OMML Project 1 - MLP and Generalized RBF Network

Alessandro Quattrociocchi - 1609286
Tansel Simsek - 1942297
Simone Fiorellino - 1960415

Question 1

	MLP	**RBF**
Optimisation Method	CG with Gradient-based	CG with Gradient-based
Parameter Setting	tol=1e-05	tol = 1e-05
Message	Optimization terminated successfully.	Optimization terminated successfully.
Starting value of objective function	2.917e+01	3.448e+00
Final value of objective function	6.494e-04	1.301e-03
Number of iterations	8541	1230
Number of function evaluations	13796	2331
Number of gradient evaluations	13796	2331
Norm of the gradient at the starting point	3.302e+01	3.036e+00
Norm of the gradient at the final point	4.085e-05	3.512e-05
Computational time(sec)	6.591	1.129
Initial training error	2.916e+01	3.447e+00
Final training error	2.942e-05	2.433e-04
Test error	1.153e-04	1.197e-03
Final Setting	ρ=1e-05, σ=1.05, N=40	ρ=1e-05, σ=1, N=60

The final choice for the hyper-parameters (ρ, σ, N) has been made by the grid search and k-fold cross validation. The hyper-parameter space is taken as the following for both MLP and RBF networks, ρ : [1e-5, 1e-4, 1e-3], N : [5, 10, 20, 30, 40, 50, 60, 70, 80], and σ : [.95, 1, 1.05]. We have used 5-fold cross-validation and run cross-validation 10 times for each combination in order to observe different samples in each fold and by doing that fairly measure the effects of hyper-parameters. Our main aim was to compare the average validation loss/error and average train error, and choose the parameters when these two measures' values get close to each other(difference between avg. train loss and avg. validation loss is the lowest). The information we have obtained can be listed as following:

1. MLP Network: Approximately, until 40 neurons(N), avg. validation and avg. train loss kept decreasing but after 40 neurons, while avg. train loss still decreasing, avg. validation loss started to increase(over-fitting). In this case, we have found the best N value as 40 in order to avoid very specific model that may memorise the train data and outperform in the unseen/test data.
   RBF Network: Very similarly, having 60 neurons(N) gave the best performance on the validation sets. After 60 neurons, some oscillations are observed on the avg. validation accuracy but almost never performed better than 60 neurons.

2. MLP & RBF Networks: In most of the cases, ρ equals 1e-3 and 1e-4 gave the highest avg. validation loss because of the great amount of penalisation factor, and was unable to capture the relationship between the input and output variables(under-fitting). Therefore, selecting ρ hyper-parameter as 1e-5 is found as the best choice to have a good interpretability of the data.

3. We haven't observed a huge difference between σ values, since they selected still close to the original σ value(1) that has been using mostly in the tanh function. While the best MLP Network is found when σ equals to 1.05, the best RBF Network is detected when σ equals to 1.

As a result, MLP Network performed better than RBF Network in terms of quality of approximation, since in MLP model both train and test error can be seen lower than RBF model. On the other hand, having very good quality of approximation for MLP cost higher computational time, almost 6 times more number of iterations, gradient and function evaluations. So, It can be seen that RBF Network has more time efficiency in optimisation.

The gradients for the MLP Network is,

$$\frac{dE}{dw} = \frac{1}{P}(\hat{y} - y)\left[v\frac{d(tanh(wx+b))}{dw}\right]x + \rho w = \frac{1}{P}(\hat{y} - y)\left[v\frac{4\sigma e^{2\sigma(wx+b)}}{(e^{2\sigma(wx+b)}+1)^2}\right]x + \rho w$$

$$\frac{dE}{db} = \frac{1}{P}(\hat{y} - y)\left[v\frac{d(tanh(wx+b))}{db}\right] + \rho b = \frac{1}{P}(\hat{y} - y)\left[v\frac{4\sigma e^{2\sigma(wx+b)}}{(e^{2\sigma(wx+b)}+1)^2}\right] + \rho b$$

$$\frac{dE}{dv} = \frac{1}{P}(\hat{y} - y)\left[v(tanh(wx+b))\right] + \rho v$$

The gradient ∇E for the RBF Network can be expressed in the form:

$$\nabla E(w,c) = \begin{pmatrix} \nabla_v E(v,c) \\ \nabla_c E(v,c) \end{pmatrix}$$

where $\nabla_v E$ and $\nabla_c E$ are the partial gradients with respect to v and c. The expression of these components are given below:

$\nabla_c E(v,c)$:

$$\nabla_{c_j} E(v,c) = v_j \frac{1}{P}\sum_{i=1}^{P}\left(\sum_{h=1}^{N} v_h \phi(||x^i - c_h||) - y^i\right)\nabla_{c_j}(||x^i - c_j||) + \rho c_j$$

where,

$$\nabla_{c_j}(||x^i - c_j||) = \frac{2}{\sigma^2} e^{-(\frac{||x^i - c_j||}{\sigma})^2}(x^i - c_j)$$

So we'll have

$$\nabla_c E(v,c) = \begin{pmatrix} \nabla_{c_1} E(v,c) \\ \vdots \\ \nabla_{c_N} E(v,c) \end{pmatrix}$$

$\nabla_v E(v,c)$:

$$\nabla_v E(v,c) = \phi(||x^i - c_j||)^T \cdot \frac{1}{P}(\phi(||x^i - c_j||)v - y) + \rho v$$

Question 2

	MLP	**RBF**
Solver	ECOS	ECOS
Problem Status	Optimal	Optimal
Number of iterations	14	12
Computational time(sec)	0.010s	0.008s
Training error	4.762e-3	1.207e-3
Test error	5.446e-3	2.584e-3
N, σ, ρ	40,1.05,1e-05	60,1,1e-05

In order to minimize the quadratic convex functions, It was used the CVXPY open-source Python-embedded modeling language. In particular as solvers, we used ECOS, a numerical software for solving convex second-order cone programs (SOCPs). In terms of iteration and time, it was the best of all tested open-source solvers. In this question, we are training fewer parameters and solving a problem with less complexity with respect to the first one. Indeed, in general, there will be a worst performance with a shorter computational time. Practically if we fix the parameters, w and b for the MLP and c for the RBF, with a proper configuration, then we could reach comparable performance w.r.t. Q1 in terms of test error, with shorter computational time.

- For the MLP we sampled the values of the parameter from a truncated normal. We repeat this procedure in order to have a better initialization of parameters w and b.
- Instead for the RBF we have implemented two different ways for the initialization of the centers:
  - Kmeans
  - Selection of the centers by randomly picking N of the P points of the training set. We repeated this selection several times in order to pick the best random choice.

  We obtain close performance using these two approach. Kmeans have a faster selection of the parameter c. For the final solution, we choose the "random sampling" solution.

Q1 vs Q2

	Q2MLP	**Q1MLP**	**Q2RBF**	**Q1RBF**
Number of iterations	14	8541	12	1230
Computational time	0.010s	6.591s	0.008s	1.129
Training error	4.762e-3	2.942e-05	1.207e-3	2.433e-04
Test error	5.446e-3	1.153e-04	2.584e-3	1.197e-03

In the case of the MLP, we observe a difference of one order of magnitude in the test error.
Instead, in the case of the RBF, the test errors have the same order of magnitude.
In conclusion, using the same hyper-parameters, the two approaches seems comparable in term of test error. Again, the main difference is the complexity of the model, we have fewer parameters and a more easy problem. So this allows us to reach a good approximation in less time with a good initialization of parameters.

Question 3

As we know from the table in exercise 1, the best performing network model is the MLP network, so we applied the two blocks decompositions to this last one. A natural two blocks splitting is respect to the non convex problem and the convex problem, minimizing the risk function $\min E(\omega_1, \omega_2)$ w.r.t (ω_1, ω_2), where the risk is defined as following:

$$E(\omega_1, \omega_2) = \frac{1}{2P} \sum_{p=1}^{P} (\omega_2^T g(\omega_1; x^p - y^p))^2 + \lambda_1 ||\omega_1||^2 + \lambda_2 ||\omega_2||^2 \tag{1}$$

The non-convex block remains non-convex and we have no methods other than iterative routines to minimize the cost function; on the other hand, it is possible to make some observations on the convex block.

