University of Rome "La Sapienza"

Optimization Methods for Machine Learning

Project 1

Malick Alexandre Ngorovitch Sarr, Vigèr Durand Azimedem Tsafack

November 24, 2018

INTRODUCTION AND SET UP

In this project, our goal was to implement neural networks for regression. Indeed, we were tasked to reconstruct in the region $[-2, 2] \times [-2, 2]$ a two dimensional function $\mathrm{F} : \mathbb{R}^2 \to R$ (graph at figure 4.1). We were not provided with the analytic expression of the function but we only had a dataset obtained by randomly sampling 200 points having the form

$$\left\{(x^p, y^p) : x^p \in \mathbb{R}^2, y^p \in \mathbb{R}, p = 1, ..., 200\right\}$$

The data set was split in a training and testing set separated with the ratio 75% (traning) 30% (test) with a seed of 1792126. On this data, 2 machine learning architectures were used and each network was optimized using 3 optimization techniques. The network architectures used were the Multilayer Perceptron (MLP) and the Radial Basis Function and the optimization techniques used are Full minimization, Two block method and Decomposition.

This project was made with Python along few of its libraries: numpy, Scipy, Pandas and Sklearn

1 Full Minimization

For the full minimization, we defined a shallow Feedforward Neural Network with only one hidden layer. We denote by π the hyper-parameters of the network settled by means of a heuristic procedure and ω the parameters to be settled by minimizing the regularized training error

$$E(\omega; \pi) = \frac{1}{2P} \sum_{p=1}^{P} \|f(x^p) - y^p\|^2 + \rho\|\omega\|^2 \tag{1.1}$$

where $f(x)$ is the model provided by the network, $\omega = (v,\, w,\, b)$ are the weights, $\pi = (N,\, \rho,\, \sigma)$ are the hyperparameters, P is the dimension of the training set. Full minimization is achieved by minimizing all the weights using gradient descent. We applied full minimization on two networks.

For the first part, We defined a shallow MLP with linear output:

$$f(x) = \sum_{j=1}^{N} v_j g\left(\sum_{i=1}^{n} w_{ji} x_i - b_j\right) \tag{1.2}$$

where v corresponds to the weights from the hidden to the output layer, w the weights from the input layer to the hidden layer, x the input data, b the bias n the number of input per neurons, N dimension of hidden layer, and g activation function, defined as:

$$g(t) = tanh\left(\frac{t}{2}\right) = \frac{1 - e^{-\sigma t}}{1 + e^{-\sigma t}} \tag{1.3}$$

The hyperparameters have been tuned with a K-fold cross validation (using sklearn.modelselection.KFold) to minimize them. The combination combinations of the hyper-parameters used are the following:

$$(N, \rho, \sigma) \in \Big\{1, 2, 3, 4, 16, 20, 30, 50\Big\} \times \Big\{10^{-4}, 10^{-5}\Big\} \times \Big\{1, 2, 3\Big\} \tag{1.4}$$

The training of the model was done using the bfgs solver with a maximum number of iterations set at 60000.The result showed us that the network with the parameters $N = 50$, $\rho = 10^{-5}$ and $\sigma = 1$ had the lowest error on the test. Using the MLP network with those parameters produced an error of 0.000835255 on the training set and had an error of 0.004334223 on the test set. Analyzing the figure 4.2 , we realize that at $N < 16$, we have a diminishing error on both the training and the test error. At $N > 16$ we observed that both train and test error reach their optimal state, as both errors, reach a plateau. This means that the difference between the errors at $N = 16$ and $N = 50$ are so close, that the network would produce approximatively the same result. Additionally, underfitting trends may happen on when $N < 16$ but overfitting almost never happen as the difference between the train and test errors are quiet insignificant. Lastly, the motivation in picking an MLP of $N = 50$, instead of $N = 16$, is related to point 2. Indeed, the network of 50 nodes took 3 hours to train as opposed to 40 mins for a network with 16 nodes. That choice was motivated by the fact that increasing the number of nodes improved the result for the extreme learning. More on that in the next section.

