Optimization Methods for Machine Learning

Project 1

Group JAB : Aurèle Bartolomeo, Joonas Järve, Baudouin Martelée

November 2021

Contents

Introduction

Goal and Context

The present work aims to reconstruct a two dimensional function $F : \mathbb{R}^2 \to \mathbb{R}$ from which we don't have the analytic expression. The reconstruction (in the region [-2;2]x[-3;3]) will be based on 250 points $(x^p; y^p)$ where y^p is defined as $F(x^p)$ in addition with a small random noise. In a nutshell, we will use neural networks based first on Multilayer Perceptron, then on Radial Basis Functions.

Technicalities

Here are some important remarks about our implementation

Initialization of the weights

The weights of our RBF networks are initialized by sampling from a normal distribution. However, for MLP networks, we initialize the weights by sampling uniformly randomly from the range $[-1/\sqrt(n); 1/\sqrt(n)]$ where n is the number of inputs to the node. This is called the Xavier initialization method.

MLP and RBF Comparison

Note that we have taken the exact jacobian for RBF networks, which improves drastically the optimization time. For this reason, it is a bit tricky to compare RBF and MLP in terms of optimization time. However, we can compare them in terms of number of function evaluations, for example.

1 Question 1. (Full minimization)

In this section we construct two shallow Feedforward Neural Network : a MLP and a RBF network. The goal is to find a function $f(x)$ which approximates the true function F. The regularized training error will be calculated using the following formula :

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

where ω/π are the parameters/hyperparameters. Observe that the regularization parameter ρ belongs to π.

1.1 MLP

As activation function for the MLP network we use the hyperbolic tangent

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

where the spread parameter σ will also belong to the set of hypeparameters π.

We searched for the best hyperparameters (N, σ, ρ) by performing a grid search. Our final values for these parameters are (respectively) : 32, 1 and 0.0009. The optimization routine that we use is CG (conjugate gradient), without setting tolerance or maximum number of iterations. The 3 dimensional plot of the approximating function can be found in the Appendix of the corresponding section and we can already recognize the shape of the function we are looking to reconstruct. In order to analyse how the error behaves compared to the hyperparameters, we decided to plot the testing error depending on every hyperparameter *rho*, *sigma* and N (plots are in the Appedix). For each hyperparameter, we fix the two other values as the one we found with the grid search. We observe that the three plots have a different behaviour. For hyperparameter *rho*, the error seems to have a maximum peak around 0.0003 and decreases when going smaller or greater. The *sigma* curve has the opposite behaviour : the lowest error is around 1.5 while it increases if *sigma* is smaller or greater. The last plot is maybe the most interesting. We observe that the more N increases, the more the error decreases. However at a certain point the error seems to reach a plateau. This can be explained by the fact that a too small number of units may cause underfitting (the network is too simple). However, the fact that the error does not decrease anymore after a certain point could be the sign that the model is starting to become too complex, with a risk of overfitting if N goes up. Particular values of this task (regarding error and computational time) can be found in the final table.

1.2 RBF

As Radial Basis Function for our network we choose the Gaussian function

$$\phi(||x - c_j||) = e^{-(||x-c_j||/\sigma)^2}$$

As for MLP, the spread parameter σ of the function ϕ will be added to the list of hyperparameters. After performing a grid search we decide to take as hyperparameters : N=32, σ=1, rho=0.0009.

AS before, we use a conjugate gradient optimization routine with no tolerance or maximum number of iterations.

The plot of the approximating function can be found in the Appendix. We analyse the occurence of overfitting/underfitting when varying the values of our hyperparameters. We decide to use the same method as for MLP, that is, we plot the test error in function of each hyperparameter (see Appendix). When looking at a particular hyperparameter, the value of the other are fixed to the best value that we found. The plot concerning ρ is very clear. Its best value seems to be around 0.0009 and the error increases as we move away from this value. This is easy to explain : if ρ is too small and since it controls the regularization parameter, it will not prevent overfitting as it should. Conversely, if ρ grows too much, it will shrink the weights values and the model selected will underfit, making the testing error grow. In the error plot for σ we note that it is minimal at 1, while the error keeps decreasing as N grows until 32.

The initial loss is 1.5768 while the final loss is 0.0750. We have also calculated the number of iterations and it has value 68.

1.3 Comparison of the two networks performance

Recall that the majority of the values that we discuss in this section are listed in the summary table at the end of the report. At first, we can compare the plots of the approximating function obtained and we note that the MLP one seems to fit better with the true function plot, even though the RBF one is not bad. We confirm this first comment when we look at the testing error for both models. The testing error for RBF if approximately 4 times the one that we have for MLP. However, the computational cost for MLP is much higher. The time for optimizing the full MLP network is 10 times greater than for RBF (12.6 seconds compared to 1.10). It is confirmed also by looking at the number of function evaluations (63210 vs 373) and gradient evaluations (490 vs 361).