Fixing the weights and biases (ω_1) we have that $g(\omega_1; x^p) = g_i(\omega_1; x^p)_{i=1,\dots N}$ and assuming the neurons output as $z_i(x^p) = g(w^T x^p)$ and $y(x) = v^T z^p$ the objective function can be written as :

$$E(v) = \frac{1}{2P} \sum_{p=1}^{P} (g(Ux^p)^T v - y^p)^2 + \lambda_2 ||v||^2 \tag{2}$$

This form contains a $P \times N$ hidden matrix G_{pi} so the risk can be assume the well know structure of a strictly quadratic problem.

$$E(v) = ||Gv - Y||^2 + \lambda_2 ||v||^2 = v^T (G^T G + \lambda I) v - 2Y^T G v + ||Y||^2 \tag{3}$$

We can summarise the decomposition algorithm as follows:

$$\omega_1^{k+1} = \omega_1^k - \eta \nabla_{\omega_1} E(\omega_1^k, \omega_2^k) \tag{4}$$

$$\omega_2^{k+1} = \omega_2^k - \eta \nabla_{\omega_2} E(\omega_1^{k+1}, \omega_2^k) \tag{5}$$

Accordingly with this setup we report some results obtained:

- Python optimization toolbox
  - **Block 1**: *scipy.optimize.minimize* with gradient evaluation, $tol = 1e-4$ and CG with Gradient-based.
  - **Block 2**: CVXPY optimization tool with ECOS solver.
- The total number of sub-problems solved is: 25.
- The number of function evaluations (evaluation of the non convex + evaluation of the convex block) is: 22904
- Total number of gradient evaluation is: 22652
- Time spent on optimisation (from the call to the end of the solver) is: 8.850s
- Training Error: 1.79×10^{-4}
- Test Error: 4.20×10^{-4}

Q2 vs Q3 MLP Network Comparison

	MLP Q2	**MLP Q3**
Train Error	4.762e-3	1.79e-4
Test Error	5.446e-3	4.20e-4
Execution time	0.010s	8.850s

This summary table shows how the two networks perform very differently. We can notice how the value of the validation loss is lower in the MLP Extreme Learning, although the plot is quite similar to that generated in 1.1 and to that of the ground truth. The other substantial difference is in the execution time, in fact if for the first case we need only a few tenths of a second, in the two block decomposition we are on the order of 10 seconds. Finally, the number of iterations carried out in the first case is in the order of tens, while function evaluations reaches thousands in the second case. The number of iterations is so high because it is related to the value of the tolerance set at 10^{-4}, min_delta $= 10^{-4}$ and patience $= 10$. This profound difference is due to the fact that in exercise 2.1, having fixed randomly the values of w and b, we solve a quadratic problem by going to optimize only v, giving up many parameters to be estimated. While in the case of the two block decomposition we are not omitting parameters to be estimated but we are only simplifying the problem of an optimisation in two sub-problems. For this reason the results are comparable to what was obtained in 1.1 more than in 2.1.

		settings N σ ρ	**Final training error**	**Final test error**	**Optimisation time(sec)**
Q1.1	**Full MLP**	40, 1.05, 1e-05	2.942e-05	1.153e-04	6.591
Q1.2	**Full RBF**	60, 1, 1e-05	2.433e-04	1.197e-03	1.129
Q2.1	**Extreme MLP**	40, 1.05, 1e-05	4.762e-3	5.446e-3	0.010
Q2.2	**Unsupervised RBF**	60, 1, 1e-05	1.207e-3	2.584e-3	0.008
Q3	**2-Block Decomposition**	40, 1.05, 1e-05	1.79e-4	4.20e-4	8.850
Q.3 Bonus	**Best model**	60, 1, 1e-05	5.067e-05	-	12.126

Note that: In question 3 setting seed will allow us to start from the same initial point, but later on gradient evaluation could add the randomness. Thus, the results can be slightly different on your local machine.