Full minimization was also done on the Radial Basis Function network. The error function was the same at 1.2 and the model was defined as:

$$f(x) = \sum_{j=1}^{N} v_j \phi(\|x^i - c_j\|) \tag{1.5}$$

where v weights from the hidden to the output layer c_j is the center of the j-th hidden neuron and ϕ is the activation function, defined as:

$$\phi(\|x - c_j\|) = e^{-(\|x - c_j\|/\sigma)^2} \tag{1.6}$$

with $\sigma > 0$.

The hyperparameters have also been tuned with a K-fold cross validation (using sklearn.modelselection.KFold) in order to minimize them. The combination combinations of the hyper-parameters used are the following::

$$(N, \rho, \sigma) \in \Big\{1, 2, 3, 4, 16, 20, 30, 50\Big\} \times \Big\{10^{-4}, 10^{-5}\Big\} \times \Big\{1, 2, 3\Big\} \tag{1.7}$$

The training of the model was done using the bfgs solver with a maximum number of iterations set at 60000.The result showed us that the network with the parameters $N = 50$, $\rho = 10^{-5}$ and $\sigma = 1$ had the lowest error on the test. Using the RBF network with those parameters produced an error of0.001876159 on the training set and had an error of 0.006393212 on the test set. The figure 4.2 shows us that similarly as the MLP that at $N < 20$, we have a diminishing error on both the training and the test error. At $N > 20$ we observed a small increase then decrease of both errors. A slight increase on the test set was observed at $N = 30$ which could have been a sign of overfitting. But the increase was insignificant and the error dropped again at $N > 40$.

Comparing both Models, at $N = 50$,the MLP took 10691 seconds to optimize and to train whereas the RBF took 10588 seconds, so both model had approximatively the same running speed. However, much more function evaluation where made in the MLP with 246844 as opposed to 158688 for the RBF. In both models, underfitting happens when N is very low, ergo there aren't enough neuron in the hidden layer to train the network. Over-fitting almost never happens as we observed a plateau past the optimum points and the difference between the errors on the train set and test set are close to each other. No trend of increasing test error while decreasing train error was observed. The function representing the approximating function obtained by the MLP and by the RBF networks is represented at fig. 4.1

2 Two Blocks Method

The idea behind the extreme learning procedure, was derived to exploit the structure $minE(\omega_1, v)$ with $\omega_1 \in R^q$ and $v \in R^N$. It is a two step process, first the weights are fixed randomly (or by procedural guess) on ω_1^0, then we choose v^0 by solving to global optimality the convex linear least square (LLSP).

$$min \sum_{p=1}^{P} ||v^T g(\omega_1^0, X^P) - y^P|| + \lambda ||v||^2$$

This technique speeds up the training of the network. For our convenience, we were tasked of using the same hyperparameters used in the previous section. Ergo, we performed extreme learning training on both MLP and RBF through an optimization process using the input data and minimizing the error for both MLP and RBF.

Regarding the MLP network, all the weights w_{ji} and the biases b_j, with $j \in \{1, ..., N\}$ and $i \in \{1, ..., n\}$, have been set up randomly. Then v was optimized following the above principle using the scipy.optimize.minimize tool. We ran it using the bfgs solver with a maximum number of iterations set at 60000. The training error obtained in here is 0.122384356 while the testing

error is 0.184854404. In our case, even though the extreme learning performed much faster than the full optimized version, it was less accurate with a test error of .18. Training took roughly 8944 sec. as opposed to the 10691 sec. that the full-MLP took. The number of function evaluations were reduced as well, with the 2774 as opposed to the 246844 function evaluation of the fully optimized one.

Regarding the RBF network, the centers c_j with $j \in \{1, ..., N\}$ were chosen using the Sklearn Kmeans built in method. The minimization was applied to the weights v. The train error obtained here is 0.00930792 where as the test error obtained using extreme learning is 0.015193839 which is slightly worse than the one using full minimization 0.006393212, yet the running time for the random selection of centers was way faster than the fully minimized one; 966 sec. as opposed to 10587 sec. The mimization was done with Scipy.optimize.minimize, with a maximum iteration of 60000 with the bfgs solver. The number of function evaluation was cut down too with 13624 function evaluation for the extreme learning compared to the 158688.