2 Question 2. (Two blocks methods)

2.1 MLP

In this section we have implemented an Extreme Learning procedure. We keep the same network hyperparameters as in Question 1.1. We find the optimal weights using a conjugate gradient method (called "CG" in the python package scipy). We observe the following differences with Full MLP :

- the train error is greater than for Full MLP (0.017 vs 0.002)
- the test error is greater (0.021 vs 0.005)
- the number of function evaluations is greater (106038 vs 63210)
- the number of gradient evaluations is greater (822 vs 490)
- the optimization time is greater (23.42s vs 12.59s)

We see that the computational effort with this method is greater, and does not improve the results in terms of training et test error. We have also made a plot of the approximating function in this case (see Appendix) and it seems to be a bit less resembling to the true function than the one we had with the full MLP.

2.2 RBF

Now, we take the same network design as in Question 1.2, that is, we take the same hyperparameters : N=32, σ=1 and ρ=0.0009. We then find the optimal weights by minimizing the regularized error using a conjugate gradient method (called "CG" in the python package scipy). We observe the following differences with Full RBF :

- the train error is greater than for Full RBF (0.042 vs 0.019)
- the test error is greater (0.053 vs 0.023)
- the number of function evaluations is greater (1247 vs 373)
- the number of gradient evaluations is greater (1247 vs 361)
- the optimization time is two times lower (0.53s vs 1.10s)

The plot of the approximating function (see Appendix) is quite similar to the one we had in Question 1.2.

3 Question 3. (Decomposition method)

We have implemented a two-blocks decomposition method for our RBF shallow network. We set the hyperparameters at the values selected in Question 1. The main principle behind this decomposition is to alternate the convex minimization with respect to the output weights with the non convex minimization with respect to the centers c. We used a conjugate gradient optimization routine to solve the two block subproblems. The maximum number of iterations is set to 40 and the tolerance to 0.001. The number of outer iterations at the end of the optimisation is 4. The number of function and gradient evaluation is 4709. The optimization time is 3 seconds. All these values are greater than for Question 2, which leads to the conclusion that it is less efficient. Quaitatively, this decomposition leads to a better training error (0.007 vs 0.042) and a better testing error (0.011 vs 0.053). The plot of the function obtained is in the Appendix and seems very close to the true function.

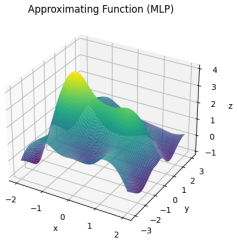
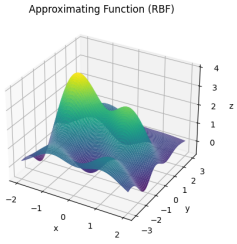
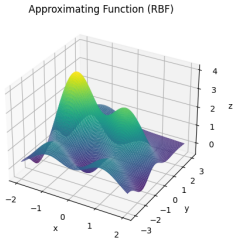
4 Conclusion : Final Table

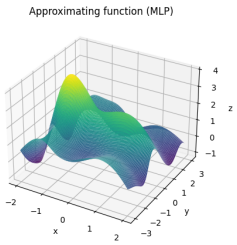
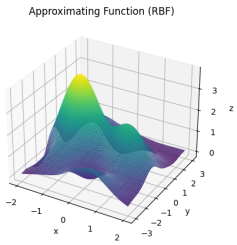
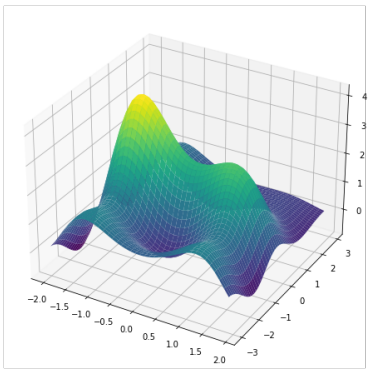
When compiling all the results we discuss in this report, we obtain this final table, that sums up the most important information.

Ex		N	Sigma	Rho	Final train error	Final test error	optimization time
Q1.1	Full MLP	32	1.5	7e-04	0.0029	0.0059	12.5911
Q1.2	Full RBF	32	1	9e-04	0.0194	0.0238	1.1027
Q2.1	Extreme MLP	32	1.5	7e-04	0.0172	0.0217	23.42
Q2.2	Unsupervised c RBF	32	1	9e-04	0.0429	0.0536	0.5324
Q3	Decomposition Method (RBF)	32	1	9e-04	0.0071	0.0113	3.0077

APPENDIX (Figures)

Approximating functions comparison

Question 1.1	Question 1.2	Question 3
		

Question 2.1	Question 2.2	True function
		

Question 1.1 (MLP)

