C++ Multi-Layer Perceptron

Neural network for 2D localization with input data from environmental sensors with the presence of noise.

Delivery of project AA1 2013

Francesco Brundu
matr. 438905

September '14

Summary

Introduction

For the Cup it is required to develop a multi-Layer feedforward neural network, to solve a regression task on two target variables, representing the 2D coordinates (x, y).

A training set with 990 training examples is provided in the format <id, var1, var2, var3, var4, var5, target X, target Y> where column 1 pattern name (id), the middle 5 columns are continuous value variables (from environmental sensors) there is some noise, and the final 2 columns are the target of two continuous values representing the position on the X and Y axis respectively.

A blind test set is also provided: 353 data, same input format, 2 columns of target are missing.

To learn the model, the back-propagation algorithm is used in its in-line (stochastic) version.

The loss used for the evaluation of the performance in the competition is the Mean Euclidean Error:

= 1/— ÿÿ ÿ ÿ =1 ² = 1/— ÿÿ(=1)² + ( )²

La loss per il training è il (Root) Mean Squared Error (loss del least means square)

= 1/— ÿ(ÿ) =1 ² = 1/— ÿ((, =1)² + ( ) 2)

While the modified loss for Weight decay is:

( ) = ( ) + ÿ/— wH = ( ) + ÿ/— ÿ ÿ ÿ((l) =0 =1 d (lÿ1) d (l) =1)²

The developed neural network was first tested on test datasets (monk) to solve classification problems with and without the presence of noise.

Later test datasets were created for regression problems with and without the presence of noise.

Finally, having verified the neural network's ability to solve the problems described above, the network was trained on the dataset to solve the competition task. During the various tasks, the performance of the developed network was compared with that of external tools made available by knime.

The various datasets mentioned above will be described later. For the pre-processing of the data I used the knime and excell software, also used for the comparison of the results with other models, the post-processing and the visualization of data and results.

Method and design choices

The implemented neural network (feedforward multi-layer perceptron) is a multi-level network with the units of one level connected with all those of the next level. The user through a string decides the number of levels and the number of units per network level, a number representing the number of units of the level separated with a space from the number representing the units of the next level. For example "2 4 1" to indicate a network with 2 input units, 4 hidden units and one output unit. During the creation of the network for each level, in addition to those indicated by the user, a last unit of bias is added with an output set at 1. The weights associated with the synapses that connect a neuron to all those of the next level are modeled with an internal weight vector
to each unit. Each layer / layer is implemented as a unit / neuron vector and the network is a layer vector.
To make the code more uniform and avoid unnecessary complexity, all units and layers are the same, which implies that the output units also have an output weight vector, which will be ignored, and a baias unit also ignored. For the cup the net with a hidden level was used.

For network training, the standard back-propagation algorithm is used in its On-line version. The activation function adopted is the logistic / sigmoid function, this speeds up the backpropagation algorithm since the output value of the unit calculated in the feedforward phase is reused for the calculation of the gradients of the output units.

The network provides different training modes, one through the train function, which implements the training proposed for these tasks, and one through the feedforward, backprop, getresults etc. functions. which allow to define another learning modality according to other criteria.

At the beginning of the training the weights are initialized with small random values (-0.5, 0.5), this allows the algorithm to start each time from a different hypothesis, so two consecutive workouts on the same data do not end up on the same local minimum. This allows to calculate an estimate of the error committed by a model also as the average error between various "trials" with the same data and parameters.

Another precaution adopted is to randomize the order of the training patterns before each epoch.

The weights update rule has been tested in various versions, the first derived from the MSE error and the second derived from the MEE, the first was chosen as it provides a faster convergence than the second which divides the delta obtained by the units of output from the first for the norm of the total error committed.
The function minimized by the training algorithm is in the case in which the data does not present noise, and in the case of noisy data a modified version with a regularization component based on the value of the network weights. weight decay, which favors solutions with small weights . The effect of this regularizer is to reduce the non-linearities / abrupt jumps in the calculated function, with the assumption that the noise favors these solutions instead. You switch from one to the other by specially modifying the lambda parameter, eg. with 0 the complexity penalty is not counted and the standard backpropagation is obtained. In both weight update rules, moment is used.

The network can solve both classification and regression problems, which can be decided via a parameter when the network is created. In the case of a regression problem linear output units are used, which translates into the feedforward phase in the use of the identity activation function for these units, in the backpropagation phase the derivative for the calculation of the error of the units of output is therefore 1.

The proposed network can use both indirect (early stopping) and direct (weight decay) regularization.

Main stop criteria taken into consideration (verified every k = 3 epochs):

- Excessive number of epochs;
- Loss on the internal validation set below a given threshold; (method not used for the Cup task)
- The gradient is below a given threshold (eg 0.0001), implying a local minimum point; (method not used for the Cup task)
- In the case of an **early stop ,** the error increases on an internal validation set, saving the network weights when a new minimum occurs, after which it continues for a predetermined number of epochs, and if in this interval has not occurred a new minimum you can interrupt the training; The saved weights are restored and the model returned.

The validation set inside the training procedure in the case of early stopping is created within the train method and is kept separate from the data with which the network is trained.

Z-score normalization:

$$Normalized\ (e_i) = \frac{e_i - \bar{E}}{std(E)}$$

Normalization [0,1] of inputs and targets according to the function:

$$Normalized\ (e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

Where: Emin = minimum value for variable E. Emax = maximum value for variable E.

Internal code denormalization of targets and output, where - Emin corresponds to translate and Emax - Emin to scale. Function used to calculate the MEE error in the original scale, a procedure used only in the construction of the final model and in the output of the blind test results.

Model selection and evaluation of the product model

The error estimation on each model has been designed and implemented with a cross validation procedure with k variable and simple validation. Features made available by the classes developed for the management of datasets.

In the case of cross validation I choose a k = 3 or 5. A small k produces a bias due to the validation set, a common heuristic is to choose k = 10, but computationally too expensive. The dataset is divided into k sections, one is used in turn to estimate the generalization error and the rest to train the model.
Finally, the estimated error of a given model is the average generalization error over all folds. While in the case of simple validation (holdout) I use 1/3 of the data as validation set and the rest as training set, and the estimate is made on the single validation set, in this case we have a less accurate estimate, but the time to make it it is considerably less. For model selection then I'm going to use a holdout.

To estimate the error, the same fold is performed 3 times (10 trials is too expensive), so as to have different starting hypotheses and arrive at a different local minimum, and the error on that fold is estimated as the average on the trials.
Through the procedure just described I choose the final model.

The models to choose from here are defined by the hyper-parameters chosen, the choice is made by trying various combinations between a grid of values to be associated with the various parameters to be chosen, estimating their effectiveness through the validation process.

Hyper-parameters optimized with a grid of values, heuristics chosen by trying the various limit values, and those chosen seem to be quite representative:

- number of hidden units chosen from: {min, .., max} where min and max vary according to need;
- ÿ learning rate chosen from: {1 / 2i } i, then I used the grid {0.5, 0.1, 0.01, 0.001};
- ÿ chosen from: {1/2 ^ i} later I used the grid {0.0, 0.1, 0.5, 0.7} instead;
- ÿ chosen from: {1/2 ^ i} later I used the grid {0.1, 0.01, 0.001, 0}.

A variant proposed by the haykin for the choice of the number of hidden neurons, looking for the lowest number of neurons necessary to achieve the performance of a Bayesian classifier (within 1%), starting from 2 neurons and increasing if necessary. I used a variant in which I also compare myself with an rprop MLP.

For the choice of the values for the learning parameter ÿ and the moment constant ÿ it provides ÿ, ÿ which on average:

- they converge to a local minimum in the least number of epochs;
- or who obtain the best generalization in the least number of epochs.

An improvement is to use this procedure to detect a range on which to then search with a more accurate grain. This proposal was not always implemented due to the long processing times, but only when described in the experimental chapter.

The test function provides a parameter that enables the printing of the training and validation error on file "error.csv" (in the case of validation set within the training) and the printing of the results on "output.csv" so as to be able to make the analysis and comparison graphs between network output and target. The blind test instead prints the results on "finalout.csv". The network accepts input for training and validation via a csv file “input.txt” and input for testing via “test.txt”.

Structure of the code

I briefly describe what are the main components developed, available as an attachment and extensively commented in the code. The comments in the code are given in English, as are all the rest of the code.

The code has been designed to be modular so as to be able to replace or modify functions such as the activation function of a unit without having to touch the code relating to network management.

Neuron class: Represents the basic entity of the network. It implements the basic functionality of a unit, such as calculating the output through an appropriate activation function defined internally in the class, or updating the weights of the input connections based on the chosen formula.

Net class: class that implements the actual network and coordinates its functioning and learning, through the composition of neurons. Coordinates learning operations such as feedforward and backprop, and those of testing or applying the network.

Csvdataloader class: Offers basic functionality to read a csv file line by line, avoiding comments and blank lines.

Datasetmanager class: Using the csvdataloader loads the dataset inside an array, so as to load the dataset into memory. It also offers the possibility to perform a cross validation of k folds defined in the constructor, providing an iterator on fold which for each fold initializes a train and a validation dataset.

