Computational Intelligence

Project Report

Master's degree in:

Embedded Computing Systems

Academic year 2015/2016

Submitted by:

Elena Lucherini

Sara Falleni

Scuola Superiore Sant'Anna, January 10, 2016.

Contents

1 Introduction

Let us consider a working environment in which there are a set of tasks that might be assigned to different people. Each task may expose the worker that performs the task to a set of risks. With this project, we would like to provide the decision maker, who is in charge of task assignment, with information related to people's risk perception. Risk perception is the way a person perceives and evaluates the characteristics of dangerous situations.

Criticality factors are what influences risk perception. We can distinguish between *general criticality factors* (personal and psychological factors, related to the worker independently of any past or future task) and *task-related criticality factors*.

1.1 Approach

The project is divided into three parts:

- The development of an MLP neural network to compute each worker's general level of risk perception, and three MLP neural networks (as many as there are risks associated with a given task t_j) to compute the caution level of worker w_j with respect to the associated risk.
- The development of two classifiers which, evaluating different representations of the same pattern (that is, a worker), makes coherent decisions, as it will be explained in the following pages.
- The development of a population of chromosomes containing a set of classifiers with different implementation, and a set of functions to manage the population. A genetic algorithm is applied to the population, in order to minimize the error given by the classifiers, to obtain the best configuration.

2 Part I: MultiLayered Perceptron Neural Networks

Our first step was to implement the General Level of Risk Perception network and the three Risk Caution networks. For each MLP network, we used the same procedure, explained as follows.
The first function used, **create_mlp**, creates and trains a pool of networks with a single hidden layer having an amount of neurons between 2 and 10 (9 configurations). For each configuration of hidden neurons, the network is trained 9 times, and each resulting network is saved into a structure (a total of 81 networks is saved). In addition to this, the performances (mean errors) of each network are stored. A graph of the best performance for each configuration of hidden neurons is displayed, for a total of 9 points on the graph. 70% of the given input data is used as training set, another 15% forms the validation set, while the remaining 15% is the test set.
The **choose_mlp** function is then used on the 3-4 neural networks with the best performance, to obtain their histogram and regression graphs. This way, we can choose the appropriate trade-off among the amount of hidden neurons, the performance of the network, and the results of the histogram and regression graphs.

2.1 General Level of Risk Perception (GP)

Given a worker w_j, the network computes the general level of perception for the risk by w_j. The input is a 70x4 matrix of factors (from F1 to F4), while the target output is the 70x1 vector gen_perc.

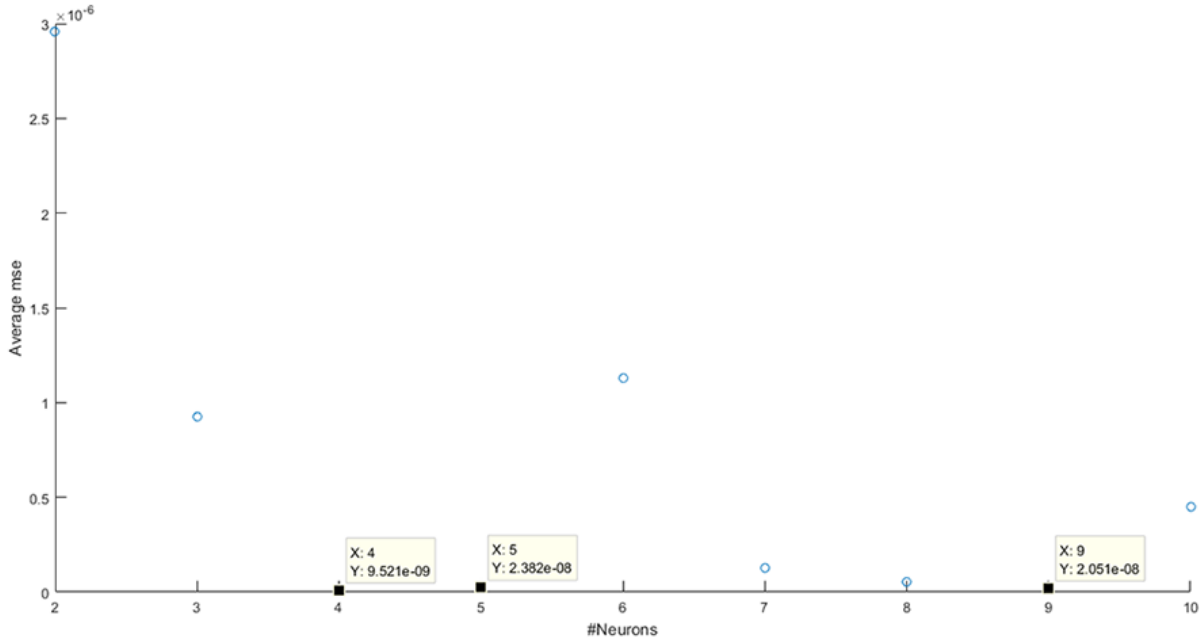


Figure 2.1: Performance of GP, given by the execution of *create_mlp*

The figure shows that the lowest errors is obtained with the networks with four, five and nine neurons in the hidden layer. At this point, we compared the histogram and regression graphs, searching for the best trade off:

