sigma = 1, \rho = e - 09$. As before the choice is a result of maintaining a balance of performance, optimization time and complexity of a model. The plot of the predicted function was generated (Figure 4).

1.3 Comparison of MLP vs. RBF network

As it is shown in the summary table, using roughly the same number of neurons MLP performed far better both in terms of error and computational time. Probably a better choice of the initial centers c for the RBF (e.g.: as the centroids of a cluster) would have provided better results. The plots validate those observations.

2 Question 2. (Two block methods)

In this part the same as before two learning machines were considered. Now the hyperparameters are fixed as the best ones found with the Grid-Search. Additionally, the parameters are divided in two blocks: the output weights are in the second block, and all the others are in the first block.

2.1 MLP with Extreme Learning

The second block is being trained, while the parameters in the first block are fixed.

The implementation of the class EL_MLP is located in a file **EL_MLP.py**. The code execution with initialization, training and predicting is implemented inside the file **run_21_DaJoLi.py**. After running this file, the plot of the approximating function is saved in the working directory as **Predicted_Function_Plot21.png**.

Analysis & Results

The MLP model was trained with fixed hyperparameters - the best ones obtained in the first question and the same initial parameters. As it is shown in the summary table, the results are far worst that for the fully-minimized MLP, but it was totally expected by us because the non linear part was fixed. What was done is a linear regression on a transformation of the input, with no guarantee that this transformation is going to adapt to be learnt linearly. With significantly greater number of intermediate neurons the results probably would be much better. It was also observed that before the optimization the error on the test set was 227, and after 1. Hence, the algorithm have learnt as much as possible. It can be seen from the plot (see figure 4) that the approximation is much worse than the fully-minimized MLP.

2.2 RBF with Extreme Learning

About implementation

The implementation of the class EL_RBF with all needed functions is located in a file **EL_RBF.py**. The code execution with initialization, training and predicting is implemented inside the file **run_22_DaJoLi.py**. After running this file, the plot of the approximating function is saved in the working directory as **Predicted_Function_Plot22.png**.

Analysis & Results

Let us keep in mind the consideration done in question 2.1. This time both the before-training error and the after-training error on the test set were lower: respectively 30 and 0.4. Possibly it is because the generalized RBF are a generalization of the regularized RBF, for which having fixed centers is a key concept.Therefore, it is natural that fixing the first block of parameters is more beneficial to RBF than to MLP networks. As before the prediction function is plotted (see figure 6).

3 Question 3. (Decomposition method)

This time, only the MLP is considered with the parameters divided in two blocks as the question before. However, in this case there is no parameters fixed. The current methodology is to find the optimal value for one block fixing the other alternatively.

3.1 MLP with decomposition method

The implementation of the class DM_MLP is located in a file **DM_MLP.py**. The code execution with initialization, training and predicting is implemented inside the file **run_3_DaJoLi.py**. After running this file, the plot of the approximating function is saved in the working directory as **Predicted_Function_Plot3.png**.

Analysis & Results

The optimization of MLP network with the same starting parameters and hyperparameters as in question 1.1 is performed. This time, parameters are optimized alternating the optimization between the first and the second block of parameters. Respectively, the **BFGS** method was used for the first non-convex block, and a **CG** method for the second quadratic block, also with analytically computed gradients. To stop the training, both the norm of the gradient and the size of the improvement on the error function was being checked. The stop of optimization was when any of these values was below fixed thresholds. As shown in the summary table, the validation error is slightly above the train and test error. Initially, it was considered as overfitting, but as the test error is lower an investigation proceeded. The difference is in the stopping criterion used for the optimization. During training the final network the whole set is used. Instead, during the estimation of the validation error a slightly smaller dataset is used and some observations are kept out for cross validation. While optimization the parameters on the smaller dataset, the stopping condition is always with respect to the second gradient and during the final training with respect to the first one. Probably the convergence on the smaller dataset is much harder, blocking the model to reach good minimum points. The predicted function (see Figure 7) seems to be very similar to the plot obtained in question 1.1.

3.2 Comparison to Question 1

Similar results were obtained for both models, even if the decomposed MLP performed slightly worst than fully-minimized MLP. The small difference is noticeable looking at plots: see Figures 2 and 7. However, the optimization time is lower, even if not by far.

3.3 Comparison to Question 2

In comparison to the results obtained with random selection of the hidden layer parameters, here the results are naturally better. In fact, in this case the search is for the same minimum as in the question 1.1, only with a different optimization technique. Therefore, all the observations about the comparison of question 1.1 and question 2.1 are valid as much as before.