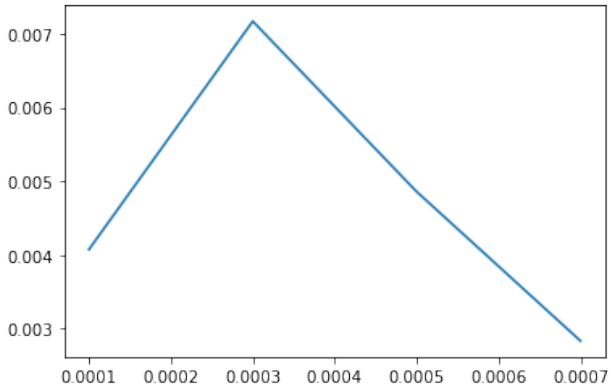


Figure 1: Evolution of the test error with rho

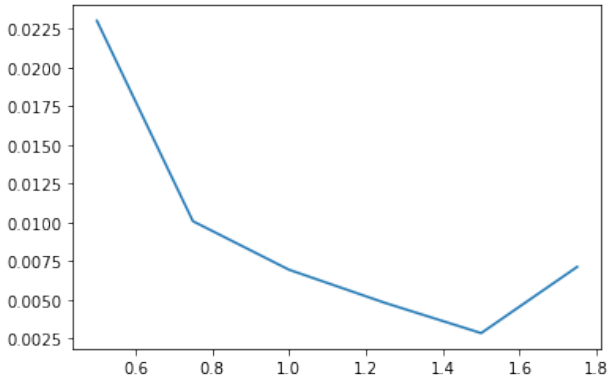


Figure 2: Evolution of the test error with sigma

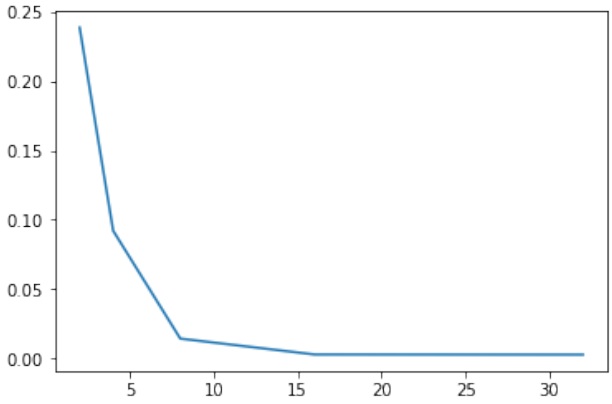


Figure 3: Evolution of the test error with N

Question 1.2 (RBF)

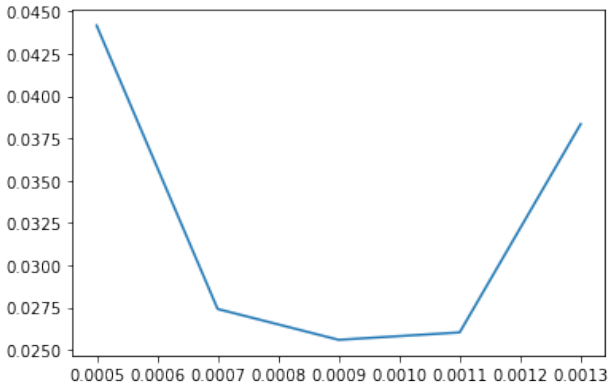


Figure 4: Evolution of the test error with rho

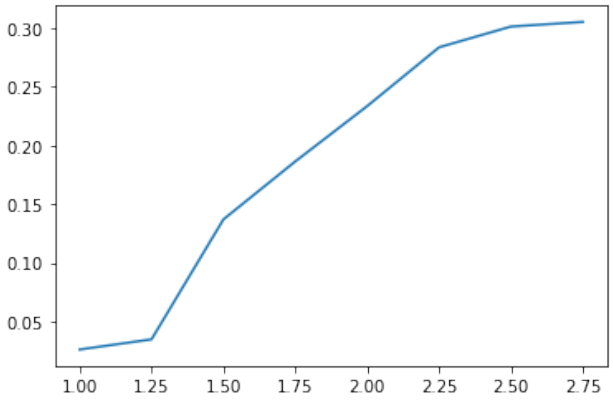


Figure 5: Evolution of the test error with sigma

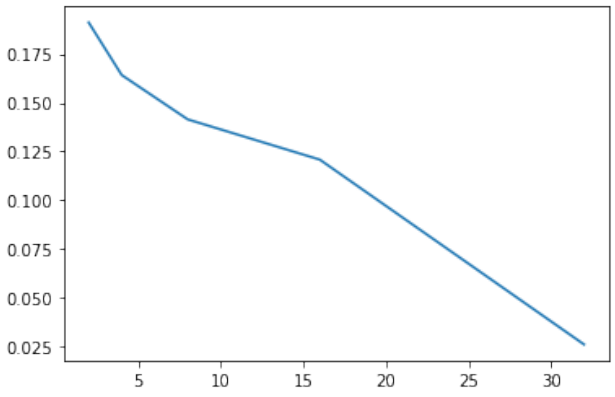


Figure 6: Evolution of the test error with N