To conclude, in term of performances, both model ran faster and used less resources as the models run in the first section. This models allowed us to train much faster and could potentially give the opportunity to train either with larger amount of nodes, or to train to get quick results by reducing the training time. One trade-off using this technique is that it is slightly less accurate that the fully optimized one.

3 Decomposition Method

This section describes the results obtained by using two block decomposition method on an MLP network. The idea behind it is that starting with $(\omega_1^0, v^0, b^0$, we alternate further optimization steps in ω, b and v. Those alternated optimization were ran the following way. First we fixed ω and b randomly and we minimized with respect to v, then we minimized with respect to ω and b using v_new (which correspond to the values of v on the previous step). This paradigm was run k times until the error is = 0 or an early stopping procedure was encountered. The conditions for an early stopping are: the weights stop changing or the error on the test set decreases for 4 successive iterations.

The decomposition method took about 3496 sec to run. It outperformed the previous models used having a train error of 0.00274962 and a test error of 0.004083175. Like the previous models, minimization was done with Scipy.optimize.minimize, with a maximum iteration of 60000 with the bfgs solver This model was run with $N = 16$, and it took about 3496 sec to run. In terms of performance, this model ran slower than the others, keeping in mind that it was trained with only 16 Neuron. However, it outperformed the other models in term of error minimization at it is the "best model" out of the 5 used for out tests.

.

results

Question	Typw Network	N	σ	ρ	Training Error	Test Error	Time
1.1	Full MLP	50	1	10^{-5}	0.000835255	0.004334223	10691s
1.2	Full RBF	50	1	10^{-5}	0.001876159	0.006393212	10587s
2.1	TBM MLP	50	1	10^{-5}	0.122384357	0.184854404	8944s
2.2	TBM RBF	50	1	10^{-5}	0.00930792	0.015193839	966s
3	decomp. MLP	16	1	10^{-5}	0.00274962	0.00408317	3497s

Table 4.1: Results presented in the above section. The training error computed following formula presented in the assignment 1.1 on the training set.

4 Annexes

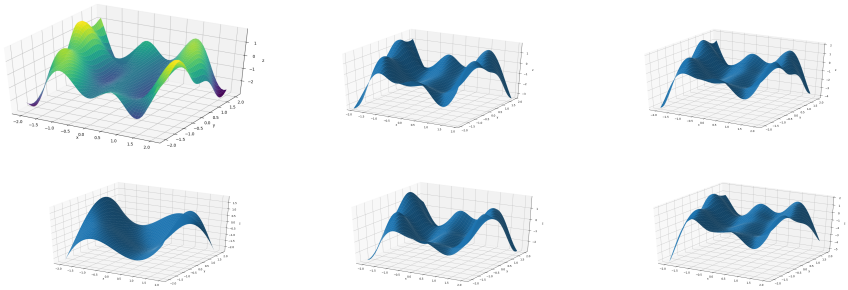

Figure 4.1: left to right: the function plot provided, Full MLP, Full RBF, TBM MLP, TBM RBF and decomposition. (TBM= Two Block Method) The hyperparameters used for each network are specified in table 4.1.

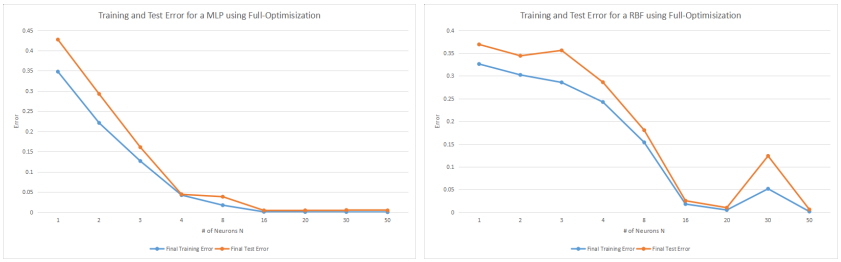

Figure 4.2: In order: Error obtained varying the number of Neurons for both MLP and RBF 4.1.