# Optimization method for machine learning : Homework 1

Guadagnino Tiziano, Paoli Andrea, Alfano Domenico

December 3, 2017

## Question 1 : Full optimization

In the first part we deal with the full optimization problem for both a shallow feed-forward neural network and a radial basis function network; in both cases the training problem can be expressed as :

$$\theta^*, v^* = \arg\min_{\theta,v} \frac{1}{2P} \sum_{p=1}^{P} \|f(x^p; \theta, v) - y^p\|^2 + \rho \cdot (\|\theta\|^2 + \|v\|^2) \tag{1}$$

where $P$ is the number of training examples, $\rho$ is the regularization factor, $v$ represents the vector of output weights and $\theta$ represents the vector of hidden layer weights for the multi-layer perceptron or the set of centers for the radial basis function network.[1]

In both the MLP and the RBF network we use the L-BFGS algorithm (implemented in **scipy.optimize**) for solving the optimization problem. Unfortunately, this method does not return the number of gradient evaluation done during the optimization process; for this reason, in the reported results, the number of gradient evaluation are omitted.

We can see L-BFGS as a lighter version of the standard Quasi-newton method BFGS. The main difference is that the BFGS method compute and store at each step the whole inverse Hessian matrix approximation $\hat{H}^{-1}$ using the sequence of vectors $\{s_k\}$ and $\{y_k\}$ that represents the difference in gradient and in the variable in every step: this require $O(n^2)$ space in memory, where $n$ is the dimension of the variable. Instead L-BFGS compute and store the inverse Hessian approximation $\hat{H}^{-1}$, using only the last $k$ vectors of the sequences, in such a way that the matrix can be stored in $O(n)$ space. In a certain sense L-BFGS try at each step to approximate the corresponding BFGS step, however, even if the method loss some convergence speed, can perform many more iterations in a given amount of time, so in certain cases L-BFGS can converge even faster than BFGS.

[1] In the following we refer to $\theta$ as the set of variables in the hidden layer, so hidden layer weights for the MLP and centers for the RBF.

For performing the grid search we focus on the structural risk minimization principle, so essentially we don't consider only the MSE result on the test set but also some measure of complexity of the learning algorithm. Our choice, even if it is a theoretically incorrect measure of the VC confidence, is the number on hidden units in both network, so in some sense we prefer solution that are also easy to reach from a computational point of view.

## Shallow feed-forward neural network

In all the following points regarding the MLP we implement the network using *Tensorflow* library, that automatically compute the gradient symbolically when we use the **scipy** optimizer interface inside it. Even if this library use parallel GPU computation in all the mathematical operation the number of parameter that characterize the models are so small that we can directly compare the results (in terms of computational time) with the one of RBF network implemented in *numpy*. The output of the MLP can be written as :

$$f(x^p; v, \theta) = \sum_{i=1}^{N} v_i \cdot g(\theta_i^T \cdot x^p) \tag{2}$$

where $g(\cdot)$ is the tanh function and $\theta_i$ is the vector of hidden layer weights for the $i$-th neuron.
In the figure we can see a comparison with different setting of the parameter, in particular in the figure [1] we fix the number of units to $N = 5$ and use two different settings for $\rho$. In the first figure we can see the effect of a relative high value of the regularization factor ($\rho = 10^{-3}$), where essentially the norm of the weights vectors is forced to stay near zero, limiting the range of values that this weights can reach during the optimization process. As effect we can see a strong under-fitting in the resulting approximation function.

In figure [2] we do exactly the opposite fixing the value of $\rho = 10^{-5}$ varying the number of units. With $N = 3$ we can see the effect of under-fitting, where the approximator does not have enough expression power for determine the "shape" of the target function. This is due to the few number of units used in the network, in fact we can see that with $N = 20$ we can really improve the approximation. On the other hand if we increase to much $N$ we can face the problem of over-fitting, getting an even worser approximation function, as we can see in the case of $N = 100$.

After a long grid search (using the principle of structural risk minimization) we find that the best configuration for the hyper-parameters that gives a reasonable mean square error on the test set is $N = 20$ and $rho = 10^{-5}$ (we use the standard tanh function implemented in *Tensorflow*, so we don't need to tune the $\sigma$ parameter in this case).