Exercise 1 Plots

MLP Networks Plot

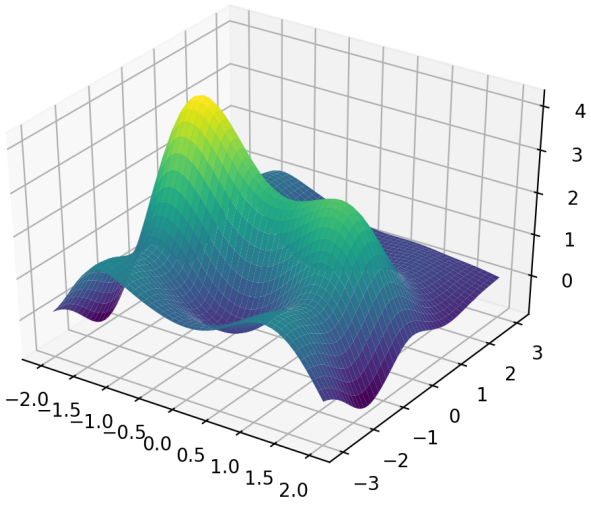

Figure 1. Plot 1.1

RBF Networks Plot

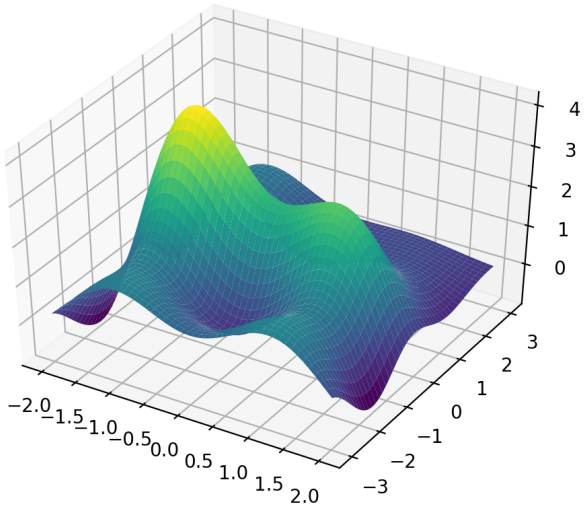

Figure 2. Plot 1.2

Exercise 2 Plots

MLP Networks Plot

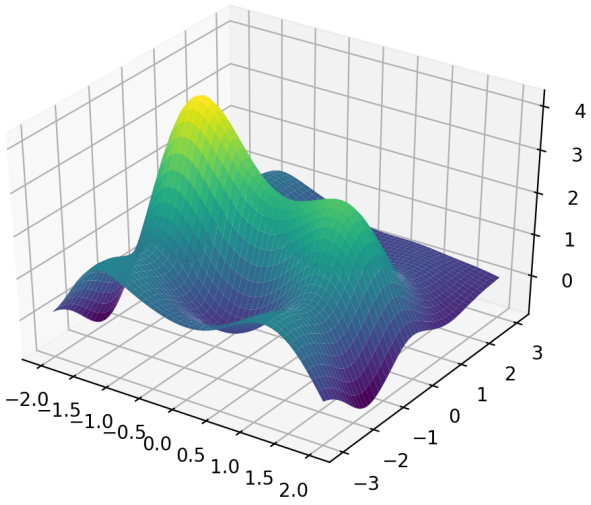

Figure 3. MLP Plot 2.1

RBF Networks Plot

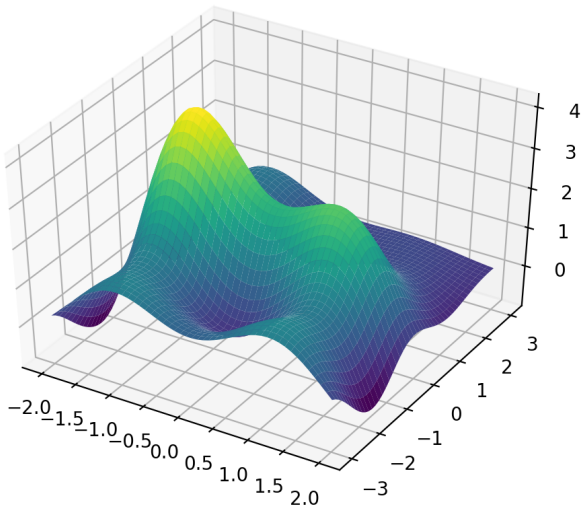

Figure 4. RBF Plot 2.2

Exercise 3 Plot

Two Blocks Decomposition MLP Networks Plot

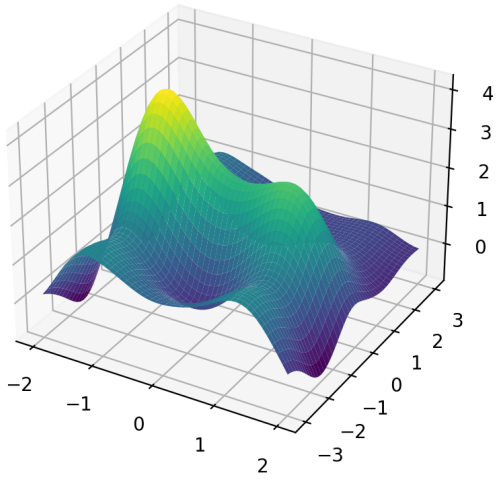

Figure 5. MLP Plot ex3