The code for the creation of the synthetic regression task to be used for testing the network was also developed separately. The latter will be described in the next chapter.

While fungenerator is the code related to the creation of the synthetic regression dataset, it very simply creates two files funout_test.csv and funout.csv in which it saves the function without the noise and the one with the noise, respectively.

Experimental design

Three phases of testing and experimentation are foreseen for the development and verification of the network in preparation for the competition task.

1. Network testing phase during implementation with a customized dataset representing the xor function, 2 inputs and a target without errors;
2. Network testing phase with the 3 Monk classification problems, with 6 discrete attributes in 1ofk encoding as input and a binary target, the first two problems are without noise, while the third is not, so in the latter case it is possible try with regularization techniques to improve the result obtained without; such techniques as proof must then be able to learn the concept expressed by the two previous problems;
3. Network testing phase with a synthetic regression problem on two target variables given an input, the dataset is noisy; This is the last test on a dataset other than that of the competition.

Monk problems are classification problems on six discrete attributes and each training / test data is of the form: <attribute # 1> <attribute # 2> <attribute # 3> <attribute # 4> <attribute # 5> <attribute # 6> -> <class>

attribute#1, attribute#2, attribute#4 ÿ {1, 2, 3}

attribute#3, attribute#6 ÿ {1, 2} attribute # 5 {{1, 2, 3, 4} class ÿ {0.1}

1. MONK-1: (attribute_1 = attribute_2) or (attribute_5 = 1)
2. MONK-2: (attribute_n = 1) for exactly two n (chosen in {1,2, ..., 6})
3. MONK-3: (attrib_5 = 3 and attrib_4 =1) or (attrib_5 != 4 and attrib_2 != 3)
    3.1. with 5% noise

A detailed description of the monks can be found on the repository site [6].

Experimental results

Network testing

First acceptance test with a noiseless dataset representing the xor function. The network was used in its simplest version with two input units, three hidden units and one output with sigmoidal activation function, with eta 0.3 and alpha 0.1, the stop criterion used measures the loss on the training set by comparing its value with the symbolic threshold of 0.001. This was possible due to the fact that the dataset has no noise and is
representative of the objective function to be searched. After about 800 epochs the loss (MSE) assumes values below the predetermined threshold.

This is due to the target chosen in the dataset which is 1 and 0 respectively positive and negative, in fact by slightly modifying the targets respectively to 0.9 and 0.1 there is a faster convergence without saturating the weights of the units in about 600 epochs.

With a linear output activation function, thus performing a regression we obtain a training error (MSE) after about 120 epochs <0.001.

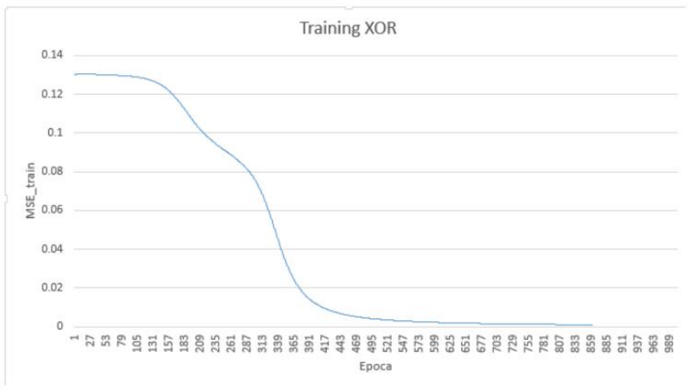


Figure 1- Training con output unit sigmoidale.

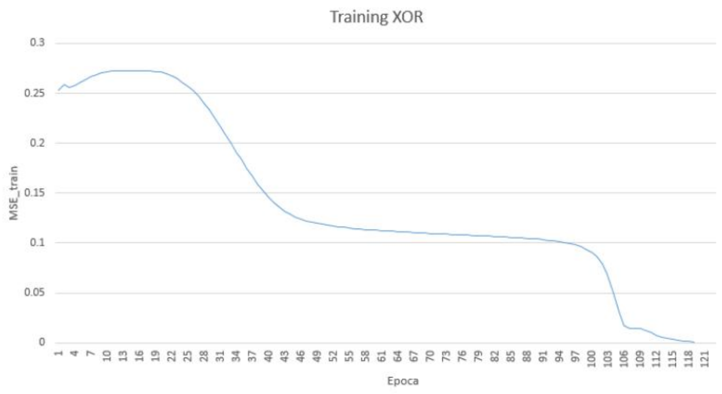
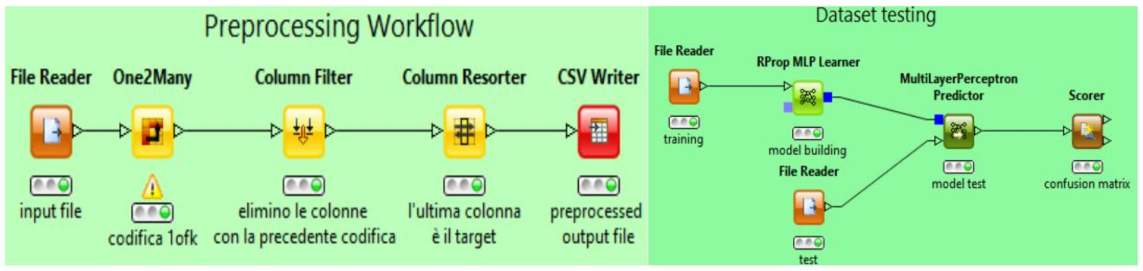


Figure 2- training con output unit lineare.

Monk 1

The Monk 1 dataset was applied to a network with 4 hidden units, a learning rate of 0.3 and an alpha coefficient for momentum of 0.3.

The input taken from the repository, in the format with columns separated by a space, was preprocessed in a format compatible with knime, eliminating the first space which is recognized as an empty column, and at this point the input features have been encoded 1ofK, producing 18 binary features of which the last one as target.
The training file used is the csv resulting from this preprocessing.



To ensure the correct functioning of the preprocessing, I verify the actual correctness of the data through an external knime tool that implements a multi-layer perceptron; The learning algorithm used is RProp, 4 hidden units (with 2 and 3 it fails to correctly learn the objective function) and 1000 limit iterations.
Learning converges with less than 90 epochs reaching 100% accuracy.

At this point I will check if my network can do the same. The units have a sigmoidal activation function. Several trials have been performed with random initialization of the unit weights, and with this example configuration 100% accuracy is obtained.

Already after a few epochs both the training error and the validation error approach zero, it is observed that initially the two deviate due to the initial settlement of the network, which then continues in an asymptotic convergence, stopped as soon as the validation error falls below the 0.001 threshold.

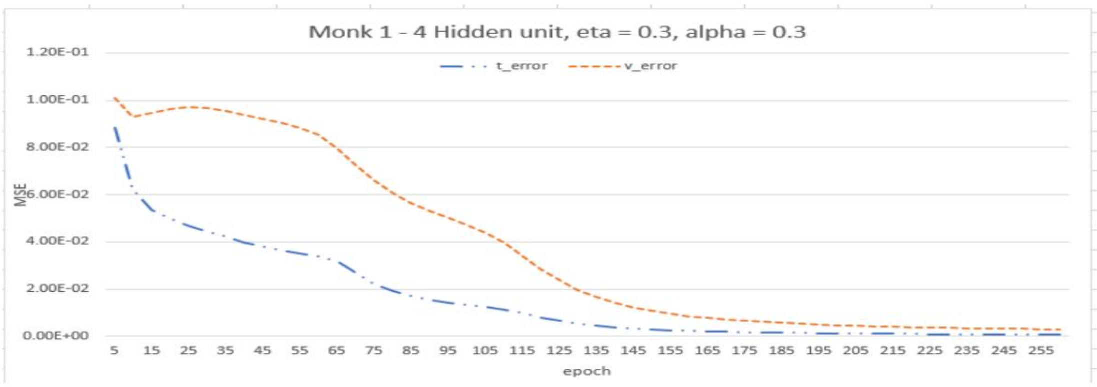
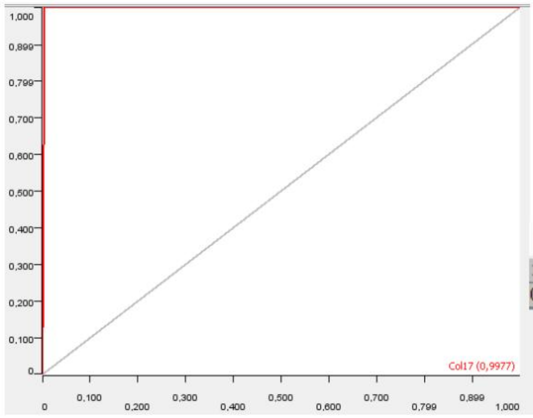
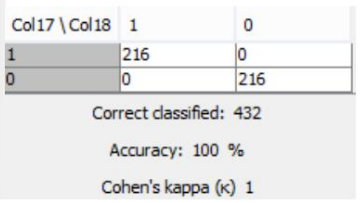


Figure 3-Curve di training error e validation error.

Same preprocessing for the test set file applied to the network with a specific secondary file for the final test on the product model, from which a file with the results was generated, and comparing them with the target results, the confusion matrix was calculated and the other measures shown.

The results on the test set are reported on a file in csv format, and with the appropriate knots / functionalities of knime the results have been analyzed through the confusion matrix and the ROC curve





Col17 \ Col18	1	0
1	216	0
0	0	216

Correct classified: 432

Accuracy: 100 %

Cohen's kappa (κ) 1

Figure 4-ROC curve and confusion matrix with accuracy.

Monk 2

The Monk 2 dataset was applied to a network with 4 hidden units, a learning rate of 0.3 and an alpha coefficient for momentum of 0.3. The input was decoded 1ofK with knime, producing 18 binary features of which the last one as a target. All units use the sigmoid type activation function. Several trials have been performed with random initialization of the unit weights, and with this example configuration 100% accuracy is obtained. Already after a few epochs both the training error and the validation error approximate the
zero.

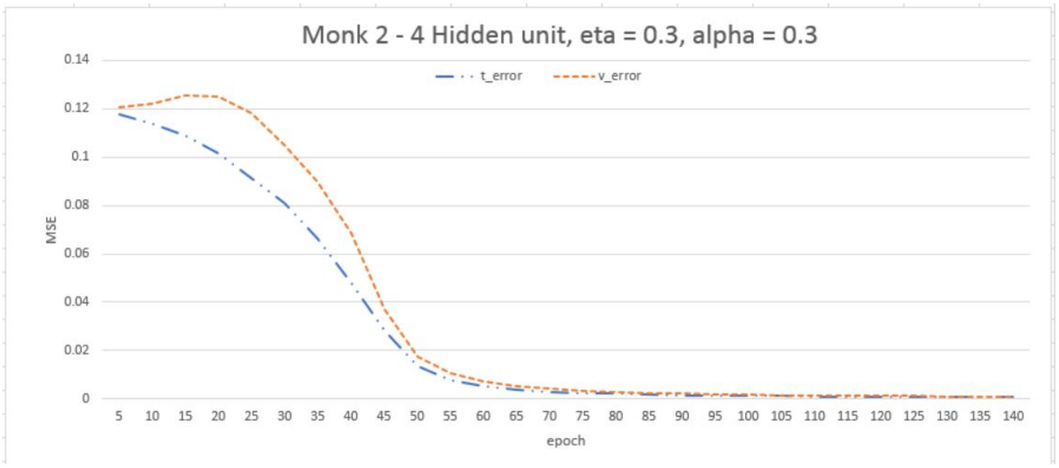
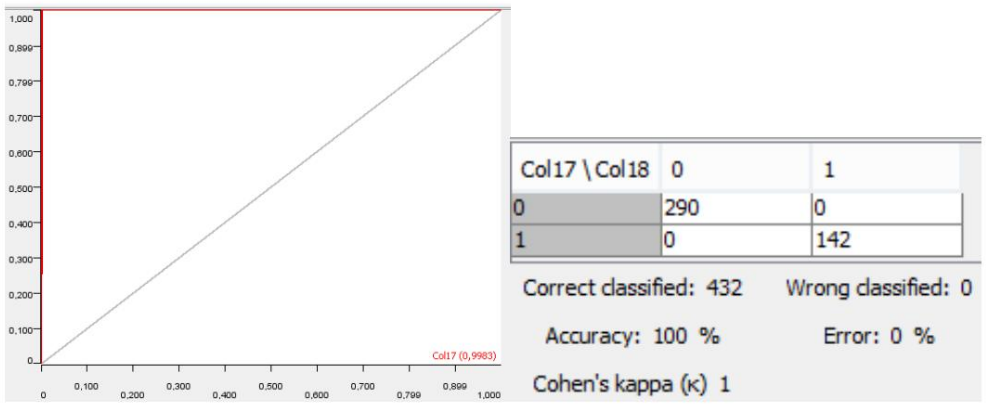


Figure 5-Training curve with train error and validation error.

The results on the test set are reported on a file in csv format, and with knime the results were analyzed through the confusion matrix and the ROC curve.



Col17 \ Col18	0	1
0	290	0
1	0	142

Correct classified: 432 Wrong classified: 0

Accuracy: 100 % Error: 0 %

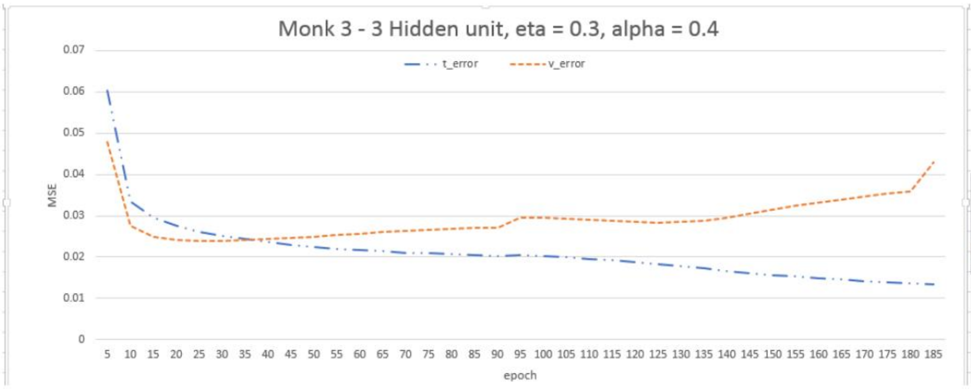
Cohen's kappa (κ) 1

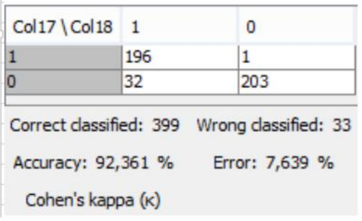
Figure 6-ROC curve and result confusion matrix.

Monk 3

With 2 hidden units ÿ, ÿ constant, the network obtains 91.6% accuracy on the validation set, which does not change by increasing the number of units while keeping the parameters constant. Less than those used in the comparison paper (4 units). I used three hidden units, eta 0.3 and alpha 0.4, with a stopping criterion based on the error in validation, when it perceived a 10% increase in the error the algorithm ended.

The result on various trials is maintained around 92% accuracy, observing the already known presence of noise in the data.





Col17 \ Col18	1	0
1	196	1
0	32	203

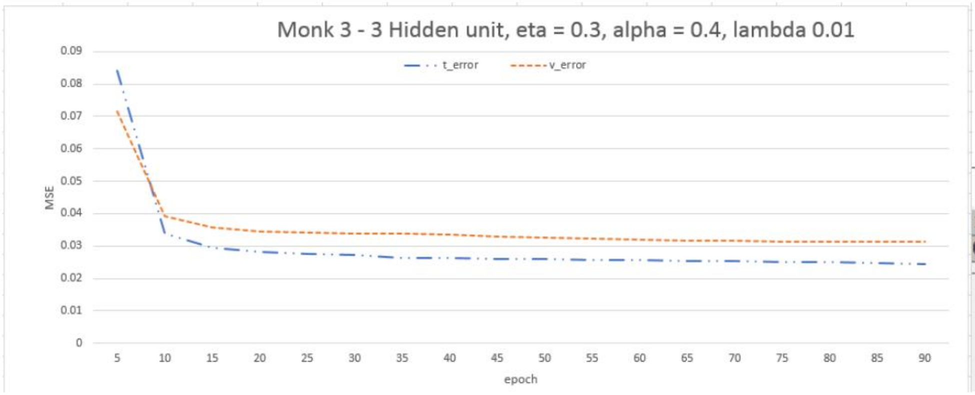
Correct classified: 399 Wrong classified: 33

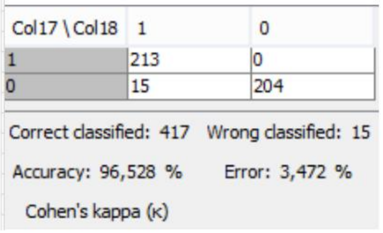
Accuracy: 92,361 % Error: 7,639 %

Cohen's kappa (κ)

Figure 7- Graph of the training curves and confusion matrix.

I try to apply the weight decay regularization with lambda 0.01 and I observe on various runs an average improvement in accuracy, around 96-97%.





Col17 \ Col18	1	0
1	213	0
0	15	204

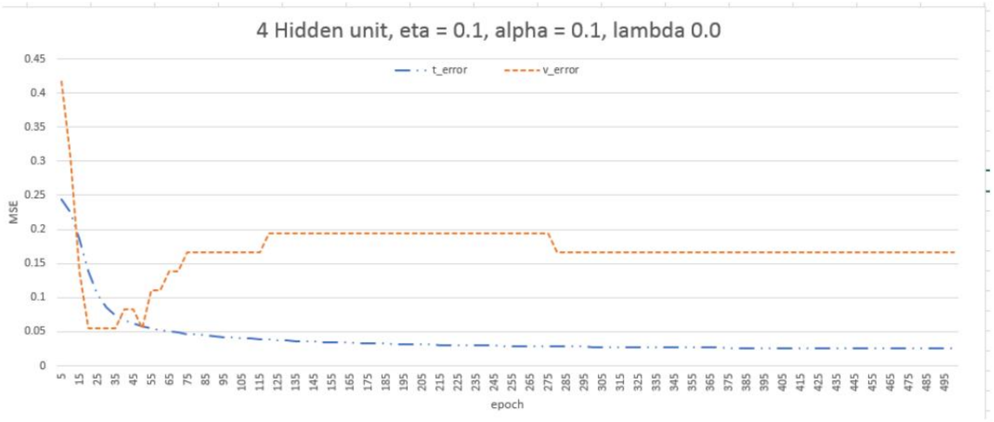
Correct classified: 417 Wrong classified: 15

Accuracy: 96,528 % Error: 3,472 %

Cohen's kappa (κ)

Figure 8- Graph of the training curves and confusion matrix.

The weight decay regularization was also applied with the two previous problems and confirmed the generalization capacity with an accuracy of 100%.



Same result (96.7 accuracy on test set) achieved with early stopping, and another model with 4 hidden units parameters alpha = 0.1 eta = 0.1. In 20 epochs the minimum error was reached on the validation v_error represents the error rate = 1-

set. The

accuracy.

Figure 9- Graph of the training curves.

For the sake of completeness, it is noted that by running the algorithm several times the error on the external test set still has a certain variability, due to the starting point among others, it also reaches a peak of 98%, as well as during the various trials during the model selection. It should be noted that the test set is not used to make decisions, but only as a final evaluation of the chosen model.

Regression 1

So far the network has been tested on classification problems, since the competition problem is a regression problem on two target variables, for the final test I use an artificially constructed regression problem. The preprocessing of this problem consists in the normalization of the input features, in order to have mean 0 and standard deviation 1. The dataset was created specifically by applying a noise with mean 0 and standard deviation 0.2 to the generated targets. The domain of x was generated with a uniform distribution between -3 and 3.

ÿ , + 2 ÿ sin(1.5 ÿ) + (0,0.2) , + 2 ÿ cos(0.5 ÿ) + (0,0.2)ÿ

The two target functions are shown in the figure. Function with adding noise as training input, test set instead without.

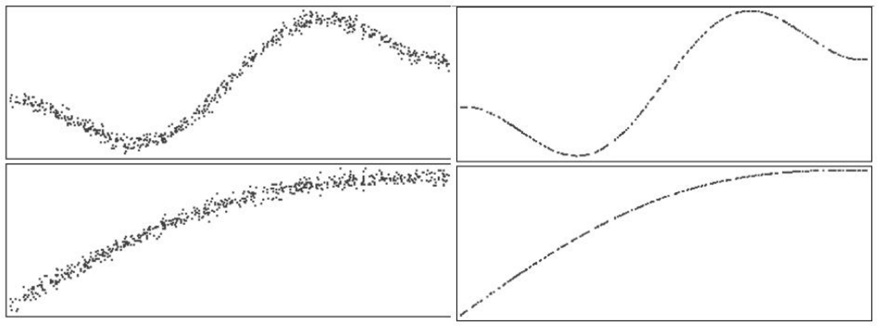


Figure 10-Graph of training and test targets.

The network in this case is configured to have 1 input unit and 2 linear output units, the number of hidden units is chosen by trying various configurations and choosing the one with the least number of units and the best generalization error, measured on a validation set.

Training chart with early stopping, which quickly reaches the minimum error value in the first 150 epochs.

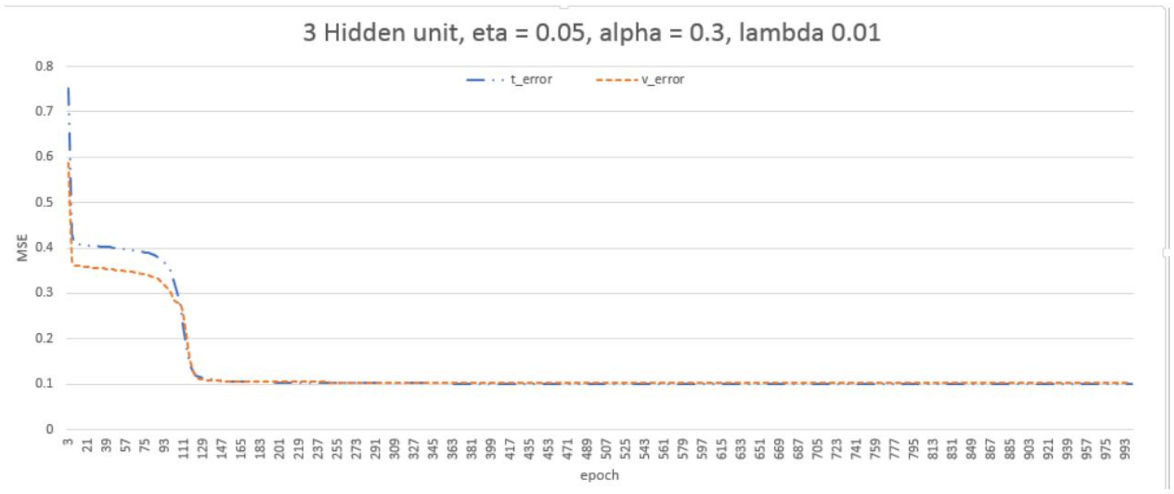


Figure 11- Graph of the training curves.

Loss (MSE) calculated on the test set = 0.013

The network with 3 hidden units was able to approximate the searched output very well, with 2 instead there were greater errors and graphically it was not possible to obtain the correct curve.

The correlation (Pearson's product-moment coefficient) between the target y1 and the calculated one is 0.999 as can be seen graphically from the plot of y1 with respect to the prediction y1. Similar results were obtained for y2.

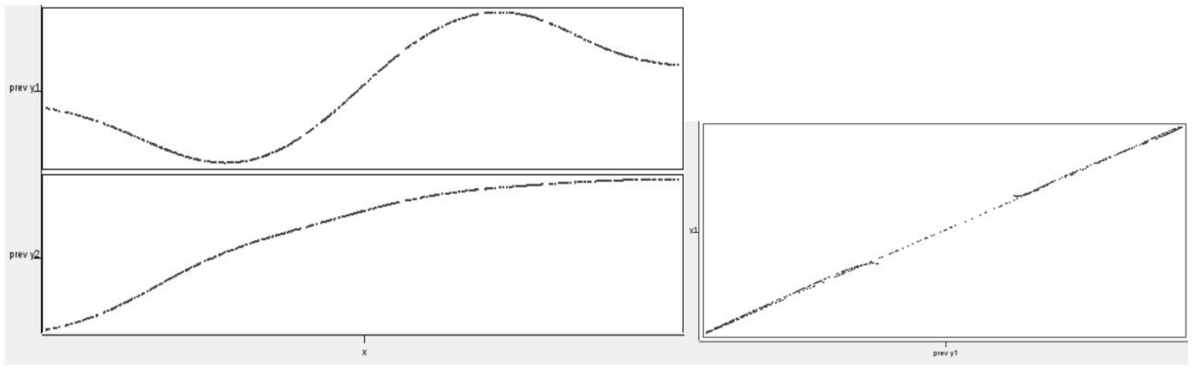


Figure 12-Graph of the output calculated by the network and graph of y1 and y1 calculated in comparison.

This type of analysis will be repeated for the Cup task, as it provides a simple tool to verify the ability to correctly predict the target.

Cup task

Preliminary analysis and preprocessing

Let's start with the analysis of the statistics on the dataset provided. There are no missing values. The inputs have an unbalanced distribution, as does the target variable y (Col6), while the target variable x (Col5) has a uniform distribution over the entire range of values.

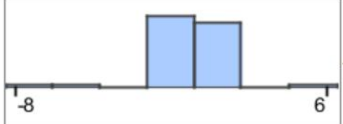
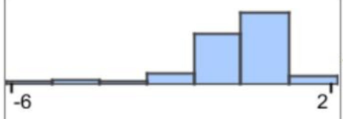
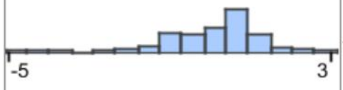
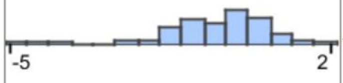
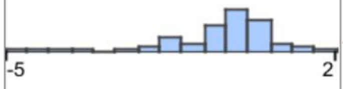
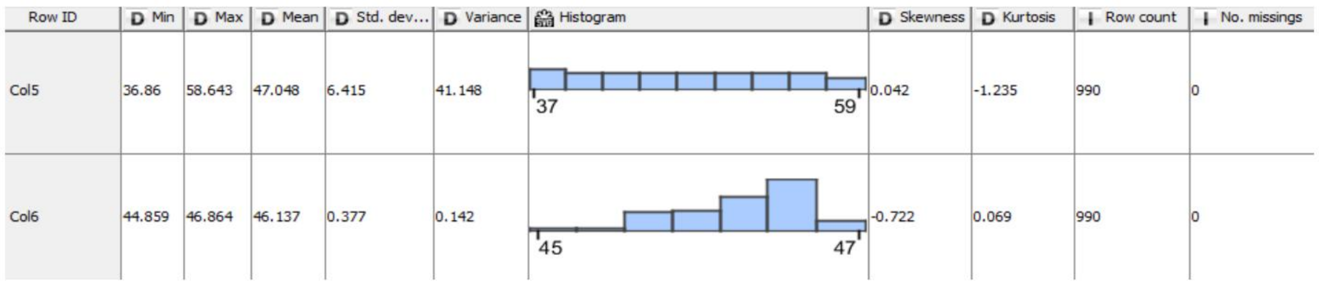
Row ID	Min	Max	Mean	Std. dev...	Variance	Histogram	Skewness	Kurtosis	Row count	No. missings
Col0	-7.567	5.627	0.001	1	0.999		-1.556	19.305	990	0
Col1	-5.819	2.263	-0.021	0.997	0.994		-1.469	5.557	990	0
Col2	-4.665	2.623	-0.014	0.991	0.983		-1.275	3.881	990	0
Col3	-4.753	2.478	0.002	1.003	1.005		-1.005	3.267	990	0
Col4	-4.997	2.484	0.035	1.021	1.043		-1.558	4.741	990	0

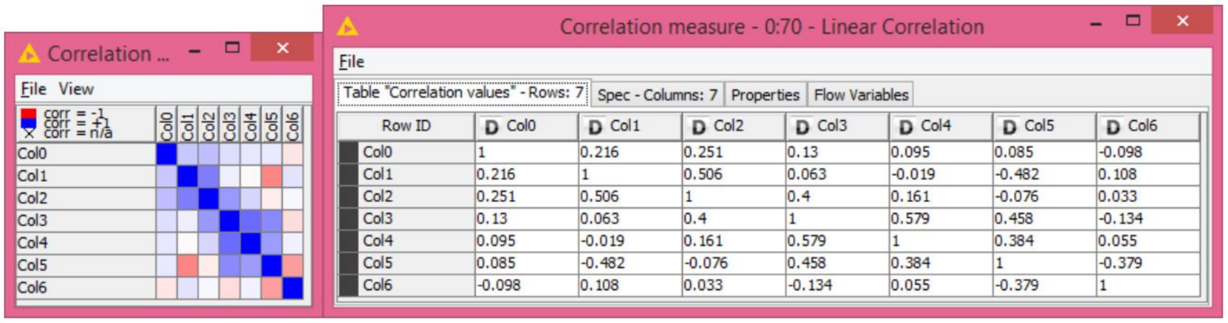
Figure 13-Statistics of the input variables.



Row ID	Min	Max	Mean	Std. dev...	Variance	Histogram	Skewness	Kurtosis	Row count	No. missings
Col5	36.86	58.643	47.048	6.415	41.148	37 … 59	0.042	-1.235	990	0
Col6	44.859	46.864	46.137	0.377	0.142	45 … 47	-0.722	0.069	990	0

Figure 14-Statistics on target variables.

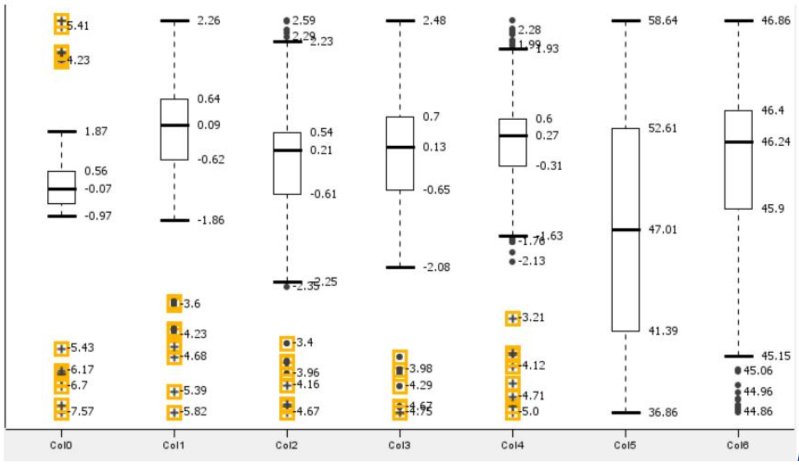
From Pearson's product-moment coefficient we see that the two target variables have a negative correlation, and that some of the inputs are correlated to each other.



Row ID	Col0	Col1	Col2	Col3	Col4	Col5	Col6
Col0	1	0.216	0.251	0.13	0.095	0.085	-0.098
Col1	0.216	1	0.506	0.063	-0.019	-0.482	0.108
Col2	0.251	0.506	1	0.4	0.161	-0.076	0.033
Col3	0.13	0.063	0.4	1	0.579	0.458	-0.134
Col4	0.095	-0.019	0.161	0.579	1	0.384	0.055
Col5	0.085	-0.482	-0.076	0.458	0.384	1	-0.379
Col6	-0.098	0.108	0.033	-0.134	0.055	-0.379	1

Figure 15-Correlation matrix in visual and tabular form.

Analyzing the box plots of the variables (in original scale), we can observe the presence of many outliers in the input variables. By selecting them (highlighted) we can see that they do not correspond to particular or extreme values of the target variables. It can be observed by selecting one point at a time that an outlier for one column corresponds to an outlier in the others as well, but not in the targets.



Figures 16-Bx plot of the variables of the input dataset.

Continuing our investigation to understand if they refer to erroneous detections on the target plot, we observe that the corresponding target points and are concentrated entirely in the right end of the graph, it could be inferred that the extreme points are concentrated in that area.

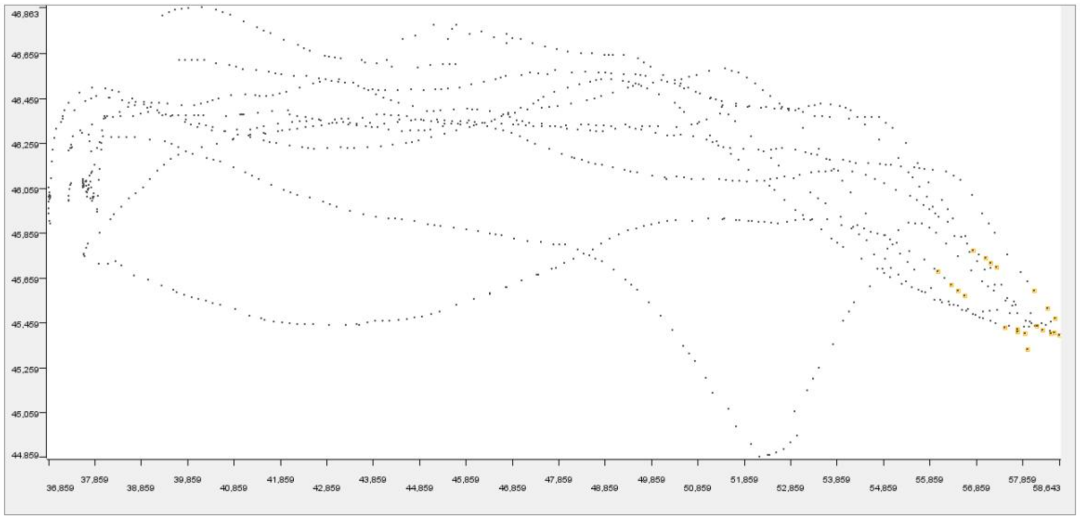


Figure 17_ Graph of x and y targets. Highlighted outliers found.

By performing a filter on the selected points (16 points), a practically unchanged objective function is obtained.

To see if this operation has positive effects on learning or not, the test was done by learning on the filtered dataset and testing on the original one. The result confirmed that not knowing how to behave, the network "shoots" the points in positions even very far from where they should be, a fact that can also be observed by an error, in this case, much higher. The effects of this preprocessing in this case are therefore harmful, by restricting the learning domain we are no longer able to predict correctly for the extreme points in the test phase, these points contain useful information for generalization purposes.

Patterns are observed in the data from the scatter plot of the target variables x and y (Figure 17). The first objective is to find a model capable of representing “tightening” these patterns, or part of them as the presence of noise is known. A first test is to verify by training various models starting from the available data if you can get a good approximation of the training set. If you can find a model capable of representing the data with a good approximation, you can hope to get a good generalization as well.

For this phase I use two external tools offered by knime that implement an MLP with Rprop and MLP with backprop, and I compare the results with those obtained from my network.

For the numerical stability of the developed network and for the use of tools one it is necessary to perform a normalization between [0,1], the tests with the z-score cause numerical overflows on my network.

The two tools were performed with various configurations, mainly the behavior was analyzed when the number of hidden units and maximum learning time changed, which as far as the Rprop algorithm are the two main parameters, while for the backprop after various tests they are alpha and eta 0.3 and 0.1 were used which resulted in a configuration with which to try various dimensions of the hidden layer, in order to search for a better number of units for the generalization. In the tests the search range for the number of iterations went up to 50,000 epochs and for hidden units up to 100 in the rprop case and 200 in the backprop case.

The result of each test was analyzed

- Graphically: comparing the graphs obtained with the two predicted targets and with only one at a time.
- Analytically: verifying the correlation between the predicted target and the given one; with the measure of the error clerk.

Choice of the hidden units by fixing alpha and age, observing the variation of the error as the number of units varies, and we take the one that gives a minor error. Having decided the number of hidden units with the grid process, I look for the best values for alpha eta and gamma. I train the selected model on all data and apply it to the blind test set.

Results of the implemented network and tools

Let's start with a comparison of the effect of the learning parameters on the error curve on a training of 300 epochs, the data have been normalized, the values are only indicative for the comparison. The small number of epochs was chosen only to have a qualitative view of the algorithm's behavior.

From these comparisons it can be observed that as the eta parameter decreases, a smoother and slower variation in the error curve corresponds, while as it increases the variation becomes sharper and more discontinuous, this in accordance with the theory, as more or less importance is given while learning the new pattern seen compared to the previous ones. As will then occur with the model selection, the alpha and eta parameters that obtain the best performance on this problem are alpha of about 0.5 and eta with values between 0.01 and 0.1.

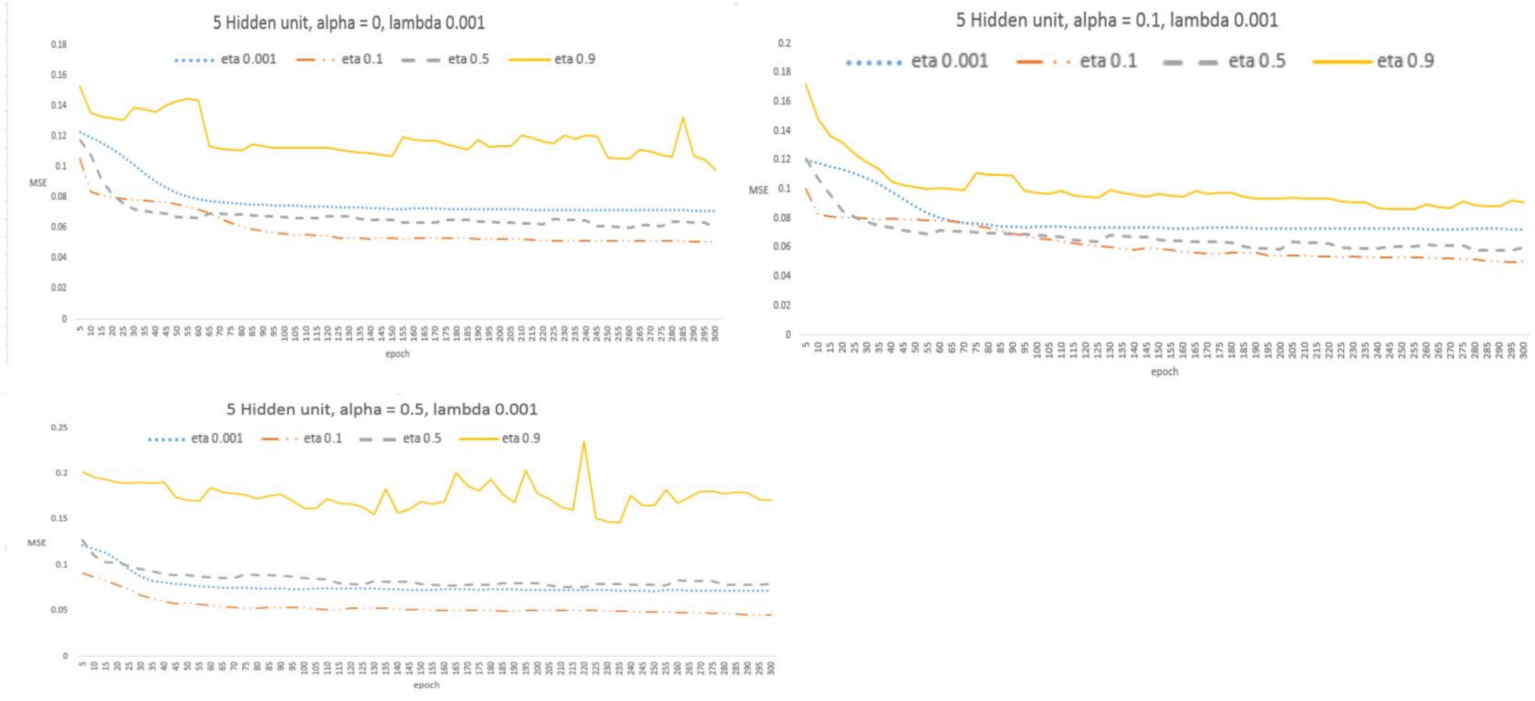


Figure 18_Graphs during the training of the validation error as the alpha and age hyperparameters vary.

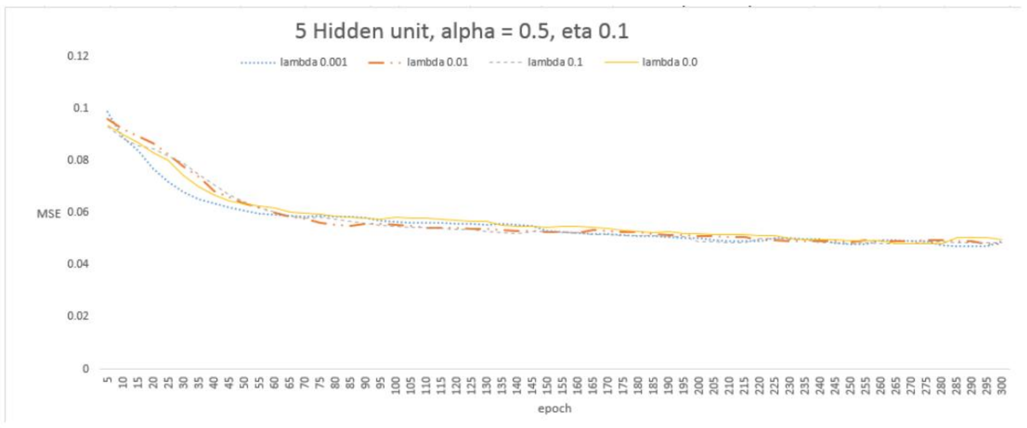


Figure 19_Graph as lambda varies.

This graph instead compares how it varies based on the lambda regularization parameter. Observing that a small lambda makes the greatest difference by making the curve smoother and smoother, and above all it would seem to increase the speed with which it reaches the "minimum".

The best result was obtained with the MLP Rprop (knime), with 25 hidden units and 50,000 was not very encouraging. Plotting the target axes, and the learned axes, using the training set as a test set, we observe that the network is unable to represent the training data. The main difficulty lies in the target variable y (col 6). From the graphical comparison between y target and y calculated it can be seen that the precision is low, there is no precise but sparse diagonal, the correlation index between the various configurations has reached a peak of 0.80 with this. Using the x target and the calculated y, an approximation of the overall form is also achieved, but without too much detail.

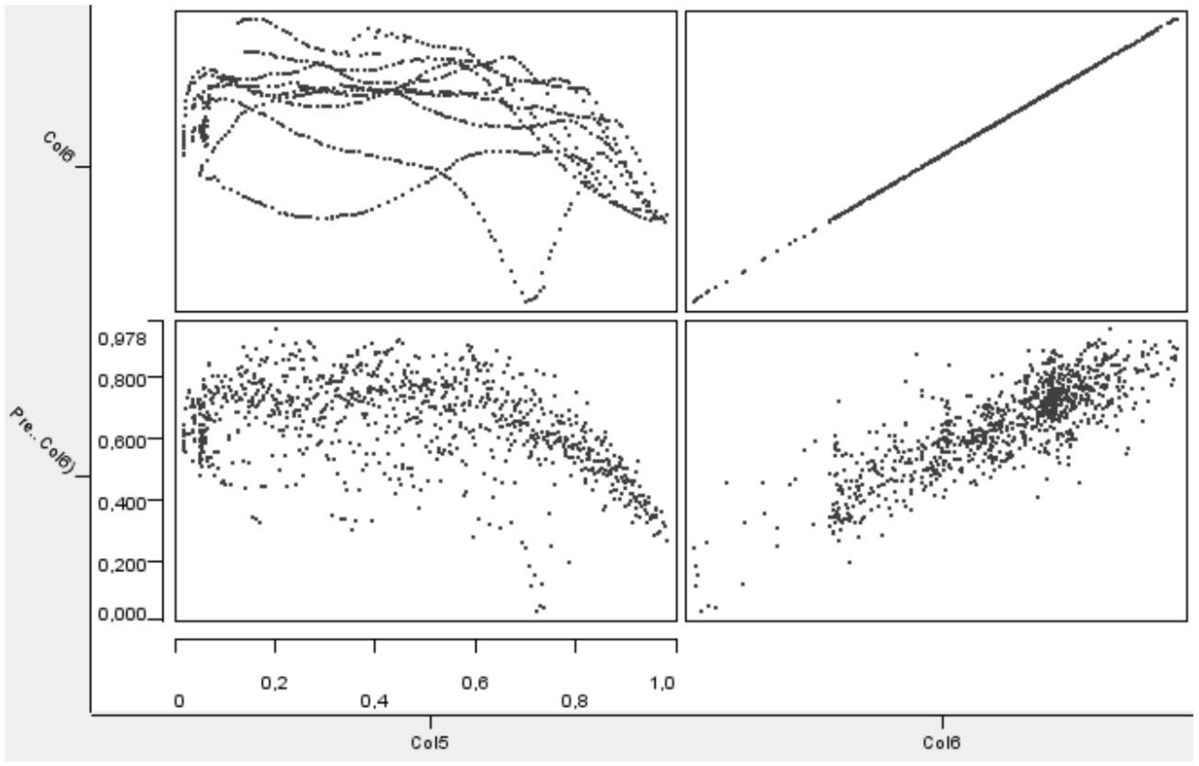


Figure 20_Graph comparing the targets and the calculated variables, col5 = X and col6 = Y.

On the variable x (col5) the results are better, the correlation index is 0.98, visible from the more compact diagonal in the graph. Another confirmation of a better approximation of x is obtained by plotting the predicted x with the y target, and obviously from the measures of the error relative to x.

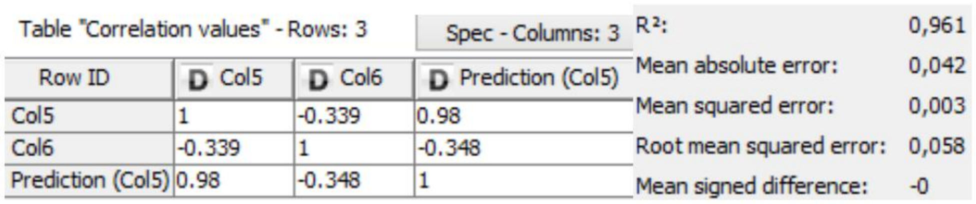


Table "Correlation values" - Rows: 3 | Spec - Columns: 3

Row ID	D Col5	D Col6	D Prediction (Col5)
Col5	1	-0.339	0.98
Col6	-0.339	1	-0.348
Prediction (Col5)	0.98	-0.348	1

R²:	0,961
Mean absolute error:	0,042
Mean squared error:	0,003
Root mean squared error:	0,058
Mean signed difference:	-0

Figure 21_Table of correlation and errors calculated on the approximation of X only.

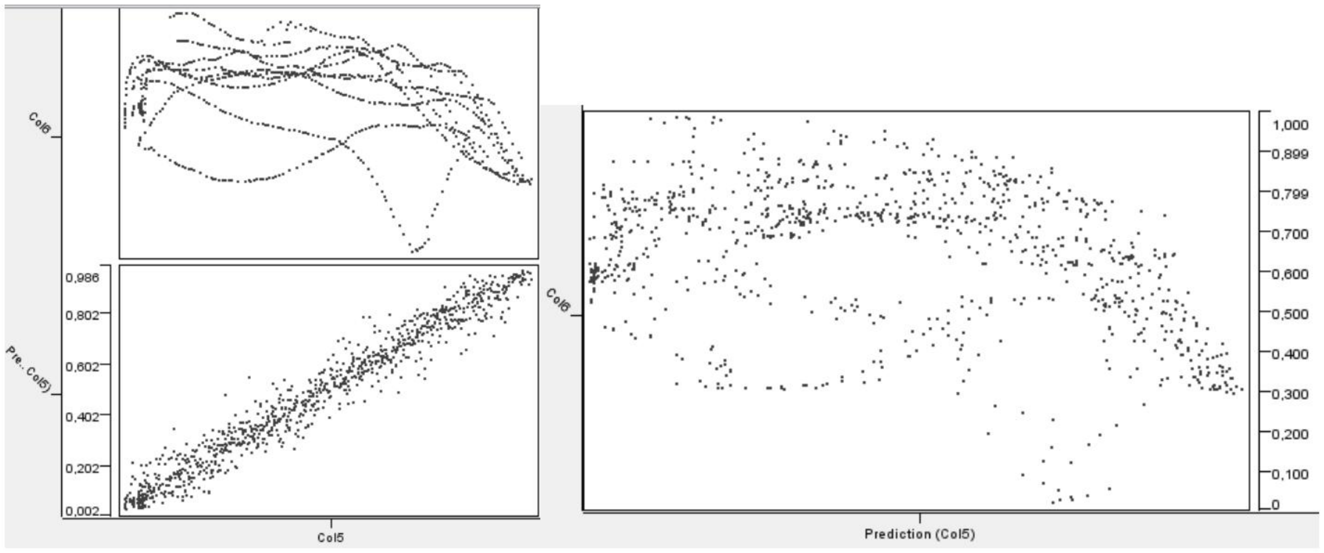


Figure 22_Graphs relating to the comparison between X target (col5) and calculated.

Without degrading the accuracy of the result too much, Rprop on y with 14 hidden units and 50,000 iterations achieves very similar results, using a much simpler model.

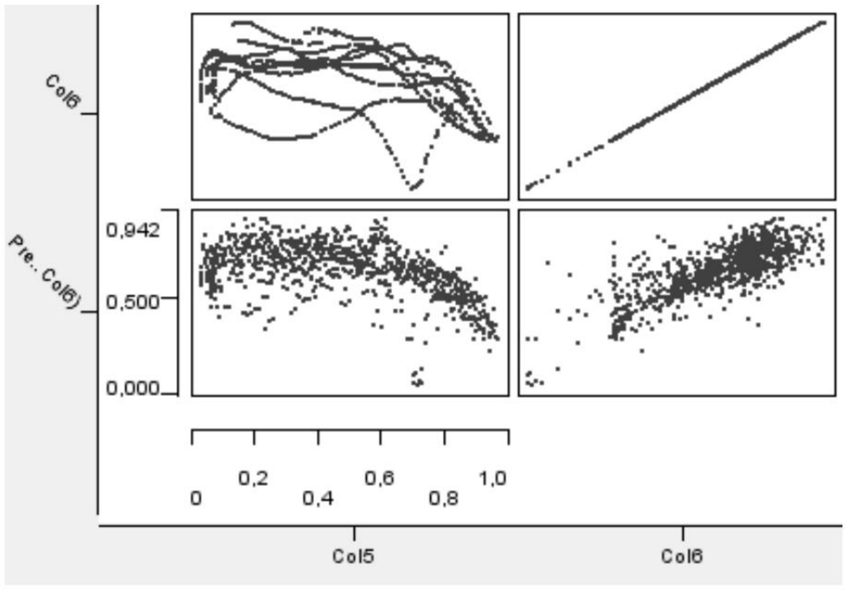
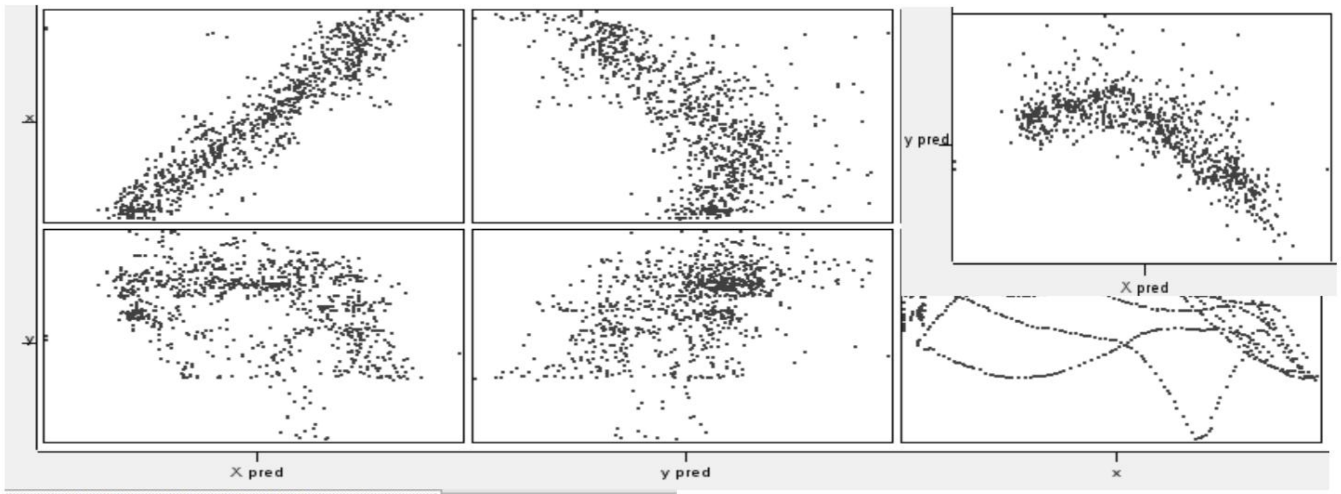


Figure 23_Graphs of comparison between Y (col6) and Y calculated.

With these results, I don't expect to get a very low error on this dataset. But I was able to identify a range of values on which to search to select the final model.

From the tests carried out with my implementation, following the model selection procedure described above, with 25 units I get similar results but worse than the tool analyzed before. The difficulty in approximating the variable Y is confirmed, the correlation of which, as seen in the comparison graphs, is low (0.5). Better on X, keeping the original y manages to see an approximation of the original figure.



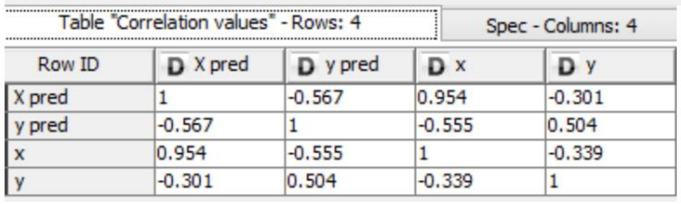


Table "Correlation values" - Rows: 4 Spec - Columns: 4

Row ID	X pred	y pred	x	y
X pred	1	-0.567	0.954	-0.301
y pred	-0.567	1	-0.555	0.504
x	0.954	-0.555	1	-0.339
y	-0.301	0.504	-0.339	1

Figure 24_Graph and correlation matrix for comparison between target variables and those calculated by my network with 25 hidden units.

Putting together x and y predicted by the model we obtain a simplified representation of the learned function, evident the loss of the lower forms of the figure.

Comparing the results just obtained with a model with 25 hidden units, where the model selection did not fall below an error (mse) of 0.158, 0.161 on the normalized data, I can see that the performances are not significantly degraded. With Eta parameters 0.01 alpha 0.5 lambda 0.01 the learning curve is fairly regular without particular behaviors, it shows an asymptotic convergence, which however does not fall below a certain threshold.

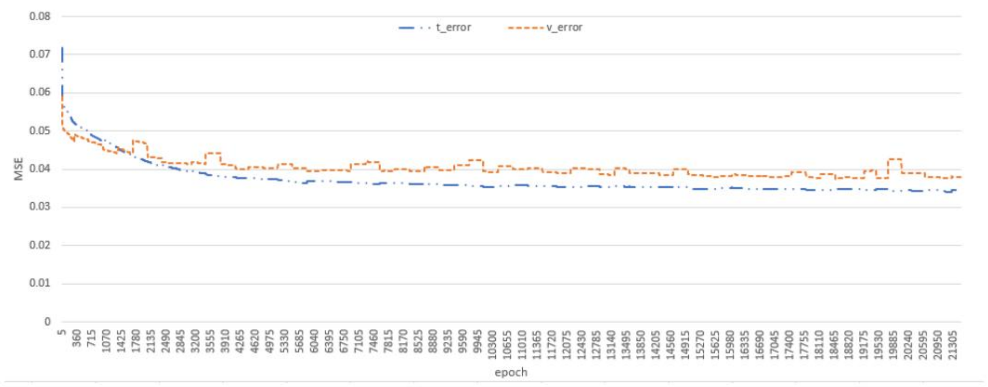


Figure 25_Learning curve with mse on training and validation set.

Training set test (mse) 0.15. The poor prediction on the y axis continues, while the prediction on the x axis is slightly better. This increase, however, is not worth the increased complexity of the model, which risks worsening its ability to generalize. In fact, from the estimate with hold out there is no appreciable improvement.

Further tests and research based on holdout showed the possibility of further reducing the number of units down to 5 units. To consider that a more accurate estimate of the error would be obtained with a cross validation, but due to the computational costs it has not been done.

Hidden units	Validation MSE media su 3 trial Media epoche	Emin
5	5.85	15k
10	7.42	8k
20	8	12k
30	7.6	16k
40	6.7	7k

Figure 26_Testing table with several hidden units.

Rerun the search with a finer grain between 5 units and 10 units, yielding 5 units with a training set error of 27.

At this point, the eta, alpha and lambda parameters are tuned, as described with a grid search,
obtaining: eta = 0.1, alpha = 0.1, lambda = 0.001; The three trials relating to the best parameters obtained on the normalized data a loss of 5.4 in 10k epochs, 5.4 in 30k epochs, 7 in 20k epochs respectively, with an average of 5.5 on the validation set. While the graph shows the training of the final model used to produce the output for the competition, with 11.5k epochs and a minimum reached after 6.5k epochs.

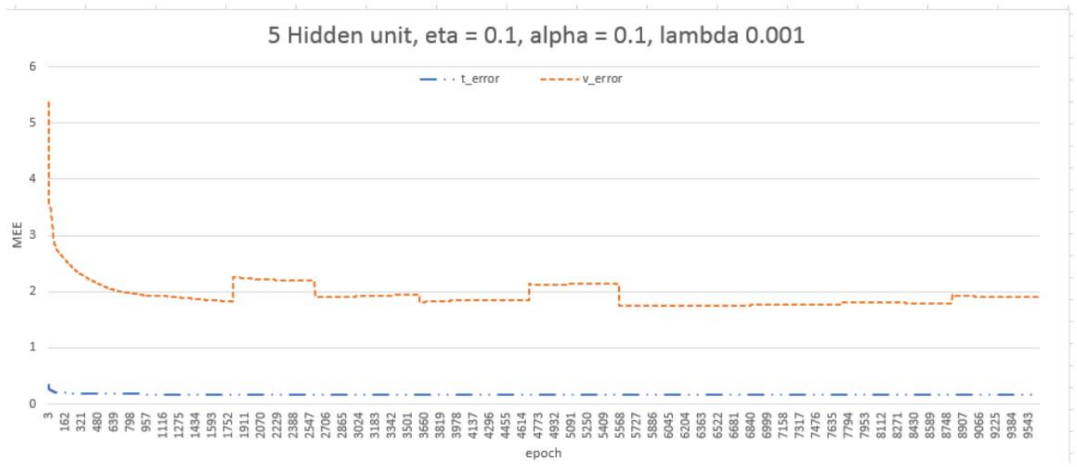


Figure 27_Training progress of the final model used for the test.

The error stabilizes at a certain value without going back thanks to the normalization due to the weight decay.

The estimation of the prediction error on the final model is made by cross validation with 5 fold, resulting in an average loss mee of 1.9.

Conclusions

In conclusion, the graph relating to the prediction on the test set must be shown.

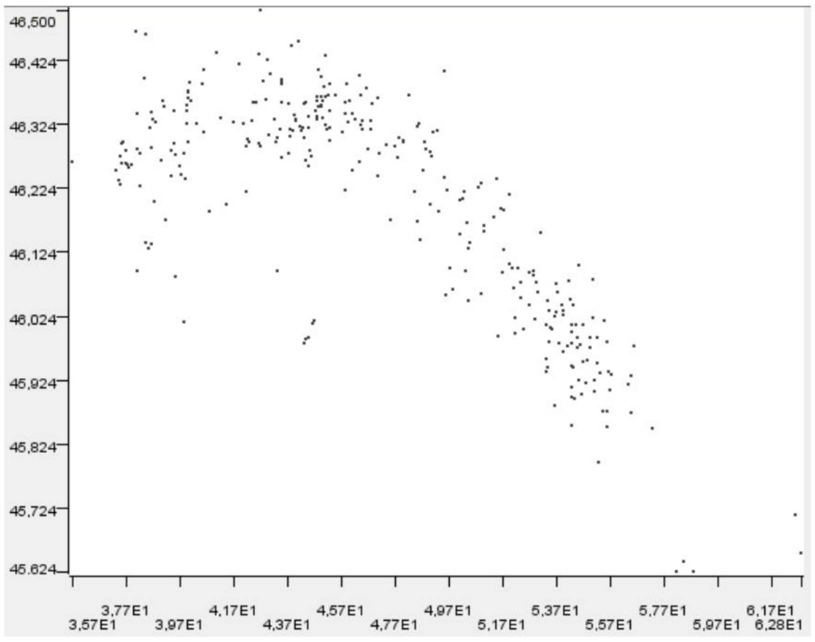


Figure 28_Graph of the prediction on the test set.

For the project I chose the C ++ language both to obtain good computational performance and to learn a new language.

The implemented network did not reach a very high prediction accuracy, compared with the other models, to solve the problem with greater precision it is necessary to continue the investigation on other models, or with greater computing power try to find a better model with a broader search.

Attached FBAA1_abstract.txt, FBAA1_LOC-UNIPI-TS.csv and the folder FBAA1_Project2013.rar with the sources.

I authorize the publication of the results.

Bibliography & Link

[1] Haykin

[2] Mitchell

[3] Early Stopping | but when? Lutz Prechelt (http://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf)

[4] Optimization methods for neural networks L. Grippo (http://www.disp.uniroma2.it/users/piccialli/retineurali.pdf)

[5] ftp://ftp.sas.com/pub/neural/FAQ.html

[6] https://archive.ics.uci.edu/ml/datasets/MONK's+Problems

[7] http://scott.fortmann-roe.com/docs/MeasuringError.html#fnref:1