## Radial basis function network

The output of the RBF network can be written as :

$$f(x^p; v, \theta) = \sum_{i=1}^{N} v_i \cdot \Phi(\|x^p - \theta_i\|; \sigma) \tag{3}$$

where $\Phi(\cdot; \sigma)$ is a Gaussian function parametrized by $\sigma$, and $\theta_i$ is the $i$-th center.
In the figure [3] we can see the effect of varying the value of $\sigma$ fixing the value of $N$ and $\rho$ ($N = 5$ and $\rho = 10^{-5}$). Essentially this parameter control the shape of our approximator, and in particular we can see it as a bandwidth for the gaussian function that determine for which range of input value we have a non-zero output.

In the figures [4] and [5] we use the same settings of the parameters $N$ and $\rho$ used for the MLP, fixing the value of $\sigma$ to 0.3 (figures [2] and [3] for the MLP). In this case we can see that the RBF is less influenced by the value of both $\rho$ and $N$, in particular the approximator seems to preserve more or less the same shape. This is probably due to the fact that both the target function and the output of the RBF are a weighted sum of exponentials. After the grid search we find that the best setting of the hyper-parameters is $N = 10$, $\rho = 10^{-5}$ and $\sigma = 0.3$.

## Comparison

The final results for the two model are reported in the 1.

| Model | MSE train | MSE test | Function Eval. | Gradient Eval. | time |
|---|---|---|---|---|---|
| MLP | $2.83 \cdot 10^{-3}$ | $3.36 \cdot 10^{-3}$ | 1055 | n.a | 1.49 s |
| RBF | $1.74 \cdot 10^{-3}$ | $2.88 \cdot 10^{-3}$ | 848 | n.a | 0.95 s |

Table 1: Results of the full optimization problem.

As we can see from the result both network perform pretty well in terms of both MSE and computational time. One consideration that can be done is that the RBF require a lot less parameter for reaching a reasonable approximation. Our explanation is that, as mentioned in the previous section, both the target function and the model are weighted sum of exponentials, so belongs to the same class of functions.

A comparison between the two approximation function and the Frank function is reported in figure [6].

# Question 2 : Two blocks methods

In the following two section we face the training problem using block decomposition methods, where essentially we optimize individual subset of the variables alternately. This subdivision of the variables in the objective can, in certain cases, reveal a special structure in the problem as in the case of the MLP and the RBF. For both architectures it is natural to divide the variables in the two blocks $v$ and $\theta$, and in particular the optimization of the objective become a linear least square problem with respect to the output weights $v$ and a non-linear least square problem with respect to $\theta$ reduced in the dimensionality with respect to the full training problem.

In this section in particular we fix the hidden variables $\theta$ using some criteria and optimize only the output weights $v$. For the fact that in this case we have a linear least square problem we choose to use the Conjugate gradient algorithm implemented in **scipy.optimize**, that have a time and space complexity that is related only to the variable dimension and not to the size of the dataset. This gave to the algorithm an high efficiency for this type of problem considering also that we have, for both MLP and RBF, a simple model with only few output weights.

In this case the problem can be defined as :

$$v^* = \arg\min_v \frac{1}{2P} \sum_{p=1}^{P} \|f(x^p; \theta, v) - y^p\|^2 + \rho \cdot \|v\|^2 \tag{4}$$

## Shallow feed-forward neural network : Extreme Learning

Using the same value of the hyper-parameters found using the grid search we obtain the approximation function shown in figure [7] in comparison with the one of the full optimization problem. As we can see, there is a strong under-fitting effect in the function obtained, due to the fact that, considering only the output weights, the learning algorithm losses expression power due to the reduced number of variables. In particular the hidden layer with fixed weights (sampled from a uniform distribution) can be considered as a feature manipulation step that essentially does not contribute much to the quality of the approximator.

## Radial basis function network : Unsupervised Selection of the Centers

For the RBF we choose, as unsupervised procedure for the center selection, the K-Means algorithm. We can see the result in figure [8], even in this case the same consideration done for the MLP holds.

## Comparison

The final results for the two model are reported in the table [2].

| Model | MSE train | MSE test | Function Eval.s | Gradient Eval. | time |
|---|---|---|---|---|---|
| MLP | $2.8 \cdot 10^{-3}$ | $3.36 \cdot 10^{-3}$ | 1055 | n.a | 1.49 s |
| MLP Extreme learning | $7.01 \cdot 10^{-3}$ | $7.98 \cdot 10^{-3}$ | 551 | 540 | 0.48 s |
| RBF | $1.74 \cdot 10^{-3}$ | $2.88 \cdot 10^{-3}$ | 848 | n.a | 0.95 s |
| RBF Unsuper. Centers | $4.59 \cdot 10^{-3}$ | $4.45 \cdot 10^{-3}$ | 582 | 83 | 0.51 s |

Table 2: Comparison table between full optimization methods and two-block decomposition methods.

Looking at the results we can state that this methods are really cheap from a computational point of view, in terms of both function/gradient evaluation and time needed. Even if in terms of mean square error both model perform really poorly this method must be considered as an alternative when we need just a fast approximation of a function.

# Question 3 : Decomposition method

For the last part we essentially alternate the optimization on $v$ and on *theta*: in this case, in a single iteration of the method, we solve in sequence two optimization problems:

$$v^{k+1} = \arg\min_{v} \frac{1}{2P} \sum_{p=1}^{P} \|f(x^p; \theta^k, v) - y^p\|^2 + \rho \cdot \|v\|^2 \tag{5}$$

$$\theta^{k+1} = \arg\min_{\theta} \frac{1}{2P} \sum_{p=1}^{P} \|f(x^p; \theta, v^{k+1}) - y^p\|^2 + \rho \cdot \|\theta\|^2 \tag{6}$$

starting from an initial guess $\theta_0$ of the hidden variables.

We choose to use this method for both the MLP and the RBF using a combination of the optimization routines of the previous sections, so the Conjugate gradient algorithm for solving the linear least square problem (Eq. [5]) and the L-BFGS algorithm for the non-linear problem (Eq. [6]).

We define the early stopping rule for the method as follow:

$$\|\nabla f(x; v, \theta)\| < \epsilon \tag{7}$$

where the $\epsilon$ is a threshold parameter that we tune heuristically.

Futhermore every time we do an iteration (one optimization step on the output variables $v$ and one on $\theta$) we reduce the value of the *g-tolerance* in both the optimization routines, that essentially have the same meaning of the $\epsilon$ in the equation [7] and have the effect to increase the accuracy of the algorithms over the iterations.

## Shallow feed-forward neural network

We can see the result of the optimization in the figure [9] in comparison with Extreme learning case. As we can see the block decomposition method outperforms completely the randomized choice of the hidden weights due to the extra amount of parameters that this method consider. In particular, using the stopping criteria ( Eq. [7] ), only two iteration of the whole optimization are needed to reach a reasonable solution.

## Radial basis function network

The result of the optimization is showed in figure [10]. In particular in this case we need just 1 iteration to complete the optimization.

## Comparison

| Model | MSE train | MSE test | Num. Function Evaluations | Gradient Eval. | time |
|---|---|---|---|---|---|
| MLP E. learning | $7.01 \cdot 10^{-3}$ | $7.98 \cdot 10^{-3}$ | 551 | 540 | 0.48 s |
| MLP Block Dec. | $2.46 \cdot 10^{-3}$ | $2.74 \cdot 10^{-3}$ | 2424 | n.a | 3.29 s |
| RBF Unsup. Centers | $4.59 \cdot 10^{-3}$ | $4.45 \cdot 10^{-3}$ | 582 | 83 | 0.51 s |
| RBF Block Dec | $1.34 \cdot 10^{-3}$ | $2.39 \cdot 10^{-3}$ | 1082 | n.a | 2.04 s |

Table 3: Comparison table between two block decomposition method and decomposition method.

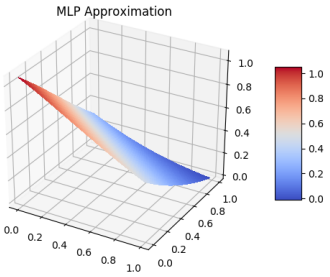
As we can see from table 3, the block decomposition optimization outperform the "fixed hidden variables" method in terms of mean square error on the test set paying the price of an higher computational cost.

In our case in particular the most performing strategy is the full optimization, as we can see in the final table. This is due to the fact that in the end the number of parameters for both model is really small and so the block decomposition method does not introduce any advantage in terms of computational time.

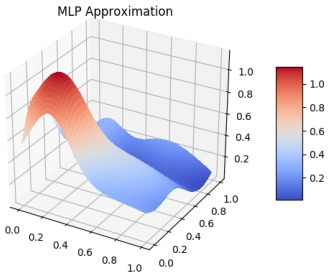
# Results

| Ex | FNN | settings | Training error | Test error | Optimization time |
|---|---|---|---|---|---|
| Q 1.1 | MLP | $N = 20$, $\sigma = 1$, $\rho = 10^{-5}$ | 0.002617 | 0.003826 | 1.49 s |
| Q 1.2 | RBF | $N = 5$, $\sigma = 0.3$, $\rho = 10^{-5}$ | 0.001331 | 0.003630 | 0.95 s |
| Q 2.1 | MLP | $N = 20$, $\sigma = 1$, $\rho = 10^{-5}$ | 0.006073 | 0.008651 | 0.52 s |
| Q 2.2 | RBF | $N = 5$, $\sigma = 0.3$, $\rho = 10^{-5}$ | 0.001905 | 0.003648 | 0.71 s |
| Q 3 | MLP | $N = 20$, $\sigma = 1$, $\rho = 10^{-5}$ | 0.002413 | 0.002460 | 1.08 s |
| Q 3 | RBF | $N = 5$, $\sigma = 0.3$, $\rho = 10^{-5}$ | 0.001236 | 0.002405 | 1.90 s |

# Plots Question 1

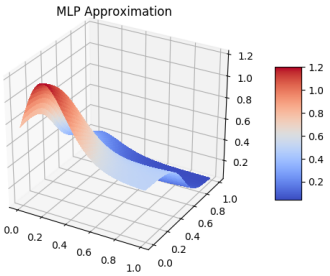



(a) MLP : $N = 5$ $\rho = 10^{-3}$

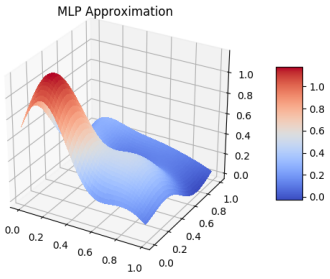



(b) MLP : $N = 5$ $\rho = 10^{-5}$

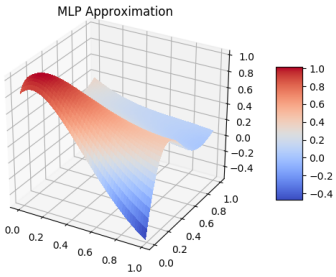
Figure 1: Comparison among different value of $\rho$.




(a) MLP : $N = 3$ $\rho = 10^{-5}$

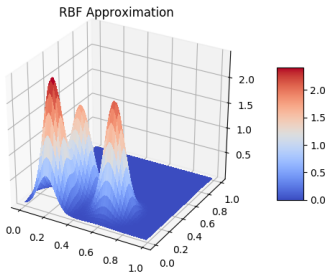



(b) MLP : $N = 20$ $\rho = 10^{-5}$

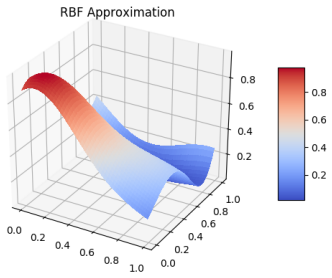



(c) MLP : $N = 100$ $\rho = 10^{-5}$

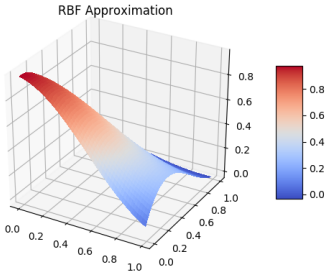
Figure 2: Comparison with different number of units.




(a) RBF : sigma = 0.1

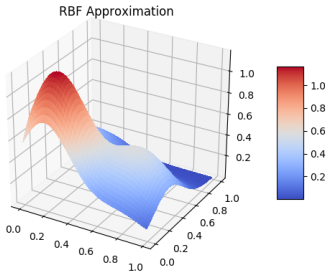



(b) RBF : sigma = 0.5

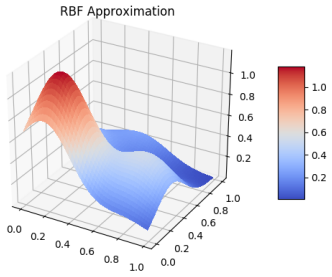



(c) RBF : sigma = 1

Figure 3: Comparison with different value of *sigma*.

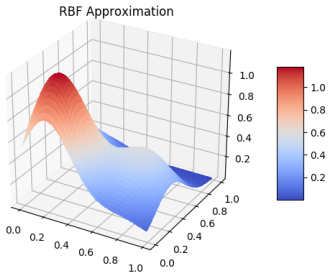



(a) RBF : $N = 5$ $\rho = 10^{-3}$

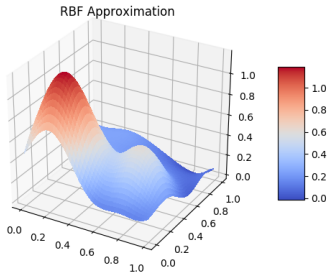



(b) RBF : $N = 5$ $\rho = 10^{-5}$

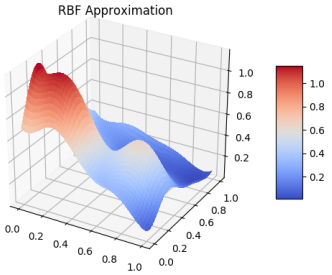
Figure 4: Comparison among different value of $\rho$.




(a) RBF : $N = 3$ $\rho = 10^{-5}$

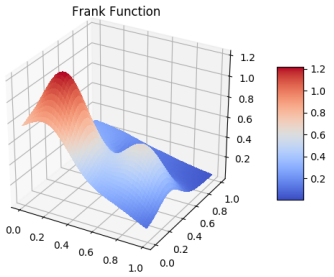



(b) RBF : $N = 20$ $\rho = 10^{-5}$

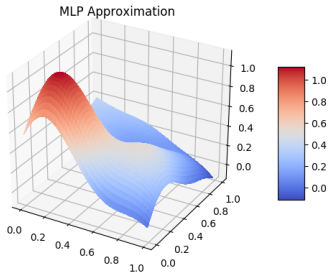



(c) RBF : $N = 100$ $\rho = 10^{-5}$

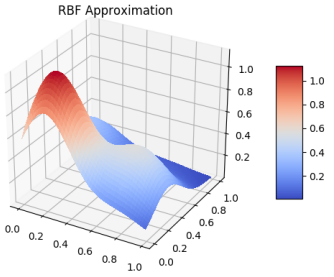
Figure 5: Comparison with different number of units.




(a) Frank function



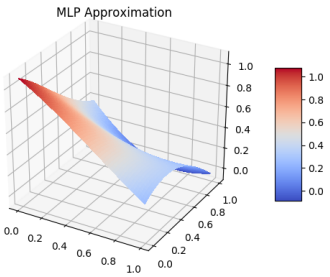

(b) MLP approximation



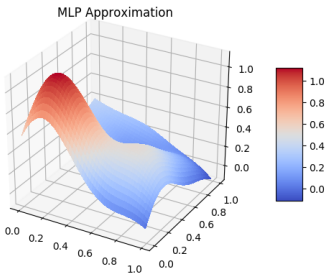

(c) RBF approximation

Figure 6: Question 1 results.

# Plots Question 2

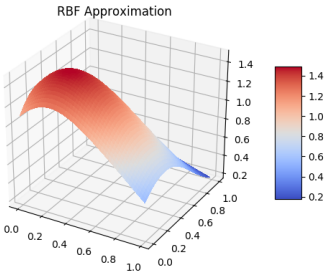



(a) MLP extreme learning

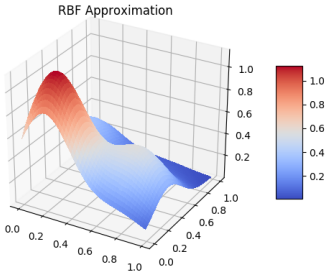



(b) MLP full optimization

Figure 7: MLP : Two block decomposition vs Full optimization.



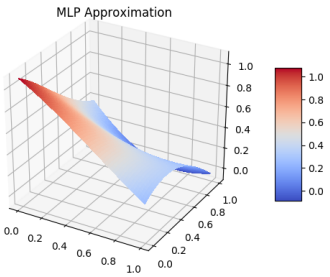

(a) RBF Unsupervised Selection of the centers



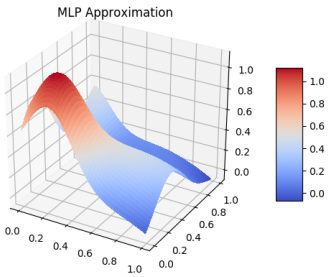

(b) RBF full optimization

Figure 8: RBF : Two block decomposition vs Full optimization.

# Plots Question 3

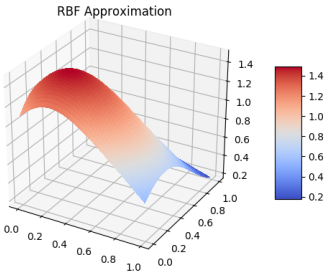



(a) MLP extreme learning

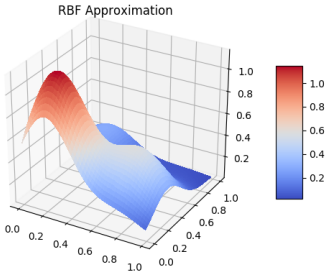



(b) MLP block decomposition

Figure 9: MLP : Two block decomposition vs Decomposition.




(a) RBF Unsupervised Selection of the centers




(b) RBF block decomposition

Figure 10: RBF : Two block decomposition vs Decomposition.