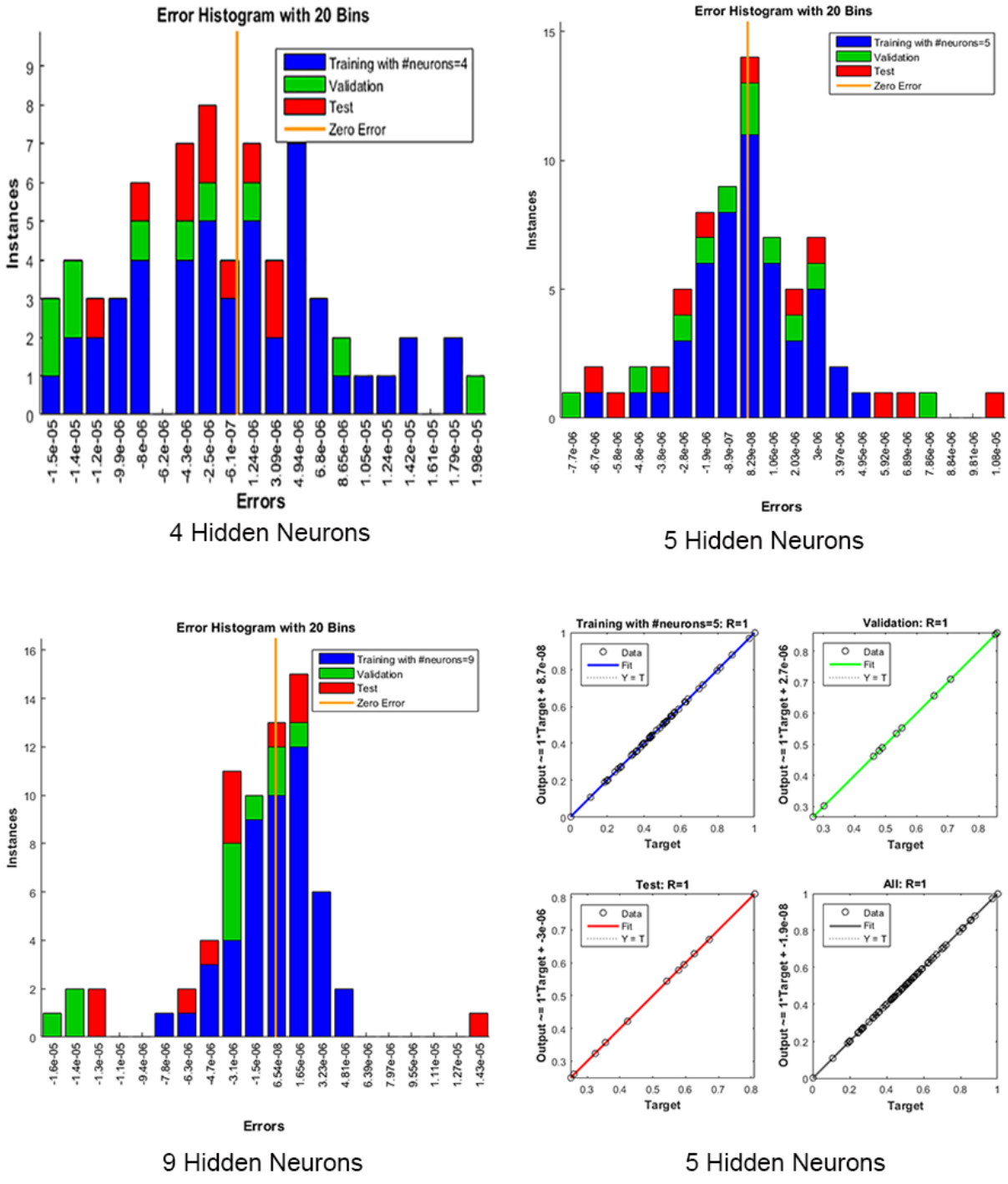


Figure 2.2: GP Histograms and Regression

The histogram of the five-hidden-neuron network is less scattered and it is the most similar to a Gaussian bell. Besides, the Error values are smaller, especially for Test and Validation, which are the most significant. Its results in the regression are perfect, so we picked the network with **five neurons** in the hidden layer.