4 Summary and figures

Table 1: Summary Table

Ex.	FFN	Settings			Training error	Validation error	Test error	Opt. Time
Q1.1	Full MLP	N=12	$\sigma \sim 0.34$	$\rho \sim 8e-08$	6e-05	1.7e-04	1e-04	16s
Q1.2	Full RBF	N=14	σ=1	ρ=e-09	0.049	0.07	0.045	118s
Q2.1	Extreme MLP	N=12	$\sigma \sim 0.34$	$\rho \sim 8e-08$	1.16	1.30	1.16	0.16s
Q2.2	Unsupervised c RBF	N=14	σ=1	ρ=e-09	0.32	0.37	0.36	0.63s
Q3	Two Block MLP	N=12	$\sigma \sim 0.34$	$\rho \sim 8e-08$	2.2e-04	7.6e-04	2.9e-04	15.5s

Figure 1:

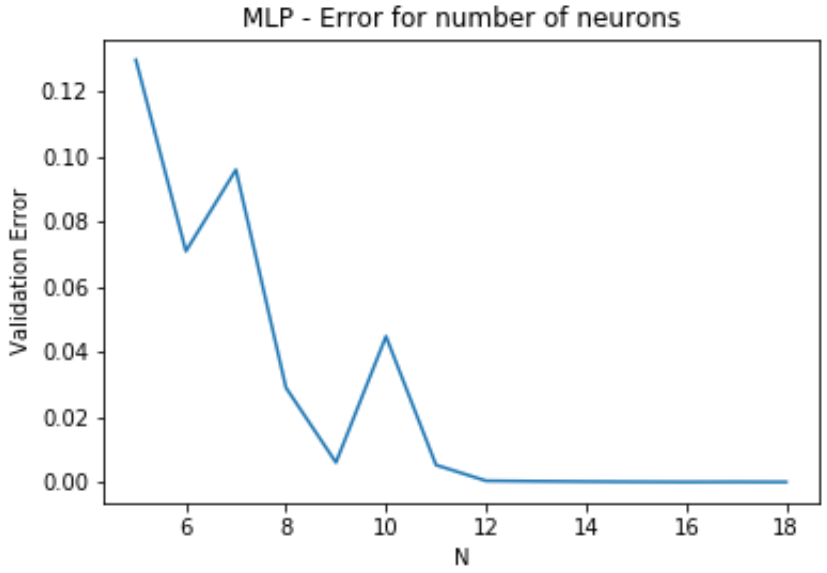

Table 2: MLP Grid-Search

N	ρ	σ	Validation error	Gradient norm	Functin evaluations	Gradient evaluations
4444.44	0.01	0.813125	29	optimal		
12	1.51429e-07	0.342857	0.000137621	2.00115e-05	210400	4208
12	1e-08	0.342857	0.00015207	2.56739e-05	72400	1448
12	1e-08	0.2	0.0001767	1.93562e-05	129600	2592
12	2.22143e-07	0.342857	0.000193113	2.35886e-05	249850	4997

Figure 2:

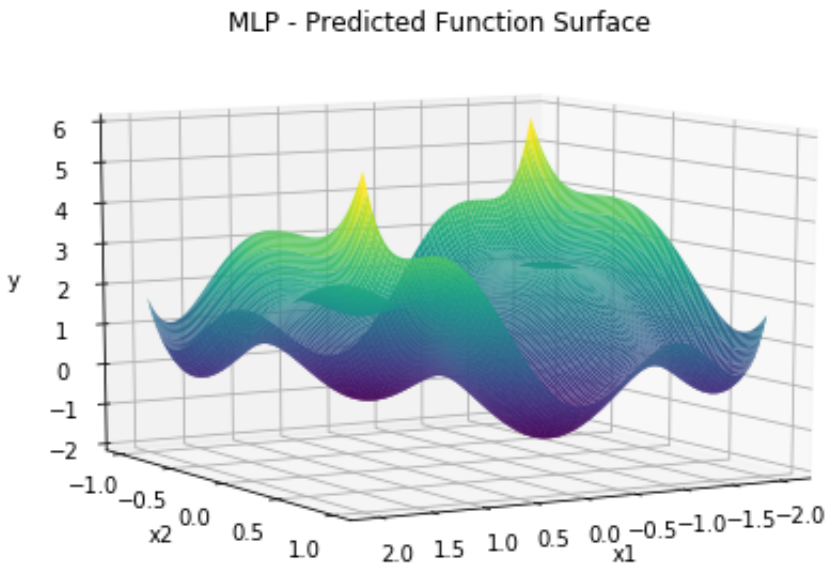

Figure 3:

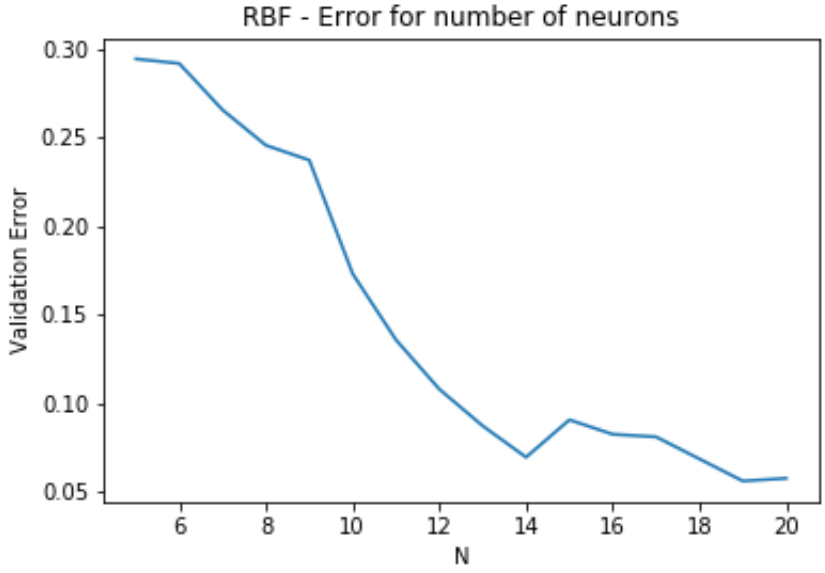

Figure 4:

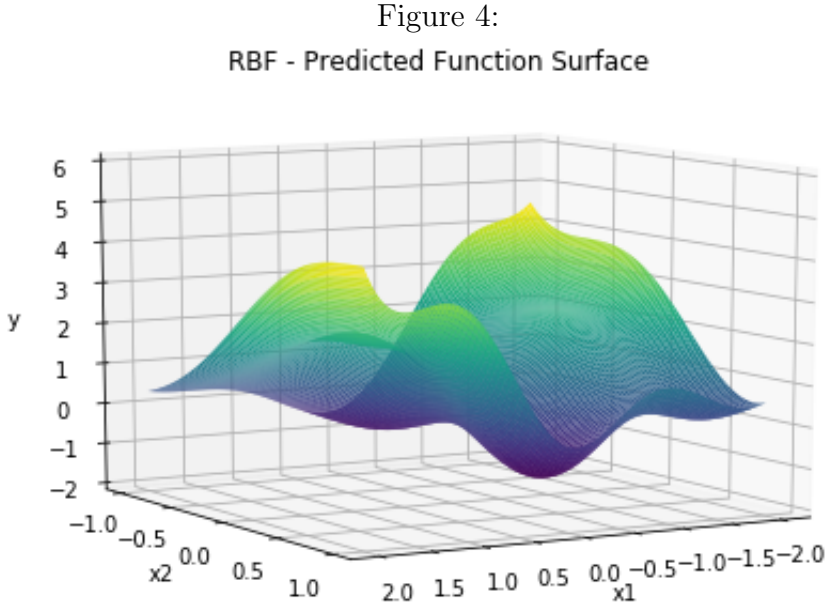

Figure 5:

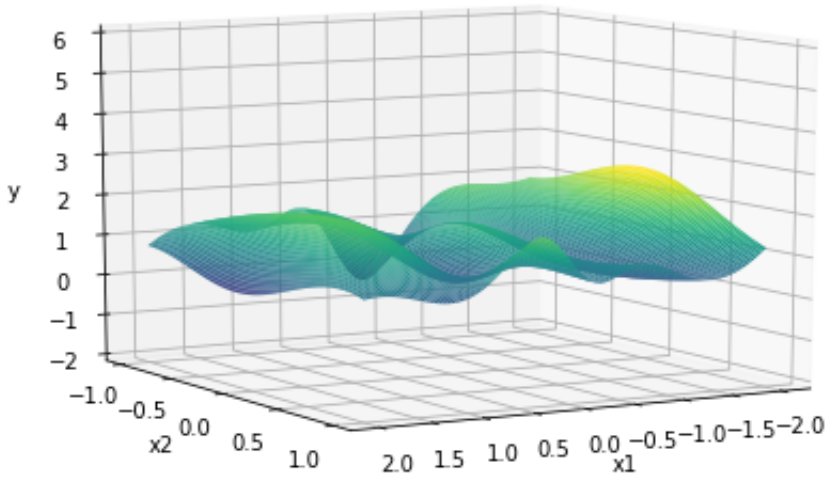

Figure 6:

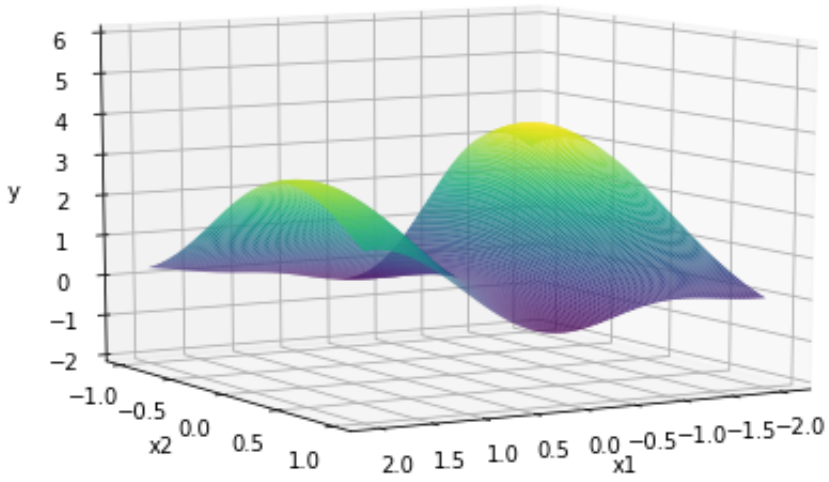

Figure 7: