POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Optimal feature rescaling in machine learning based on neural networks

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Federico Maria Vitrò**

Student ID: 927682
Advisor: Prof. Lorenzo Mario Fagiano
Co-advisors: Ing. Marco Leonesio
Academic Year: 2021-22

Abstract

The use of machine learning and deep learning algorithms in many industrial fields is becoming an established reality day by day. In particular, the study of neural networks is still in constant evolution.

The goal of this thesis is to outline a method aimed at improving the prediction performances of a neural network in order to solve regression problems. The method described and tested in the thesis is called Optimal Feature Rescaling (OFR). This technique consists in multiplying the dataset variables with parameters chosen in an optimal way by solving a global optimization program through the use of a genetic algorithm. The OFR method is useful when the training problem is non-convex, since it tries to avoid the trap of local minima.

To test the OFR technique, a case study was taken into consideration: the roundness prediction in a centerless grinding machining operation. The tests carried out showed positive results demonstrating how, by re-scaling the variables in an optimal way, it is actually possible to obtain better prediction performances.

The non-convexity of the training problem is demonstrated by the fact that different rescaling factors lead to different network performances, i.e. different local minima of the validation error, while in convex problems each local minimum is also global.

The proposed method compensates the non-convexity of the training procedure. Then, if desired, any feature scaling can be recovered by modifying the first layer weights of the considered neural network. In this thesis, the parameters obtained through the OFR technique were used so that the network is able to implicitly perform the rescaling of the variables during the prediction phase.

Keywords: optimal rescaling, neural network, global optimization, non-convex

Abstract in lingua italiana

L'utilizzo di algoritmi di machine learning e deep learning sta diventando giorno dopo giorno una realtà affermata in molti settori. In particolare, lo studio delle reti neurali è tutt'oggi in continua evoluzione.

Lo scopo di questa tesi è di delineare un metodo volto a migliorare le performance di predizione di una rete neurale al fine di risolvere problemi di regressione. Il metodo descritto e testato nella tesi prende il nome di Optimal Feature Rescaling (OFR). Tale tecnica consiste nel moltiplicare le variabili del dataset con dei parametri scelti in modo ottimale risolvendo un programma di ottimizzazione globale mediante l'utilizzo di un algoritmo genetico. Il metodo OFR è utile quando il problema di addestramento è non-convesso, in quanto cerca di evitare la trappola dei minimi locali.

Per testare la tecnica OFR è stato preso in considerazione un caso di studio: la predizione della rotondità dei pezzi in uscita da un processo di rettifica senza centri. I test effettuati hanno portato risultati positivi dimostrando come, ri-scalando in modo ottimale le variabili, si riesca effettivamente ad ottenere migliori performance di predizione.

La non-convessità del problema di addestramento è dimostrata dal fatto che fattori di riscalamento diversi portano a diverse prestazioni della rete, ovvero a diversi minimi locali dell'errore di validazione, mentre nei problemi convessi ogni minimo locale è anche globale.

Il metodo proposto compensa la non-convessità della procedura di addestramento. Quindi, se lo si desidera, qualsiasi ridimensionamento delle variabili può essere recuperato modificando i pesi del primo livello della rete neurale considerata. In questa tesi, i parametri ottenuti attraverso la tecnica OFR sono stati utilizzati in modo che la rete sia in grado di eseguire implicitamente il rescaling delle variabili durante la fase di predizione.

Parole chiave: scalamento ottimo, rete neurale, ottimizzazione globale, non-convesso

Contents

List of Acronyms

Acronymous	**associated words**
OFR	Optimal Feature Rescaling
FFNN	Feed Forward Neural Network
ES	Early Stopping
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
R^2	Coefficient of determination
H	Hypothesis space
h	Chosen hypothesis (model)
L	Loss function
PCA	Principal Component Analysis
GA	Genetic Algorithms
CV	Cross-validation
std	Standardized
CV Mean	Average of the cross-validated performances
CV Std	Standard deviation of the cross-validated performances

1 | Introduction and goals

The advent of Industry 4.0 has radically changed the way companies manufacture, improve and distribute their products. Nowadays the intertwining between automation and data exchange is becoming a common practice for the manufacturers and in a world where resources are getting scarcer day by day we all need to do more with less. In particular, artificial intelligence and machine learning are playing an increasingly central role in this landscape where the physical and virtual worlds are fused together. Similar trends are observed in many other industrial sectors.

As of today, neural networks are the basis of many of the most successful algorithms in machine learning. These networks, which try to emulate the human brain, are used to solve many prediction problems, e.g. to predict stock market prices, the effect of an actuation in robotics, etc. and for this reason they have been taken into consideration during this thesis.

1.1. Problem definition

Given a regression problem and a Feed Forward Neural Network (FFNN), this thesis focused on increasing the neural network performances via problem variables rescaling. This technique consists of multiplying the dataset features with a set of rescaling parameters which were chosen in an optimal way by solving a global optimization problem.

For this reason, this technique was called Optimal Feature Rescaling (OFR). Throughout the thesis a metaheuristic approach was used in order to find the best parameters. Typically, the loss function of a global optimization problem is non-convex. Therefore, searching for a better minimum via feature rescaling is an interesting and not trivial topic.

OFR method was then applied in a real scenario to verify its effectiveness: the roundness prediction in a centerless grinding machining operation.

1.2. Centerless grinding

Centerless grinding is a machining process characterized by a high degree of difficulty in predicting the quality of the machined parts based on process parameters. The classical machine learning-based approach, even enhanced by physics-based features embedding some apriori knowledge, achieved non-satisfactory prediction performances. As the quality of the worked part mainly depends on the choice of process parameters, e.g. work piece height, feed velocity, etc., which are inhomogeneous in terms of absolute numerical values, an "Optimal Feature Rescaling" was investigated in order to make the roundness of the worked piece more predictable.

1.3. Thesis structure

In the second Chapter of this thesis, the OFR approach is described in-depth by providing all the tools needed to understand it, and a brief overview of the concepts behind regression problems and FFNN are given. Furthermore, the method used to solve the global optimization program (genetic algorithm) is presented.

In the third Chapter, the centerless grinding problem is introduced: a description of the process variables are provided, the roundness prediction problem is formulated as a regression problem, and all the results, obtained by applying Optimal Feature Rescaling, are presented.

Lastly, conclusions and future development proposals are presented.

1.4. Original contributions

Although neural networks are nowadays a mature and widely used technique in the field of machine learning and deep learning, they are still a topic of study, in particularly with regard to the training phase. In fact, the search for the optimal network weights is one of the key elements to obtain better performances. For this reason, alongside the more common backpropagation technique, several global optimization algorithms have been tested in order to globalize the neural network training [12][13][6].

On the other hand, variable scaling is a well-established technique used during the preprocessing phase of many machine learning algorithms, e.g. the neural networks. In fact, rescaling the variables leads to an improvement in the efficiency of training neural networks [1].

Regarding the considered case study, i.e. the roundness prediction in a centerless grinding machining operation, several physics-enhanced machine learning techniques have been applied. By testing these, it was noted that the use of hybrid models led to better performances [11].

Among these hybrid models, a new approach has recently been tested. It exploits both the prediction of a physics-based simulation model and a reduced set of experimental data for a data-driven correction. The approach relies on a hierarchical semi-supervised classification, where the training data, classified on the basis of the three quality intervals of interest, are divided in a certain number of sub-clusters with respect to the process input parameters (primary features) and enhanced with the classification prediction provided by a physics-based model (apriori knowledge injection). These sub-clusters are then used in the prediction phase, either directly or through a support vector machine predictor. This approach is called Semi-Supervised Physics-Informed Classifier (SPIC) and, as in the previously mentioned cases, it achieves better performances than using non-hybrid techniques [8].

In the end, throughout the thesis, the main pointers to the state of the art are given alongside each treated aspect.

2 | Proposed method

A new approach, called Optimal Feature Rescaling (OFR), was proposed during this thesis in order to improve a Feed Forward Neural Network (FFNN) prediction performances in a regression problem. This method consists in globally optimize the training phase of a neural network model in order to obtain a set of optimal parameters, which are subsequently used to rescale the regression problem features. Since the FFNN training is typically non-convex, the performances of the network are expected to increase, demonstrating the effectiveness of this approach. Furthermore, OFR was tested in order to set the weights of the first layer of the considered network, allowing it to apply rescaling internally.

In this Chapter a brief overview of the main techniques that were applied throughout the thesis is presented:

- the regression problems are defined, describing their main elements and goal. Moreover, the cross-validation method is presented to test the performances of the considered model (Section 2.1);
- the FFNNs are then introduced, listing the characteristics of the model that was tested throughout the work. Several pre-processing techniques suggested in literature are presented as the state of the art. Furthermore, Early Stopping (ES) technique is proposed as a way to counteract the overfitting problem (Section 2.2);
- OFR method is described, highlighting the differences between the pre-processing techniques commonly used in the machine learning field (Section 2.3);
- the global optimization problems are then defined together with the used loss function and the Genetic Algorithm (GA) implemented to minimize it (Section 2.4);
- The procedures followed in Chapter 3 are presented (Section 2.5).

2.1. Regression problems

Regression problems are the most widespread machine learning problems. Given a data set composed by N input-output pairs $(\boldsymbol{x}_i, y_i)$ where $i = 1, ..., N$, the goal of a regression problem is to find a good approximation of the unknown function (a model) which maps inputs into continuous outputs and generalizes well on test data, e.g. predicting the house price based on the size of the house, a mobile robot energy consumption based on the followed path, etc. [2]:

$$\hat{y}_i = f(x_i)$$

Given a data set $\mathbf{X} \in \mathbb{R}^{N \times M}$ composed by N sample with M features (or variables) and representable by a data matrix with N rows and M columns, where each row is a sample $\boldsymbol{x}_n = (x_{n1}, ..., x_{nM}),\ n = 1, ..., N$:

$$\mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NM} \end{bmatrix}$$

and a target variable $\mathbf{y} = (y_1, ..., y_N)^T$, the goal of a regression problem is to make predictions $\hat{y}$ of the value of the target variable y for a new value of the input variable $\boldsymbol{x}$.

The observed data are affected by noise so, for a given $\boldsymbol{x}$, there is an uncertainty concerning the appropriate value for y. In order to take into account the mismatch between the real function and the chosen model a loss measure L has to be defined. Some common choices are Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, Coefficient of determination, etc.:

- Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.1}$$

  where $\hat{y}_i$ is the predicted value of y_i and N is the number of samples;

- Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{N}} \tag{2.2}$$

- Coefficient of determination (R^2):

$$R^2 = 1 - \frac{\text{SSE}}{\text{TSS}} \tag{2.3}$$

where

$$\text{SSE} = \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{2.4}$$

$$\text{TSS} = \sum_{i=1}^{N} (y_i - \bar{y}_i)^2 \tag{2.5}$$

are respectively the Sum of Squared Error (SSE) and the Total Sum of Squared Error (TSS), and $\bar{y}$ is the mean of the target variables. The coefficient of determination $R^2 \in (-\infty, 1]$ describes how well the model approximates the real function:

  - if $R^2 < 0$ it means that the model predicts the output worse than the mean value;
  - if $0 < R^2 < 1$ it means that the model predicts the output partially, but more accurately than the average value;
  - if $R^2 = 1$ it means that the model perfectly predicts the output.

Once a loss measure has been defined, an hypothesis space H has to be chosen; this latter is the set of models among which the appropriate one for the problem (h) has to be found. At the end of the process, an optimization problem has to be solved in order to find the best model h, which is the one minimizing the error function L.

2.1.1. 10-fold cross-validation

In regression problems the goal is to find a hypothesis h (or model) that generalizes well on unseen data. In order to measure the performances of the chosen model several techniques can be applied; with *k-fold cross-validation* being one of the most reliable. This technique consists in partitioning the dataset in $k = 10$ separate folds; then, $k - 1$ of the groups are used to train the model, which is subsequently evaluated on the remaining group [2]. This procedure is repeated for all the k possible choices for the held-out groups. In the

end, in order to obtain the final evaluation the k performances are averaged (see Figure 2.1 for a visual representation of the process).

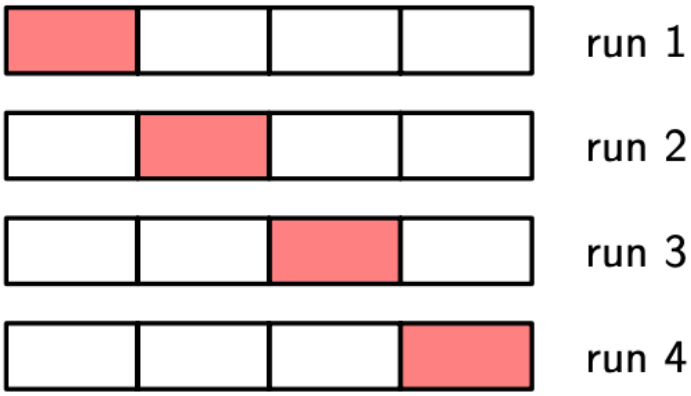


Figure 2.1: Cross-validation process representation with $k = 4$ (image taken from [2]).

During the thesis this technique was applied to the FFNN because neural networks require a large number of datapoints for the training phase. Exploiting the majority of them for the training was the preferred choice. The performances on the k folds were calculated by using as a metric the *Coefficient of determination* R^2 (2.1).

2.2. Feed Forward Neural Networks

Neural networks are one of the most powerful techniques used nowadays in machine learning. Most of the times, they show a multilayer structure composed by several neurons connected together by synapses in a way that simulates the human brain. An artificial neural network computes a function of the inputs by propagating the computed values from the input neurons to the output neuron(s) and using the weights as intermediate parameters. Learning occurs by changing the weights connecting the neurons.

The weights between neurons in a neural network are adjusted in response to prediction errors. In this way, the computed function is modified in order to make the predictions more correct in future iterations. By successively adjusting the weights between neurons over many input-output pairs, the function computed by the neural network is refined over time.

Multilayer neural networks contain multiple computational layers; between the input and output ones there are a few additional intermediate layers that are referred to as *hidden layers* because the computations performed there are not visible to the user. The specific architecture of multilayer neural networks is referred to as Feed Forward Neural Network

(FFNN), because successive layers feed into one another in the forward direction from input to output [1].

An example of a neural network used to solve a classification problem with 3 classes ("blue", "green" and "red") is shown in Figure 2.2. This network is characterized by 5 input nodes, 2 hidden layers, composed by 3 neurons each, and 3 output nodes.

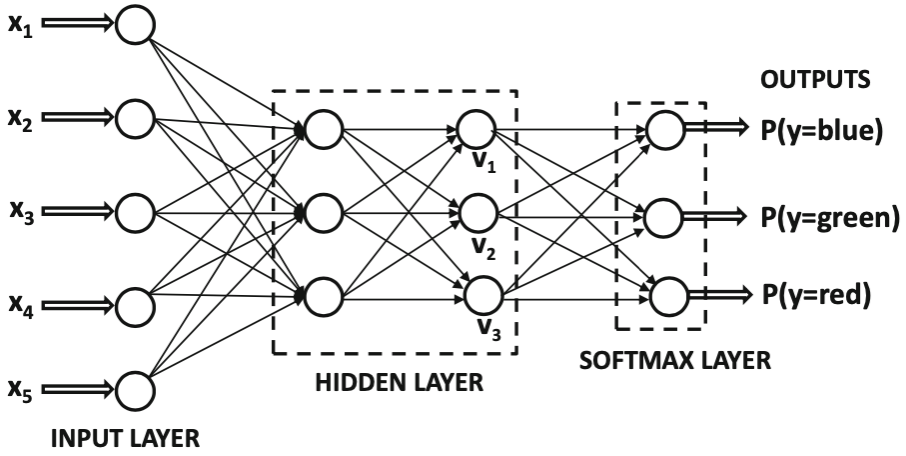


Figure 2.2: Example of neural network (image taken from [1]).

In order to improve its efficiency in the training phase, usually a pre-processing phase is carried out. The network might be able to automatically adapt to heterogeneous data, but it would make learning more difficult. A widespread best practice to deal with such data is to do feature-wise standardization [3]. During this thesis the OFR approach was tested both on raw data, i.e. no rescaled ones, and on standardized data.

2.2.1. Neural network pre-processing in literature

In literature, three are the main pre-processing operations which are usually implemented to improve the efficiency of the neural network training phase [1]:

1. *mean centering*: mean-centering the data can be useful in order to remove certain types of bias effects; in these cases, to each variable its mean is subtracted.
2. *feature normalization*: each feature is divided by its standard deviation. By combining this technique with mean-centering, the data is said to have been standardized;
3. *whitening*: the axes system is rotated in order to create a new set of de-correlated features, each scaled to unit variance. One way to achieve this result is by using

Principal Component Analysis (PCA).

When it is preferable to have all features with non-negative values, further types of pre-processing are used. In these cases, the absolute value of the most negative entry of a variable is added to the corresponding feature value of each data point. This technique is combined with min-max normalization which can be useful when the data needs to be scaled in range $(0, 1)$. In min-max normalization, each sample x_{ij} (where j is the feature and i is the number of the entry) is scaled as follows:

$$x_{ij} \Leftarrow \frac{x_{ij} - min_j}{max_j - min_j}$$

Where min_j and max_j are respectively the minimum and maximum values of the j-th attribute.

The goal of this thesis is to understand if OFR might lead to achieve better performances in comparison to the ones obtained without its use.

2.2.2. The overfitting problem in regression

When the FFNN performs well on the training data but has low cross-validated performances, it means that it is overfitting, i.e. the neural network is too complex for the considered problem. There are several techniques which can be used in order to counteract the overfitting problem, e.g. increasing the number of datapoints, regularizing the model, reducing the complexity of the hypothesis space (of the model), etc.

During this thesis, two approaches were followed:

- in the first instance, a technique called *Early Stopping* was used in order to stop the FFNN training when the training and validation errors begin to diverge. Early Stopping is described in Section 2.2.3;
- subsequently, also a *simplified neural network* with Early Stopping was also tested.

2.2.3. Early Stopping

When the FFNN is affected by the overfitting problem, as explained in the previous section, a technique called *Early Stopping* (ES) can be used in order to contrast the problem [1].

This technique consists in stopping the training of the network after only a few iterations. A portion of the training data is held out as a validation set. The FFNN is then trained

on the portion of the training data that does not include the validation set, while at the same time, the error of the model on that specific set is continuously monitored. After a certain number of iterations, a specific point can be identified where, while the error on the training data keeps decreasing, it begins to rise on validation data. This point is therefore chosen for termination, as further training causes overfitting. This technique allows to control overfitting even though the considered neural network is more complex than necessary.

It is important to not perform ES after tiny increases in the out-of-sample error, because it may be due to noisy variations. Thus, it is important to keep track of the best solution achieved so far on the validation data and to continue the training in order to check if the error keeps rising. If the increment persists for a certain number of epochs called *patience*, the training is stopped.

2.3. Optimal Feature Rescaling (OFR)

Feature rescaling is one of the most critical parts of the pre-processing phase in machine learning. In fact, it can make the difference between weak and strong machine learning models. In feature rescaling the problem variables are differently transformed depending on the method applied. Classical feature scaling techniques are *Normalization* and *Standardization*. The first one is used when a variable has to be bound between two extremes, e.g. $[0, 1]$. The second one, instead, is used to transform the data so that their mean is equal to zero and they have unitary variance. Usually, these techniques are used to:

- make the features comparable;
- make a few algorithms converge faster, e.g. training processes of neural networks (see Section 2.2.1).

In this work, feature rescaling was proposed in order to improve FFNN performances.

The idea behind this thesis is to find a set of rescaling parameters $\mathbf{w} = \{w_1, ..., w_M\}$ that can be used to rescale the M input variables of a regression problem. Given a dataset $\mathbf{X} \in \mathbb{R}^{N \times M}$ and a vector of rescaling parameters $\mathbf{w} \in \mathbb{R}^M$, the rescaled dataset $\tilde{\mathbf{X}} \in \mathbb{R}^{N \times M}$ is obtained as:

$$\tilde{\mathbf{X}} = \mathbf{w}\mathbf{X} = \left[w_1 \cdot \begin{pmatrix} x_{11} \\ \vdots \\ x_{N1} \end{pmatrix}, \quad \cdots, \quad w_M \cdot \begin{pmatrix} x_{N1} \\ \vdots \\ x_{NM} \end{pmatrix} \right] \tag{2.6}$$

and $\tilde{\boldsymbol{x}}_i = (w_i x_{1i}, ..., w_i x_{Ni})^T,\ i = 1, ..., M$ is the *i-th* rescaled feature of the problem.

In order to improve the prediction capabilities of the model (in this study, a FFNN), a vector of optimal parameters $\mathbf{w}^*$ has to be chosen, hence the name *Optimal Feature Rescaling*. Therefore, an optimization problem was defined. Since the analytical form of the cost function is not known in advance a *global optimization* approach was followed.

2.3.1. Fixing the FFNN input layer weights

An interesting aspect of the FFNN is that, in the first layer, it implicitly performs an input rescaling, multiplying the dataset features by the input layer weights [1]. If a_j^l is the j-th input to the layer l, and β_{jk}^l is the weight from the k-th neuron in the $(l-1)$-th layer to the j-th neuron in the l-th layer, such that:

$$a_j^l = f(\sum_k \beta_{jk}^l a_j^{l-1} + b_j^l) \tag{2.7}$$

where b_j^l is the j-th bias to the layer l and $f(\cdot)$ is the activation function of the considered layer.

Extending this concept by exploiting the OFR technique, it can be possible to set the first layer weights in order to optimally rescale the neural network inputs. If w_k is the k-th rescaling parameter, it can be incorporated into the k-th weight of the first layer in order to enable the network to perform input OFR without any data preprocessing. In this case the weight from the k-th neuron in the first layer to the j-th neuron in the second layer, becomes:

$$\tilde{\beta}_{jk}^1 := \beta_{jk}^1 w_k$$

while the Equation 2.7 changes into:

$$a_j^l = f(\sum_k \tilde{\beta}_{jk}^1 a_j^0 + b_j^1) = f(\sum_k \beta_{jk}^1 w_k a_j^0 + b_j^1) \tag{2.8}$$

This particular application of the OFR technique was tested during the thesis, and the obtained results were described and discussed in Section 3.6.

2.4. Global optimization

The main goal of global optimization is to find the global optimal point $\boldsymbol{x}^*$ (minimum or maximum) of the given possibly non-linear and *non-convex* continuous function $f(\boldsymbol{x})$

(where $\boldsymbol{x}$ are the *decision variables* and f is the *objective function*) from a solution space Ω (called *feasible region*) $f : \Omega \to T$, where T is any ordered set (usually $\in \mathbb{R}$) and Ω is usually a subset of $\mathbb{R}^n$ [9][14]. The solution space Ω can be subjected to equality and inequality constraints, but during this thesis an unconstrained global optimization approach was followed, so no further details are required.

Then, a loss function for the parameters optimization problem has then to be defined. Since the target model for this thesis is a FFNN, the considered objective function takes as inputs the vector of parameters (the optimization problem decision variables) and the training/validation sets, performs the feature rescaling using the input parameter vector, then trains a FFNN using the training set and returns the RMSE (2.1) calculated on the validation data (see Appendix A for the code):

$$\text{RMSE}_{validation} = f(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train}, \mathbf{X}_{val}, \mathbf{y}_{val}) \tag{2.9}$$

In this scenario, the analytical form of the objective function is not known beforehand. But, since the considered cost function, i.e. the training of the neural network, turns out to be non-convex, it is possible to find a better minimum point and therefore obtain different performances following different scalings. If the cost function was convex, then its convexity would remain unchanged even through a rescaling, and it would not be possible to reach a better minimum point:

- *affine input transformation*: if $f : \Omega \to \mathbb{R}$ is convex (where $\Omega \subseteq \mathbb{R}^n$), then also $\tilde{f}(\boldsymbol{x}) = f(A\boldsymbol{x} + \mathbf{b})$ is convex on the domain $\tilde{\Omega} = \{\boldsymbol{x} \in \mathbb{R} : A\boldsymbol{x} + b \in \Omega\}$, with $A \in \mathbb{R}^{n \times m}, \mathbf{b} \in \mathbb{R}^m$ [5].

Once the optimization problem is set, a solver should be used in order to get to a solution. Thus, during the thesis a *Genetic Algorithm* was implemented.

2.4.1. Genetic Algorithms

Genetic Algorithms (GA) are metaheuristics that belong to the family of evolutionary algorithms; they are inspired by the natural evolutionary theory and try to emulate the process of natural selection where the next generation of offspring is produced by a group of individuals selected for reproduction [10].

A GA follows 5 steps in order to provide a solution:

1. *initialization*: a population of candidate solutions is generated (the so-called *individuals*);

2. *evaluation*: the fitness function (the objective function 2.9) is evaluated for every individual;
3. *selection*: among the individuals the best candidate solutions are chosen;
4. *variation*: variations are carried out on the candidate solutions chosen in the previous step in order to produce new offspring;
5. *replacement*: the old solutions are replaced with the new offspring in the candidate population and the previous 4 steps are repeated until one or more convergence criteria are matched.

Throughout the thesis, the algorithm (see Appendix A for the code) was initialized with 20 individuals sampled randomly from a uniform distribution (in log-scale) and it was executed for 100 iterations, i.e. 2020 function evaluations.

The GA individuals, i.e. the parameter vectors that compose the initial population, were sampled from a uniform distribution between $[-3, 3]$ in logarithmic scale. They were then converted in decimal scale, i.e. $w \in [-10^3, 10^3]$, before being applied to the dataset features.

2.5. Proposed procedure

The procedure proposed in this work and tested on the case study described in Chapter 3 is reported below.

General procedure (Optimal Feature Rescaling):

1. the regression problem is defined by identifying its input and target variables;
2. a holdout procedure is performed to define the training, validation and test sets;
3. the structure of the neural network that has to be trained is chosen, i.e. the hypothesis space H;
4. the loss function that has to be minimized through the global optimization problem is defined:
    - it takes as input the decision variables of the problem, i.e. the weights to be optimized, and the training and validation sets;
    - it transforms the input parameters from logarithmic scale to decimal scale, performs the rescaling of the input variables of the training set, trains the neural network chosen at point 3 and tests it on the validation set;

- it returns the RMSE (2.1) calculated on the validation data;

5. a solver is chosen to minimize the loss function defined in step 4. During the thesis a Genetic Algorithm has been used;
6. the Genetic Algorithm is initialized by sampling from a uniform distribution 20 sets of parameters;
7. the global optimization problem is solved in order to obtain the optimal parameters set;
8. the optimal parameters calculated in step 7 are used to rescale the variables of the training, validation and test set;
9. the neural network chosen in step 3 is then trained on the rescaled training set and tested by using the cross-validation technique;
10. in order for the neural network to be able to carry out the OFR internally, the optimal parameters calculated in point 7 must be integrated within the weights of the first level of the network.

3 | Case study

All the tools introduced in the previous chapter were applied in order to test OFR in a real scenario: roundness prediction for a centerless grinding machining process. Centerless grinding is a machining operation where material is removed from the workpiece using an abrasive wheel. The work part is not clamped during the machining process, making it prone to inherent instability, and therefore the overall quality of the final piece is difficult to predict on the basis of the chosen process parameters.

A regression problem was defined to predict the roundness of the worked part and three tests were carried out in order to understand if OFR can effectively improve the performances of a neural network. In this Chapter:

- the dataset was prepared by defining the input and the target variables of the problem. The considered features were the centerless grinding process parameters, while the target variable was the roundness of the worked part (Section 3.1);
- A first FFNN was trained both on the raw data, i.e. the non-rescaled data, and on the standardized ones in order to define two baselines. Afterwards, both of the datasets were optimally rescaled by applying the OFR procedure and the FFNN was trained on them. The obtained results were then compared with the baselines (Section 3.2);
- In order to counteract the overfitting problem, the ES technique was applied and a second test was carried out (Section 3.3);
- the second test demonstrated that the overfitting problem could not be solved only with ES, so a simplified neural network was implemented and tested as done before (Section 3.4);
- the FFNN first layer weights were fixed by using the optimal parameters in order to apply OFR internally. The obtained performances were then compared to the ones achieved by applying the optimal rescaling outside the network (Section 3.6).

3.1. Dataset definition

In centerless grinding the workpiece is supported on three points by two wheels, the so-called *grinding* and *control wheels*, and a *work rest*, also known as the supporting blade. The worked part and its supports are represented in Figure 3.1:

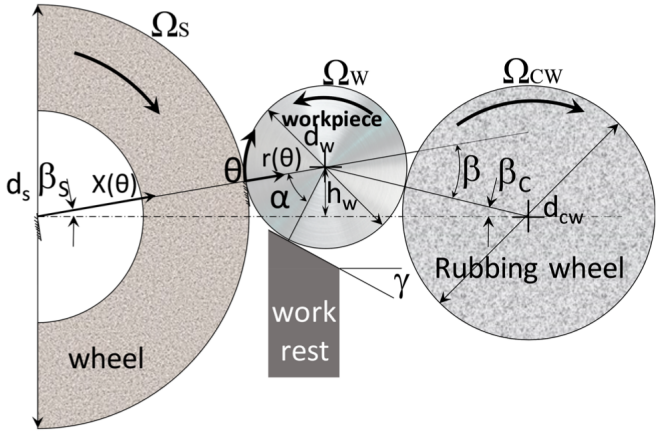


Figure 3.1: Centerless grinding setup (image taken from [8]).

The workpiece is pushed against the grinding wheel according to a feed movement, and the material is removed due to the contact between them. Each grinding operation can be defined by 12 parameters which are listed in Table 3.1.

x_0: workpiece height (h_w)	x_6: grinding wheel diameter
x_1: blade angle (γ)	x_7: control wheel diameter
x_2: feed velocity	x_8: control wheel velocity
x_3: total diametric removal	x_9: grinding specific energy
x_4: workpiece length	x_{10}: edge force component
x_5: workpiece diameter	x_{11}: grit stiffness

Table 3.1: Centerless grinding process parameters [8].

There are 9 other parameters that are characteristic of the grinding machine and not of the process. For this reason, they will be considered as fixed for this study. These parameters are listed in Table 3.2.

α_0: grinding wheel velocity	α_5: mass matrix (dynamics)
α_1: Number of workpiece elements	α_6: stiffness matrix(dynamics)
α_2: initial workpiece profile distribution	α_7: damp.matrix (dynamics)
α_3: grit diameter	α_8: mode shape vector (dynamics)
α_4: abrasive/bond ratio	

Table 3.2: Centerless grinding process fixed parameters [8].

The model of the centerless grinding process (Figure 3.2), the so-called *high fidelity model* (details and equations can be found, for instance, in [4]), was then used in order to obtain a dataset for the regression problem:

$$\mathcal{D} := \{y_i, \mathbf{x}_i\}, i = 1, 2, ..., N_s$$

where $y_i \in \mathbb{R}^+$ is the continuous value of the roundness of the final piece that measures the process performances of the i-th sampled grinding operation (i.e. the target variable), $\mathbf{x}_i \in \mathbb{R}^{12}$ is the corresponding vector of process parameters (i.e. the input variables), and $N_s = 4069$ is the number of samples.

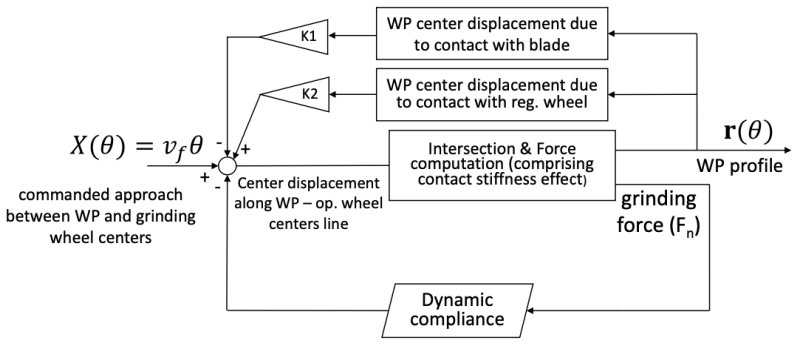


Figure 3.2: High fidelity model (image taken from [8]).

The feature set (that consisted of the 12 aforementioned parameters) was further expanded by using the output of the so-called *low fidelity model*, which is a simplified version of the (more complicated) high fidelity one (all the assumptions made for the low fidelity model are reported in [8]).

The dataset thus obtained was divided in 3 subsets by applying a *hold-out procedure*:

- 70% of the data was reserved for training;

- the remaining 30% was further divided in two equal sets:
  - 50% for validation;
  - 50% for test.

In order to reserve the majority of the data for the FFNN training, only a small portion of them was held out for the test phase. For this reason, as described in Section 2.1.1, during the thesis a cross-validation procedure was used in order to test the performances of the FFNN. Because of the dimensions of the test set, it was only used in cases where the cross-validation procedure was not a viable option.

3.2. Optimal Feature Rescaling (OFR) application

Once the dataset was defined, the OFR approach was tested by training a FFNN on an optimally rescaled set of data. The tested FFNN, i.e. the considered hypothesis space H, was composed by:

- an input layer with 128 neurons and *relu* activation function;
- 3 hidden layers with 256 neurons each and *relu* activation functions;
- an output layer with a single neuron with *linear* activation function.

MAE (2.1) was used as loss metric for the training and validation phases, while an optimizer implementing the *Adam* algorithm was used for the optimization phase. In the end, 500 *epochs* were considered during the training phase and the neural network was trained with all the training data in each one of them. As a result, the considered network was trained 500 times in order to define the final set of parameters for each layer.

The FFNN described above was implemented in Python making use of Keras. Keras is a deep learning framework for Python that provides a practical way to define and train many different deep learning models [3].

The FFNN in exam was trained on two different datasets and the performances thus obtained were used in order to define the baselines for the test. The neural network was tested on:

1. the raw data, i.e. data without feature rescaling;
2. the standardized data.

The two baselines were then compared to the performances obtained training the FFNN on:

1. the data rescaled by applying OFR on the raw ones;
2. the data rescaled by applying OFR on the standardized ones.

3.2.1. Baseline 1: performances on raw data

The first baseline was obtained by training and cross-validating 100 FFNNs on the raw data. The performances were then averaged and reported in Table 3.3.

Test	R^2 (Train)	CV Mean	CV Std
No rescaling	0.515098	0.285996	0.168344

Table 3.3: Baseline performances using raw data.

As it can be seen from Table 3.3, the network performances are rather poor. Because the FFNN training phase is very sensitive to data with different scales, the obtained performances could depend from the absence of a preliminary standardization phase (see Section 2.2).

In addition, by observing the difference between the performances obtained on the train set and the ones obtained by applying the cross-validation, an overfitting phenomenon can be seen (Section 2.2.2).

3.2.2. Baseline 2: performances with standardization

The second baseline was obtained by applying a standardization procedure. The data were rescaled by subtracting the mean and dividing by the standard deviation in order to have mean equal to zero and unitary variance, as described in Section 2.2.1. Considering a sample $\boldsymbol{x}_i$, with $i = 1, ..., M$, where M is the number of features, its standardized version is:

$$\tilde{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_i}{\sigma_i}$$

where μ_i is the mean and σ_i is the standard deviation of the i-th feature.

As in the previous case, 100 FFNNs were trained and cross-validated, and their performances were averaged and reported in Table 3.4.

Test	R^2 (Train)	CV Mean	CV Std
Standardized	0.990043	0.369556	0.337064

Table 3.4: Baseline performances using standardized data.

It can be seen that the network is completely overfitted; the cross-validated performances improved slightly but the coefficient of determination on the training data increased substantially, thus dramatically increasing the gap between the training and the cross-validated performances, i.e. the model generalizes badly on test data.

3.2.3. OFR applied to raw data

The OFR technique was applied to the raw data in order to understand if the absence of a common standardization procedure could be compensated. In this phase, an unconstrained global optimization program (Section 2.4) was defined in order to determine the optimal parameter vector $\mathbf{w}^*$ for feature rescaling. The GA described in Section 2.4.1 was used to minimize a loss function which performs feature rescaling, trains the FFNN described above and computes the RMSE (2.1) on validation data (2.9).

The optimal parameter vector $\mathbf{w}^* = (w_0, ..., w_{12})$ is reported in decimal scale in Table 3.5.

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
19.553	18.242	0.303	65.383	37.113	0.005	0.279
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
0.017	4.958	1.303	0.019	1.662	30.657	

Table 3.5: Optimal parameters for raw data in dec-scale.

Just like the two baselines, 100 FFNNs were trained and cross-validated and their performances were averaged and reported in Table 3.6.

Test	R^2 (Train)	CV Mean	CV Std
OFR on raw data	0.822084	0.475483	0.092552

Table 3.6: Performances obtained applying OFR to raw data.

In the end, the execution time of the GA was reported in Table 3.7.

Test	Function calls	$[s]$	$[h]$
GA with raw data	2020	119369.0	33.16

Table 3.7: GA execution time working on raw data.

3.2.4. OFR applied to standardized data

Subsequently, the OFR method was applied to the standardized data in order to understand if it was able to further improve the performances. Similar to what was done in the previous section, a global optimization program was solved (Section 2.4) and the parameters obtained as output were reported in decimal scale in Table 3.8.

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
1.919	0.032	0.374	0.495	0.184	0.185	0.128
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
0.022	0.239	0.517	0.937	0.030	29.463	

Table 3.8: Optimal parameters for standardized data in dec-scale.

The performances obtained training 100 FFNNs on the standardized and optimally rescaled data were averaged and reported in Table 3.9.

Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on std data	0.844512	0.561060	0.117817

Table 3.9: Performances obtained by applying OFR to standardized data.

The execution time is similar to the one obtained working with raw data, and it is reported in Table 3.10

Test	Function calls	$[s]$	$[h]$
GA with std data	2020	119886.0	33.30

Table 3.10: GA execution time working on standardized data.

3.2.5. Performances comparison

All the results obtained in Sections 3.2.1 to 3.2.4 were collected in Table 3.11 in order to make some considerations.

Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on std data	0.844512	0.561060	0.117817
OFR on raw data	0.822084	0.475483	0.092552
Standardized	0.990043	0.369556	0.337064
No rescaling	0.515098	0.285996	0.168344

Table 3.11: Performances comparison ordered with respect to "CV Mean".

The first thing to take into account is the difference between the performances obtained using the raw data and the standardized ones. If the standardization is not applied, the performances tend to be worse. Nevertheless, the performances increase on the training data makes the overfitting problem more incisive.

The second important aspect to pay attention to is the clear improvement guaranteed by the application of the OFR technique to the non-rescaled dataset. It improves not only the cross-validated performances obtained on the raw data, but also the ones obtained on the standardized ones. In fact, these performances improved by 66.3% compared to the raw data:

$$0.285996 \rightarrow 0.475483$$

and by 28.66% compared to the standardized data:

$$0.369556 \rightarrow 0.475483$$

Although the neural network in exam does not have good prediction performances, an improvement can be seen, underlining the fact that, with a better network, the OFR procedure could guarantee good results. Furthermore, it can be noticed that the OFR

mitigates the overfitting problem, reducing the gap between train error and the cross-validation output.

Last but not least, the performance improvement obtained by applying OFR to a standardized dataset is far greater than all of the previous cases, guaranteeing an increment of 95.69% compared to using of raw data:

$$0.285996 \rightarrow 0.559659$$

and of 51.44% compared to using the standardized data:

$$0.369556 \rightarrow 0.559659$$

Furthermore, the performances obtained by applying OFR on the raw data were overcome by 17.70% as well:

$$0.475483 \rightarrow 0.559659$$

demonstrating the fact that a previous standardization can be useful even when an OFR procedure is carried out. Although these results are promising, the test makes it clear that OFR cannot compensate the effect of a simple standardization, making the latter a desired step. Probably, the reason lies in the fact that removing the mean helps to improve the quality of the model, i.e. FFNN, while OFR is unable to do so as it acts only on the scale factors.

Nevertheless, analyzing the discrepancy between the performances on the train set and the ones obtained with cross-validation, it can be seen that, although the OFR mitigates the overfitting problem and reduces the gap, the ES technique described in Section 2.2.3 has to be applied in order to further reduce it.

For completeness, the GA execution times are reported in Table 3.12.

Test	Function calls	$[s]$	$[h]$
GA with std data	2020	119886.0	33.30
GA with raw data	2020	119369.0	33.16

Table 3.12: Comparison between GA execution times working on raw and standardized data.

In conclusion, in order to gain a better understanding of the cross-validated performances

and their sort, they are reported in Figure 3.3.

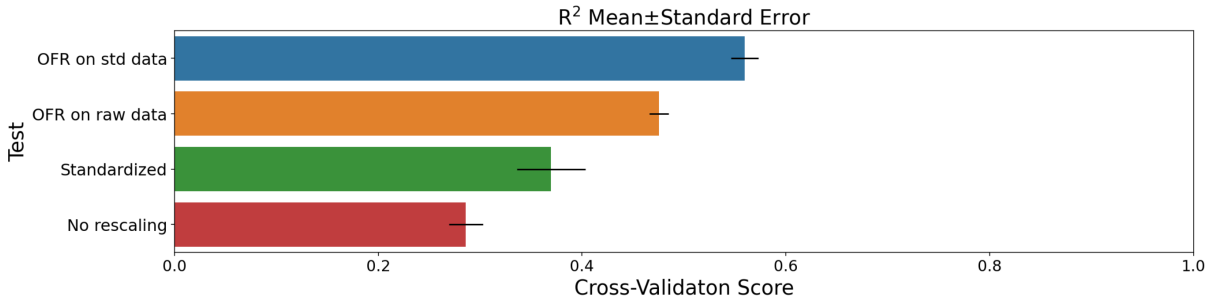


Figure 3.3: Comparison between FFNN cross-validated performances.

From Figure 3.3 the improvement brought by the application of the OFR procedure on both performances and the reduction of the overfitting problem of the used models (as it can be deduced from the previous considerations) is evident. In the next Section, tests were taken modifying the neural network by implementing the ES technique described in Section 2.2.3 in order to reduce the overfitting problem.

3.3. OFR combined with Early Stopping technique

In order to counteract the overfitting problem, an ES approach was followed as described in Section 2.2.3. The number of epochs for the FFNN training was increased to 10000, while the ES patience (see Section 2.2.3) was set to 100. The goal of this choice is to make sure that the ending of the training phase is reached before the end of the epochs. In this way, it is guaranteed that the FFNN training stops when the validation error begins to rise.

As it was done in the previous Sections, two baselines were defined in order to have a ground truth with which to compare the performances obtained by applying the OFR technique. The FFNN implementing the ES was trained both on:

1. raw data, i.e. the non rescaled data;
2. the standardized data, as described in Section 3.2.1.

Afterwards, the OFR was applied to both the dataset and the same neural network was trained and tested. All the performances were then compared and discussed.

3.3.1. Baselines with Early Stopping

The first baseline was set by training the FFNN on raw data and the obtained performances are reported in Table 3.13. The second one was defined by training the FFNN on the standardized data, as explained in Section 3.2.2 and the obtained results are reported in Table 3.14.

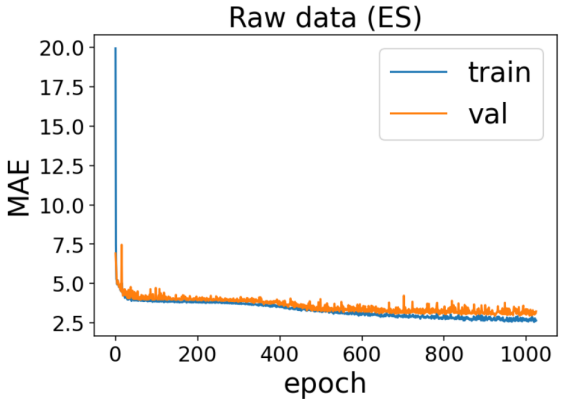
Test	R^2 (Train)	CV Mean	CV Std
No rescaling (ES)	0.655322	0.357649	0.080880

Table 3.13: Performances obtained by training FFNN with ES on raw data.

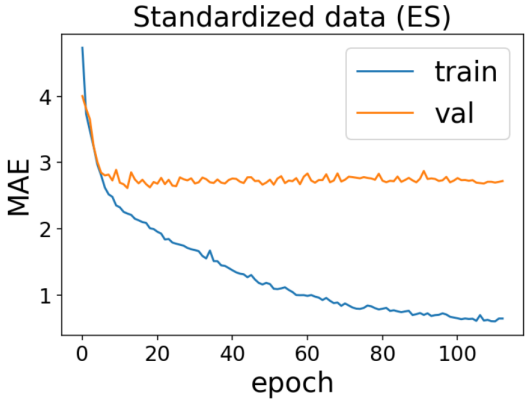
Test	R^2 (Train)	CV Mean	CV Std
Standardized (ES)	0.808376	0.456640	0.116470

Table 3.14: Performances obtained by training FFNN with ES on standardized data.

Because the ES technique was implemented, it makes sense to visualize the evolution of the loss metric, i.e. the MAE (2.1), on the training (*loss*) and validation data (*val loss*). The two plots, one for each test, are respectively reported in Figure 3.4a and 3.4b.



(a) Loss and val loss calculated on the raw data (ES).



(b) Loss and val loss calculated on the standardized data (ES).

Figure 3.4: Loss and val loss evolution on the baselines defined for the FFNN with ES.

As it can be seen from Figures 3.4a - 3.4b, the FFNN stops the training before the end of the set epochs. Furthermore, by standardizing the data, the training error was reduced

more rapidly compared to the case with raw data. Another consideration that can be done by observing the images is that in the second baseline, i.e. the one obtained using the standardized data, the training phase lasts much less compared to the first baseline thanks to the standardization, which improves the efficiency of the training phase. In order to further minimize the overfitting effect one possibility could be to stop the training around the 15th epoch, i.e. the epoch where the validation loss reaches the minimum value.

3.3.2. OFR application with Early Stopping

Once the baselines were defined the OFR technique was tested in order to evaluate its performances. Differently from the previous cases, the loss function for the optimization problem was modified in order to take into account the ES approach.

The optimal parameters (converted in dec-scale) obtained by applying the GA with the modified cost function to the two sets of data (see Section 2.4.1) are reported in Table 3.15 and 3.16.

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
62.424	1.010	2.521	20.619	19.503	0.079	0.167
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
0.747	3.112	0.079	0.799	0.765	9.226	

Table 3.15: Optimal parameters for raw data in dec-scale (ES).

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
11.671	0.213	1.803	1.623	0.302	0.121	0.039
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
0.005	0.608	1.544	0.112	0.248	150.160	

Table 3.16: Optimal parameters for standardized data in dec-scale (ES).

While the times required by the GA to provide the optimal parameters are reported in hours and seconds respectively in Table 3.17.

Test	Function calls	$[s]$	$[h]$
OFR on raw data (ES)	2020	69992.0	19.44
OFR on std data (ES)	2020	61768.0	17.16

Table 3.17: GA execution times when ES is implemented.

As expected, the GA provides the optimal parameters in less time than the case without ES. This is due to the fact that the validation error starts to increase before the 500th epoch (planned for the previous cases), so the training phase is stopped earlier.

As for the case without ES, the optimal parameters in Table 3.15 and 3.16 were respectively applied to the raw and standardized data. The obtained performances are respectively reported in Table 3.18 and 3.19.

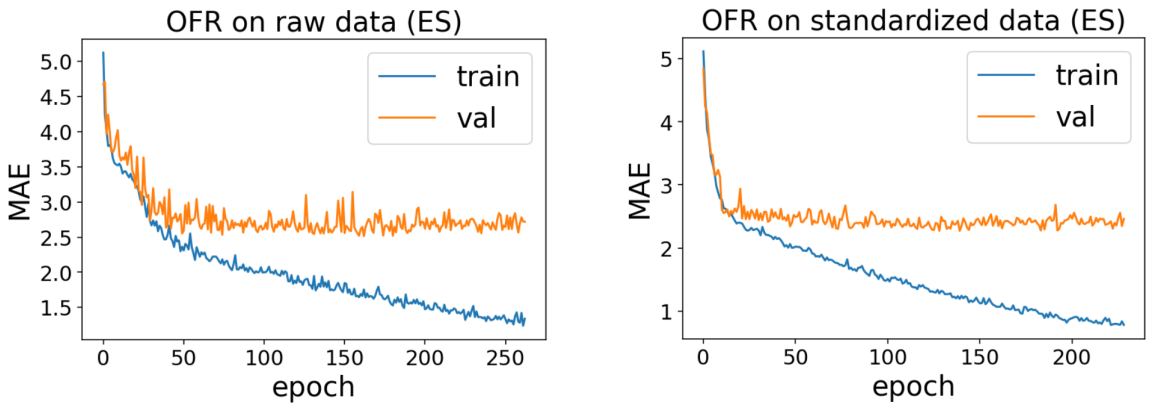
Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on raw data (ES)	0.888022	0.488795	0.111678

Table 3.18: Performances obtained by applying OFR to raw data (ES).

Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on std data (ES)	0.818443	0.527066	0.137574

Table 3.19: Performances obtained by applying OFR to standardized data (ES).

While the loss and val loss evolutions are represented in Figure 3.5a and 3.5b.



(a) Loss and val loss calculated on the raw data optimally rescaled (ES).

(b) Loss and val loss calculated on the standardized data optimally rescaled (ES).

Figure 3.5: Loss and val loss evolution obtained training the FFNN with ES on the optimally rescaled datasets.

From Figures 3.5a and 3.5b it can be noticed how the standardization procedure improves the efficiency, reduces the training time and allows a smaller training error to be achieved.

3.3.3. Performances comparison - the effects of ES

Once the baselines and the performances obtained by applying OFR were collected, it was possible to analyze and comment them. The performances are reported in Table 3.20.

Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on std data (ES)	0.818443	0.527066	0.137574
OFR on raw data (ES)	0.888022	0.488795	0.111678
Standardized (ES)	0.808376	0.456640	0.116470
No rescaling (ES)	0.655322	0.357649	0.080880

Table 3.20: OFR performances comparison ordered with respect to "CV Mean" (ES).

The improvement achieved by using the OFR technique can be seen from Table 3.20. However, the performances of the FFNN are still rather poor despite the implementation of ES. This probably depends on several factors:

1. the FFNN is too complex for the considered problem;
2. the parameters that were set for the ES were too permissive;

3. the numerosity of the data;
4. the noise on the data.

The prediction performances of the network would surely benefit from a *model selection* procedure, where several models are tested on a validation set in order to choose the best among them. However, since the prediction problem was not the topic of this thesis, this step was avoided.

As previously done, the cross-validated performances are resumed in Figure 3.6 in order to visualize them more clearly.

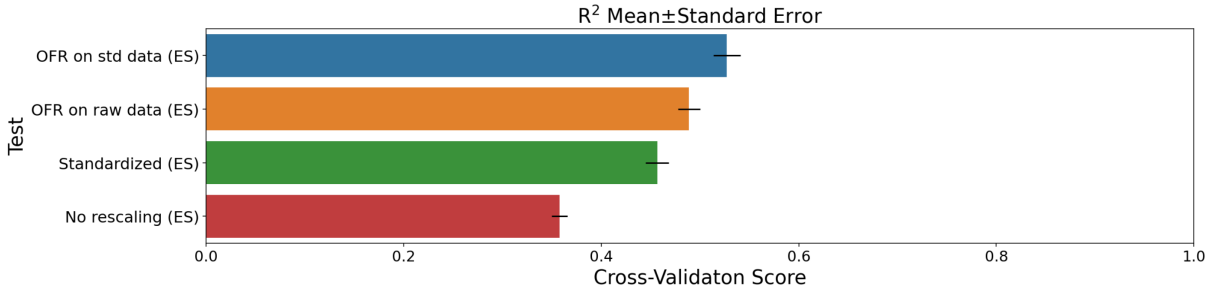


Figure 3.6: Comparison between all the FFNN (with ES) cross-validated performances.

Concerning the first of the possible causes of the persistence of the overfitting problem, a simpler net was tested in Section 3.4. In this way, it was possible to figure out what the effects of a model selection procedure would be.

3.4. OFR effects on a simplified FFNN with ES

In order to test the efficacy of the OFR method with a model not subjected to the overfitting problem a simpler neural network was tested. As described in Section 2.2.2, simplifying the structure of the FFNN is possible in order to reduce the overfitting effects.

For these reasons, the neural network described in Section 3.2 (i.e. the hypothesis space H) was simplified by drastically reducing the number of layers and neurons. The resulting model is composed by:

- an input layer with 13 neurons and *relu* activation function;
- 1 hidden layer with 100 neurons and *relu* activation function;
- an output layer with a single neuron with *linear* activation function.

Furthermore, the objective function for the optimization program was changed, reflecting the new considered model.

The ES patience was incremented to 200, in order to gain more training time for the FFNN.

As it was done in the previous paragraphs, the two baselines (one obtained by using the raw data and one by using the standardized ones) were defined in order to compare the performances obtained by applying the OFR technique. Furthermore, the evolution of *loss* and *val loss* can be visualized as before. The two plots are reported in Figure 3.7a and 3.7b.

The optimal parameters obtained by solving the global optimization problems are reported (in decimal scale) in Table 3.21 (considering the raw data) and 3.22 (considering the standardized data).

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
7.801	0.114	7.126	0.015	0.092	0.059	0.082
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
0.208	1.862	0.006	0.416	0.112	253.964	

Table 3.21: Optimal parameters (dec-scale) for raw data (simplified FFNN with ES).

w_0^*	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*
1.092	2.238	0.500	0.264	0.152	0.034	1.070
w_7^*	w_8^*	w_9^*	w_{10}^*	w_{11}^*	w_{12}^*	
1.069	0.547	0.500	1.362	0.674	13.034	

Table 3.22: Optimal parameters (dec-scale) for standardized data (simplified FFNN with ES).

While the GA execution times are reported in Table 3.23.

Test	Function calls	$[s]$	$[h]$
OFR on raw data (simplified NN)	2020	173807.0	48.28
OFR on std data (simplified NN)	2020	134213.0	37.28

Table 3.23: GA execution times for the simplified neural network (with ES).

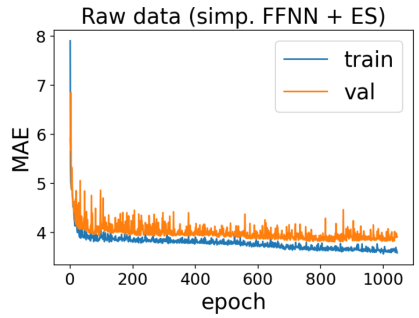
All the performances obtained by testing the simplified FFNN are reported and compared in Table 3.24, the cross-validation results can be visualized in Figure 3.8, and the loss and val loss evolutions are reported in Figure 3.7c and 3.7d.

As it can be seen from Table 3.24 the networks trained on the optimally rescaled datasets achieve better performances compared to the baselines. Furthermore, the overfitting problem was almost completely solved, significantly reducing the gap between the training error and the cross-validated performances.

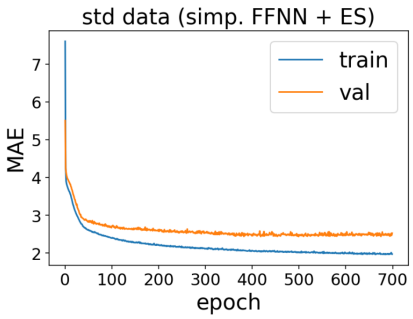
Test	R^2 **(Train)**	**CV Mean**	**CV Std**
OFR on std data (simplified NN)	0.615985	0.595238	0.098966
Standardized (simplified NN)	0.629364	0.540791	0.147672
OFR on raw data (simplified NN)	0.674978	0.533133	0.147843
No rescaling (simplified NN)	0.335624	0.237633	0.464777

Table 3.24: Performances comparison obtained by training a simplified FFNN (with ES), ordered with respect to "CV Mean".

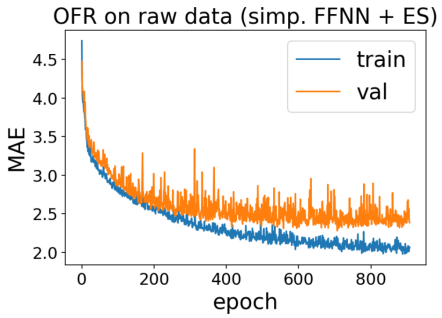
As expected, from Figure 3.7 it can be noticed that the training phase lasts much longer than the cases considered in the previous section. This is due to the fact that in this case the patience was set to 200. By observing Figure 3.7d (which corresponds to the best case scenario) it can be deduced that an optimal termination point for the training phase is reached around the 250th epoch. In fact, the validation loss after that point remains more or less constant, thus making the trade-off between training time and performance improvement no more profitable.



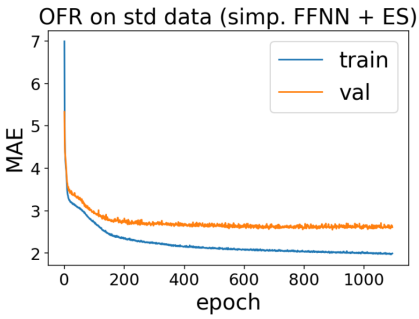
(a) Loss and val loss calculated on the raw data (simplified FFNN with ES).



(b) Loss and val loss calculated on the standardized data (simplified FFNN with ES).



(c) Loss and val loss calculated on the raw data with OFR (simplified FFNN with ES).



(d) Loss and val loss calculated on the standardized data with OFR (simplified FFNN with ES).

Figure 3.7: Loss and val loss evolution during the simplified FFNN training phase with ES.

Although the performances obtained by applying OFR to the standardized dataset were better compared to the baselines, from Figure 3.8 it is evident that even with the simplification of the neural network the performances achieved by applying a common standardization to the data are almost equal (greater in this case) to the ones obtained by optimally rescaling the raw data. Therefore, the usefulness of a standardization procedure prior to the application of the OFR technique has been confirmed in this case as well as it was already done in the Section 3.2.2.

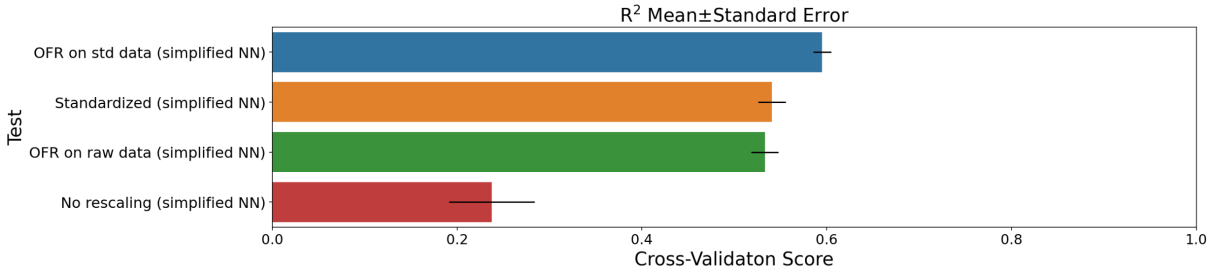


Figure 3.8: Comparison between all the simplified FFNN (with ES) cross-validated performances.

This test was carried out to demonstrate that the application of both the OFR and the model selection procedures can guarantee an interesting improvement in the prediction performances of a FFNN applied to a regression problem, confirming, in every case, the usefulness of the OFR technique in order to improve the performances of the network.

3.5. Final comparisons

In conclusion, the best performances obtained in the three tests are reported in Table 3.25 and in Figure 3.9 in order to take some final considerations.

Test	R^2 (Train)	CV Mean	CV Std
OFR on std data (simplified NN)	0.615985	0.595238	0.098966
OFR on std data	0.808739	0.559659	0.133694
OFR on std data (ES)	0.818443	0.527066	0.137574

Table 3.25: Comparison between the best performances obtained in the three tests.

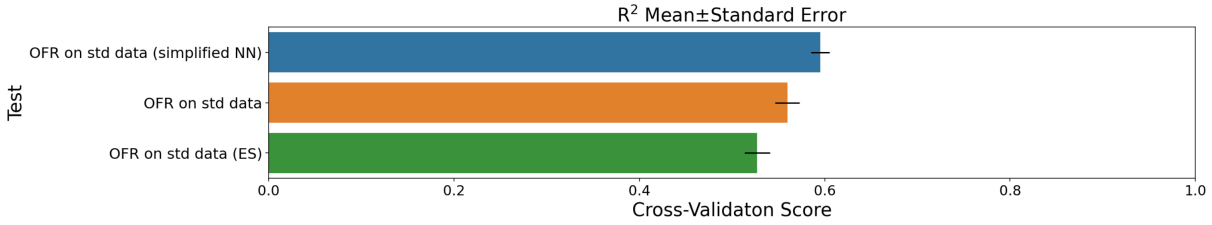


Figure 3.9: Comparison between the best cross-validated performances obtained in the three tests.

From Table 3.25 it can be seen that the simplified neural network described in Section 3.4

performs better than the more complex ones and, at the same time, it is not affected by the overfitting problem. Performing a model selection procedure before the application of the OFR technique, as demonstrated by the results obtained with the simplified network, would guarantee even better results.

In the next section a different application of the OFR technique was tested, highlighting how the combination of the method and the flexibility of neural networks can be useful in machine learning and deep learning applications.

3.6. Fixing the input layer weights of the FFNN

By exploiting the feature of neural networks to apply the rescaling of the variables within the input level, it was possible to test the particular application of the OFR technique described in Section 2.3.1.

Table 3.26 shows the performances obtained by FFNN (Section 3.2) by carrying out the optimal rescaling outside and inside the network, i.e. modifying the first level of the neural network.

Test	**RMSE (Test)**	R^2 **(Train)**	R^2 **(Test)**
Inside rescaling	5.092644	0.82803	0.247903
Outside rescaling	5.092644	0.82803	0.247903

Table 3.26: Performances obtained by fixing with OFR the FFNN input layer.

Observing the Table 3.26, it can be seen that the performances obtained by applying the optimal rescaling outside the neural network coincide with those obtained by using the unscaled data, but by modifying the weights of the first level of the network.

Thanks to the non-convexity of the training problem it is always possible to pass from the neural network original weights to the ones optimally selected. For this reason, the global optimization can be used to "globalize" the training phase of the FFNN and, in particular, the initial values of its first layer.

4 | Conclusions and future developments

4.1. Conclusions

In this thesis, a new method to improve the performances of a Feed Forward Neural Network in regression problems was defined and tested. This method was called Optimal Feature Rescaling (OFR) and consists in multiplying the variables of the problem, i.e. the features of the input dataset, by an optimal parameter set obtained by solving a global optimization problem. Since the training of the neural network is non-convex, the performances of the network should have improved, demonstrating the effectiveness of this approach.

In order to solve the global optimization problem a loss function and a solver were selected. The considered objective function takes as inputs the decision variables, i.e. the rescaling parameters, and the training and validation sets, performs the feature rescaling using the input parameter vector, then trains the chosen neural network and returns as output the Root Mean Squared Error (2.1) calculated on the validation data. As solver, a Genetic Algorithm was chosen.

The Optimal Features Rescaling approach was then applied to a real scenario to test its efficacy: the roundness prediction for a centerless grinding machining process. Because the workpiece is not fixed during the grinding process, the roundness prediction is a difficult task and represents the perfect scenario on which to test the OFR technique.

In principle the dataset was prepared. Each sample was composed by 13 variables: 12 were the centerless grinding process parameters, while the last was the output of the low fidelity model, i.e. a simplified version of the more complex process model (the high fidelity one).

A first neural network was then trained on 4 different datasets:

- the raw data;

- a standardized one, i.e. from each feature was removed the mean and the variance;
- an optimal rescaled one starting from the raw data;
- an optimal rescaled one starting from standardized data.

The results demonstrated the OFR efficacy, in fact the cross-validated performances were better than the two baselines. Nevertheless, the performances of the model were poor.

For this reason an Early Stopping technique was implemented in order to stop the training of the network when the error on validation data would begin to rise. Despite the fact that the obtained performances still confirmed the good qualities of the optimal rescaling approach, the overfitting problem did not disappear, demonstrating that a preliminary model selection was needed.

A simplified neural network was then tested with the Early Stopping technique in order to counteract the overfitting issue. In that case the problem was almost completely solved, and the cross-validated performances obtained on the optimal rescaled dataset still remained the best.

In the end, a particular application of the optimal rescaling method was tested. Because the Feed Forward Neural Network rescales the first layer inputs with its weights, the optimal parameters used in the OFR method can be incorporated in the first layer, allowing the network to perform optimal rescaling internally and thus obtaining results that are equal to the ones achieved by pre-processing the data outside the network. Furthermore, the possibility of conducting from one form of rescaling to another granted by the non-convexity of the training problem justifies the use of global optimization techniques in order to globalize the training of the neural network.

In conclusion, all the techniques described in this thesis highlighted the pros guaranteed by the use of the OFR approach, making it an interesting topic for future studies.

4.2. Future developments

This thesis has shown the potentiality offered by the Optimal Feature Rescaling. Nevertheless, several aspects of the method can be further analyzed to understand its efficacy in the machine learning and deep learning fields. A list of possible future developments derived from the tests used in this thesis is proposed here:

- the OFR method could also be tested on more complex problems compared to the case analyzed during this work. In a case with more variables, the time required by the solver to provide the optimal parameters could be too much compared to the

benefits guaranteed by the method. In that case, a classical standardization could be more suitable;

- at the same time, OFR could also be tested on a technique with less computational requirements like k-nearest neighbor. In that case, conversely, the trade-off between time and performances could be advantageous, justifying the application of the method;
- concerning the FFNN, a more accurate model selection phase could be followed, in order to understand how OFR performs on a network refined for the considered problem;
- considering the results obtained setting the FFNN first layer weights, the OFR technique could be applied in order to set the weights for all the other network layers. However, this approach would probably be prohibitive, since the size of the search space of the global optimization problem becomes very large by using many levels and neurons;
- last but not least, a different solver, like the one developed by Lorenzo Sabug Jr., Fredy Ruiz and Lorenzo Fagiano (SMGO [7]) could be applied in order to seek a more efficient way to obtain the optimal parameters for OFR.

Bibliography

[1] C. C. Aggarwal. *Neural Network and Deep Learning.* Springer International Publishing AG, part of Springer Nature, 2018.

[2] C. M. Bishop. *Pattern recognition and machine learning.* Springer Science+Business Media, LLC, 2006.

[3] F. Chollet. *Deep learning with Python.* Manning Publications Co., 2018.

[4] Q. Cui, K. Cheng, and H. Ding. An analytical investigation on the workpiece roundness generation and its perfection strategies in centreless grinding. 2014.

[5] L. Fagiano. Constrained numerical optimization for estimation and control. 2020.

[6] L. Hamm, B. Brorsen, and M. T. Hagan. Comparison of stochastic global optimization methods to estimate neural network weights. 2007.

[7] L. S. Jr., F. Ruiz, and L. Fagiano. Smgo: A set membership approach to data-driven global optimization. 2021.

[8] M. Leonesio and L. Fagiano. A semi-supervised physics-informed classifier for centerless grinding operations. 2022.

[9] L. Liberti. Introduction to global optimization. 2006. URL `https://www.researchgate.net/publication/228901769`.

[10] S. Luke. Essentials of metaheuristics. 2013. URL `https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf`.

[11] H. Safarzadeh, M. Leonesio, G. Bianchi, and M. Monno. Roundness prediction in centreless grinding using physics-enhanced machine learning techniques. 2020.

[12] R. S. Sexton, R. E. Dorsey, and J. D. Johnson. Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation. 1988.

[13] Y. Shang and B. W. Wah. Global optimization for neural network training. 1996.

[14] T. Weise. *Global Optimization Algorithms – Theory and Application.* 2009.

A | Appendix A

In this appendix, the main parts of the optimal feature rescaling code are reported:

- Main code;
- genetic algorithm code;
- the used loss functions.

Listing A.1: OFR Main.

```
# Basic libraries
import numpy as np
import pandas as pd

# Others
import time
from loss_genetic_regress import cost_function_NNregress_complex_noES
from loss_genetic_EStopping import cost_function_NNregress_complex_ES,
    cost_function_NNregress_simple_ES
import ga
from ypstruct import structure

# Preprocessing
from sklearn.preprocessing import StandardScaler

# Grab Currrent Time Before Running the Code
start = time.time()

# Import data
train = pd.read_csv("/Users/federicovitro/Desktop/Tesi_Magistrale/
    Step_11/Dataset/train_base.csv")
val = pd.read_csv("/Users/federicovitro/Desktop/Tesi_Magistrale/Step_11/
    Dataset/val_base.csv")
test = pd.read_csv("/Users/federicovitro/Desktop/Tesi_Magistrale/Step_11
    /Dataset/test_base.csv")

# I/O definition
```

```
X_train = train.loc[:,'x0':'x12']
y_train = train['t']
X_val = val.loc[:,'x0':'x12']
y_val = val['t']
X_test = test.loc[:,'x0':'x12']
y_test = test['t']

#%% De-comment if you want to standardize the data
# scaler = StandardScaler()
# Xs_train = scaler.fit_transform(X_train)
# Xs_val = scaler.transform(X_val)
# Xs_test = scaler.transform(X_test)

# columns = ['x0','x1','x2','x3','x4','x5','x6','x7','x8','x9','x10','x11','x12']
# Xs_train = pd.DataFrame(Xs_train, columns=columns)
# Xs_val = pd.DataFrame(Xs_val, columns=columns)
# Xs_test = pd.DataFrame(Xs_test, columns=columns)

#%% Problem definition
N_decision_var = X_train.shape[1]
problem = structure()
# problem.costfunction = cost_function_NNregress_complex_noES
# problem.costfunction = cost_function_NNregress_complex_ES
problem.costfunction = cost_function_NNregress_simple_ES
problem.nvar = N_decision_var # number of decision variables
problem.varmin = -3 # decision variables min bound (in log-scale)
problem.varmax = 3  # decision variables max bound (in log-scale)

problem.xtrain = X_train
problem.xval = X_val
# De-comment if you want to standardize the data
# problem.xtrain = Xs_train
# problem.xval = Xs_val
problem.ytrain = y_train
problem.yval = y_val

#%% Genetic Algorithm (GA) parameters
params = structure()
params.maxit = 100 # max iterations
params.npop = 20 # max population elements
params.pc = 1 # proportion of the children compared to the initial population
params.gamma = 0.1
```

```
params.mu = 0.2 # normal distribution mean and variance
params.sigma = 0.1

#%% Run GA
out = ga.run(problem, params)

# Grab Currrent Time After Running the Code
end = time.time()

# Subtract Start time from the End time
total_time_s = end - start
print("\ntotal time = "+ str(total_time_s)+" [s]")
total_time_h = (end - start)/3600
print("\ntotal time = "+ str(total_time_h)+" [h]")
total_time = np.zeros((1,2))
total_time[0,0] = total_time_s
total_time[0,1] = total_time_h

#%% Save the execution time
np.savetxt('/Users/federicovitro/Desktop/Tesi_Magistrale/Step_11/
    GA_ex_time/total_time_complexNN_siEStopping_Standardized.csv',
    total_time, delimiter=',')

#%% Save Results (best population + best solution)
columns = ['th0', 'th1', 'th2', 'th3', 'th4',  'th5', 'th6', 'th7', 'th8
    ', 'th9', 'th10', 'th11', 'th12', 'best_cost']
pop_mat = np.zeros((20, 14))

for i in range(0,20):
    for j in range(0,14):
        if j!=13:
            pop_mat[i,j] = out.pop[i].position[j]
        else:
            pop_mat[i,j] = out.pop[i].cost

pop_df = pd.DataFrame(pop_mat, columns = columns)
pop_df.to_csv('/Users/federicovitro/Desktop/Tesi_Magistrale/Step_11/
    GA_output/population/population_complexNN_siEStopping_Standardized.
    csv')

best_sol = np.zeros((1,14))

for i in range(0,14):
    if i!=13:
```

```
        best_sol[0,i] = out.bestsol.position[i]
    else:
        best_sol[0,i] = out.bestsol.cost

bestsol_df = pd.DataFrame(best_sol, columns = columns)
bestsol_df.to_csv('/Users/federicovitro/Desktop/Tesi_Magistrale/Step_11/
   GA_output/bestsol/bestsol_complexNN_siEStopping_Standardized.csv')
```

Listing A.2: Genetic algorithm.

```
import numpy as np
from ypstruct import structure

def run(problem, params):

#%% STEP:
    # 1- Initialization

    # Problem information
    costfunction = problem.costfunction
    nvar = problem.nvar
    varmin = problem.varmin
    varmax = problem.varmax

    X_train = problem.xtrain
    X_val = problem.xval
    y_train = problem.ytrain
    y_val = problem.yval

    # Parameters
    maxit = params.maxit
    npop = params.npop
    pc = params.pc
    nc = int(np.round(pc*npop/2)*2)
    gamma = params.gamma
    mu = params.mu
    sigma = params.sigma

    # Empty individual template
    empty_individual = structure()
    empty_individual.position = None
    empty_individual.cost = None

    # Best solution ever found
    bestsol = empty_individual.deepcopy()
```

```
    bestsol.cost = np.inf

    # Initialize population
    print('START Initialization')
    pop = empty_individual.repeat(npop)
    for i in range(0, npop):
        pop[i].position = np.random.uniform(varmin, varmax, nvar)
        pop[i].cost = costfunction(pop[i].position, X_train, X_val,
   y_train, y_val)
        if pop[i].cost < bestsol.cost:
            bestsol = pop[i].deepcopy()

    print('END Initialization')

    # Best cost of iterations
    bestcost = np.empty(maxit)

#%% STEP:
    # 2- Select parents & Crossover
    # 3- Mutate offspring
    # 4- Merge main population and offsprings
    # 5- Evaluate, Sort & Select
    # 6- go to step 2, if it is needed

    # Main loop
    for it in range(maxit):

        popc = []
        for _ in range(nc//2):

            # Select parents randomly
            q = np.random.permutation(npop)
            p1 = pop[q[0]]
            p2 = pop[q[1]]

            # Perform Crossover
            c1, c2 = crossover(p1, p2, gamma)

            # Perform mutation
            c1 = mutate(c1, mu, sigma)
            c2 = mutate(c2, mu, sigma)

            # Apply bounds
            apply_bounds(c1, varmin, varmax)
```

```
79              apply_bounds(c2, varmin, varmax)
80
81              # Evaluate first offspring
82              c1.cost = costfunction(c1.position, X_train, X_val, y_train,
    y_val)
83              if c1.cost < bestsol.cost:
84                  bestsol = c1.deepcopy()
85
86              # Evaluate second offspring
87              c2.cost = costfunction(c2.position, X_train, X_val, y_train,
    y_val)
88              if c2.cost < bestsol.cost:
89                  bestsol = c2.deepcopy()
90
91              # Add offspring to popc
92              popc.append(c1)
93              popc.append(c2)
94
95          # Merge, Sort and Select
96          pop += popc
97          pop = sorted(pop, key=lambda x: x.cost)
98          pop = pop[0:npop]
99
100         # Store best cost
101         bestcost[it] = bestsol.cost
102
103         #Show iteration information
104         print("Iteration {}: Best Cost = {}".format(it, bestcost[it]))
105
106     # Output
107     out = structure()
108     out.pop = pop
109     out.bestsol = bestsol
110     out.bestcost = bestcost
111
112     return out
113
114 def crossover(p1, p2, gamma=0.1):
115     c1 = p1.deepcopy()
116     c2 = p1.deepcopy()
117     alpha = np.random.uniform(-gamma, 1+gamma, *c1.position.shape)
118     c1.position = alpha*p1.position + (1-alpha)*p2.position
119     c2.position = alpha*p2.position + (1-alpha)*p1.position
120     return c1, c2
```

```

def mutate(x, mu, sigma):
    y = x.deepcopy()
    flag = (np.random.rand(*x.position.shape) <= mu)
    ind = np.argwhere(flag)
    y.position[ind] += sigma*np.random.randn(*ind.shape)
    return y

def apply_bounds(x, varmin, varmax):
    x.position = np.maximum(x.position, varmin)
    x.position = np.minimum(x.position, varmax)
```

Listing A.3: Loss function for the global optimization problem (without Early Stopping).

```
# Basic libraries
import numpy as np

# regression models
from sklearn.ensemble import RandomForestRegressor

# Neural Network
from keras.models import Sequential
from keras.layers import Dense

# Model evaluation
from sklearn.metrics import mean_squared_error

def cost_function_NNregress_complex_noES(th_log, X_train, X_val, y_train
    , y_val):

    # Make a copy of the train and validation sets
    X_train0 = X_train.copy()
    X_val0 = X_val.copy()

    # Log to Dec scale transformation
    th = 10**th_log

    # Train and validation data rescaling
    X_train0 = X_train0*th
    X_val0 = X_val0*th

    # Define NN
    nn_regressor = Sequential()

    # The Input Layer
```

```
    nn_regressor.add(Dense(128, kernel_initializer='normal', input_dim =
    X_train0.shape[1], activation='relu'))

    # The Hidden Layers
    nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))
    nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))
    nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))

    # The Output Layer
    nn_regressor.add(Dense(1, kernel_initializer='normal',activation='
    linear'))

    # Compile the network
    # lr = 0.01 #leaning rate for Stochastic Gradient Descent
    # opt = keras.optimizers.SGD(learning_rate=lr, momentum=0.95)
    nn_regressor.compile(loss='mean_absolute_error', optimizer='adam',
    metrics=['mean_absolute_error'])

    # Train the model
    nn_regressor.fit(X_train0, y_train, epochs=500, batch_size=32,
    verbose=0) # callbacks=[es]

    # Computing the Root Mean Square Error
    rmse_nn_val = (np.sqrt(mean_squared_error(y_val, nn_regressor.
    predict(X_val0))))

    return rmse_nn_val
```

Listing A.4: Loss function for the global optimization problem (with Early Stopping).

```
# Basic libraries
import numpy as np

# Neural Network
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense

# Model evaluation
from sklearn.metrics import mean_squared_error

def cost_function_NNregress_complex_ES(th_log, X_train, X_val, y_train,
```

```
    y_val):
13
14      # Make a copy of the train and validation sets
15      X_train0 = X_train.copy()
16      X_val0 = X_val.copy()
17
18      # Log to Dec scale transformation
19      th = 10**th_log
20
21      # Train and validation data rescaling
22      X_train0 = X_train0*th
23      X_val0 = X_val0*th
24
25      # Define NN
26      nn_regressor = Sequential()
27
28      # The Input Layer
29      nn_regressor.add(Dense(128, kernel_initializer='normal', input_dim =
        X_train0.shape[1], activation='relu'))
30
31      # The Hidden Layers
32      nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))
33      nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))
34      nn_regressor.add(Dense(256, kernel_initializer='normal',activation='
    relu'))
35
36      # The Output Layer
37      nn_regressor.add(Dense(1, kernel_initializer='normal',activation='
    linear'))
38
39      # Compile the network
40      opt = tf.keras.optimizers.Adam(learning_rate=0.001)
41      nn_regressor.compile(loss='mean_absolute_error', optimizer=opt,
    metrics=['mean_absolute_error'])
42
43      # Early stopping
44      callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=100, mode='min')
45
46      # Train the model
47      nn_regressor.fit(X_train0, y_train, validation_data=(X_val0, y_val),
        epochs=10000, callbacks=[callback], verbose=0)
```

```
48
49      # Computing the Root Mean Square Error
50      rmse_nn_val = (np.sqrt(mean_squared_error(y_val, nn_regressor.
        predict(X_val0))))
51
52      return rmse_nn_val
53
54  def cost_function_NNregress_simple_ES(th_log, X_train, X_val, y_train,
        y_val):
55
56      # Make a copy of the train and validation sets
57      X_train0 = X_train.copy()
58      X_val0 = X_val.copy()
59
60      # Log to Dec scale transformation
61      th = 10**th_log
62
63      # Train and validation data rescaling
64      X_train0 = X_train0*th
65      X_val0 = X_val0*th
66
67      # Define NN
68      nn_regressor = Sequential()
69
70      # The Input Layer
71      nn_regressor.add(Dense(X_train0.shape[1], kernel_initializer='normal
        ', input_dim = X_train0.shape[1], activation='relu'))
72
73      # The Hidden Layers
74      nn_regressor.add(Dense(100, kernel_initializer='normal',activation='
        relu'))
75
76      # The Output Layer
77      nn_regressor.add(Dense(1, kernel_initializer='normal',activation='
        linear'))
78
79      # Compile the network
80      opt = tf.keras.optimizers.Adam(learning_rate=0.001)
81      nn_regressor.compile(loss='mean_absolute_error', optimizer=opt,
        metrics=['mean_absolute_error'])
82
83      # Early stopping
84      callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
        patience=200, mode='min')
```

```
85
86      # Train the model
87      nn_regressor.fit(X_train0, y_train, validation_data=(X_val0, y_val),
        epochs=10000, callbacks=[callback], verbose=0)
88
89      # Computing the Root Mean Square Error
90      rmse_nn_val = (np.sqrt(mean_squared_error(y_val, nn_regressor.
       predict(X_val0))))
91
92      return rmse_nn_val
```

List of Figures

List of Tables

Ringraziamenti

Ringrazio il Prof. Lorenzo Fagiano per la disponibilità e la pazienza dimostratami durante la realizzazione della tesi e l'Ing. Marco Leonesio per l'aiuto fornitomi: è stato un compagno di viaggio insostituibile. Ringrazio i miei genitori, mia sorella e gli altri familiari per l'incoraggiamento, il sostegno e, soprattutto, per avermi sopportato in questi anni di duro lavoro. Per ultimi, ma non meno importanti, i miei amici, che mi sono stati accanto distraendomi quando la pressione diventava troppa. Il mio percorso universitario volge al termine dopo molta fatica, lasciandomi tuttavia la consapevolezza di aver concluso qualcosa di grande. Ora sono pronto per altre avventure.