2.2 Risk Caution 1 (R1)

To generate the input matrices of the remaining MLP networks, we wrote a function called **risk_counting**, which counts the number of actions associated to a given risk for each worker w_j. The function, thus, creates a 70x3 matrix, where the *j-th* row identifies worker w_j and, in each column, the amount of actions associated to a given level of risk - column 1 for **low**, colum 2 for **medium** and column 3 for **high**.
For the input matrix of network R1, we used *risk_counting* on tasks from A11 to A18, only referring to Risk 1. The desired outputs are given by *risk_caution (R1)*. Finally, we used *create_mlp* and *choose_mlp*, as previously explained.

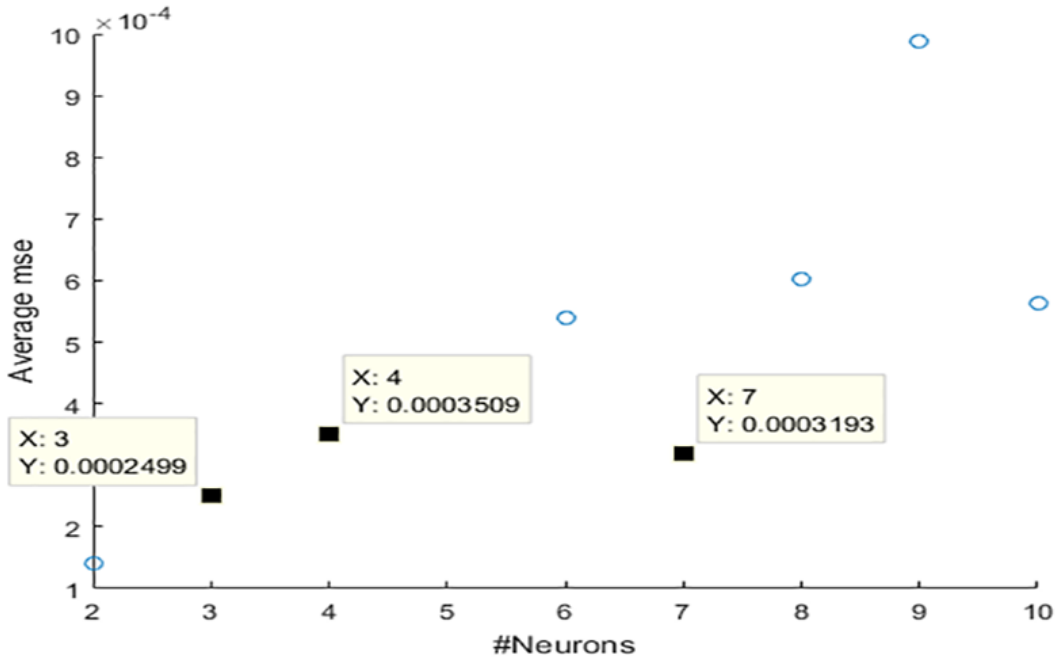
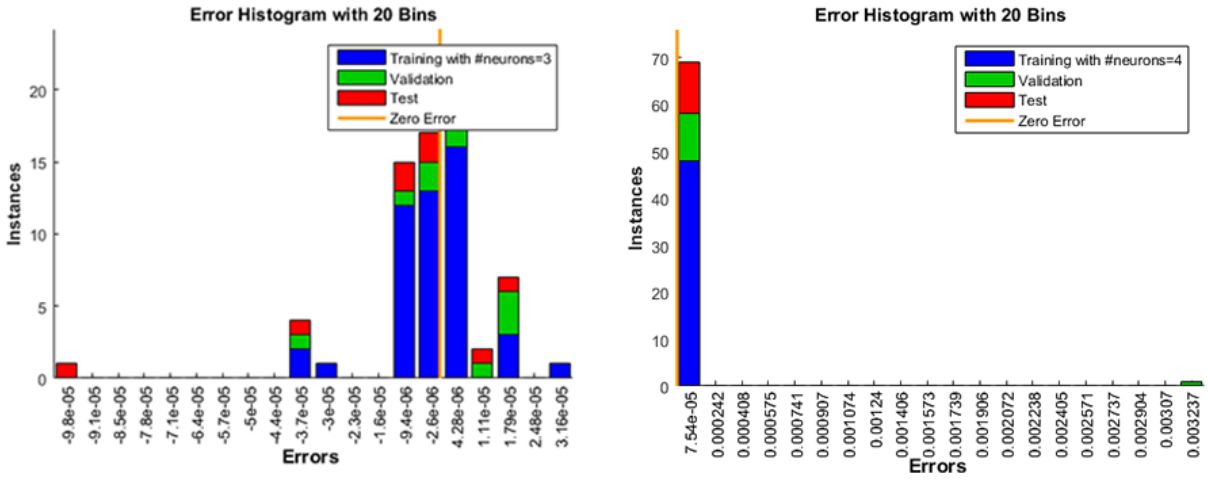
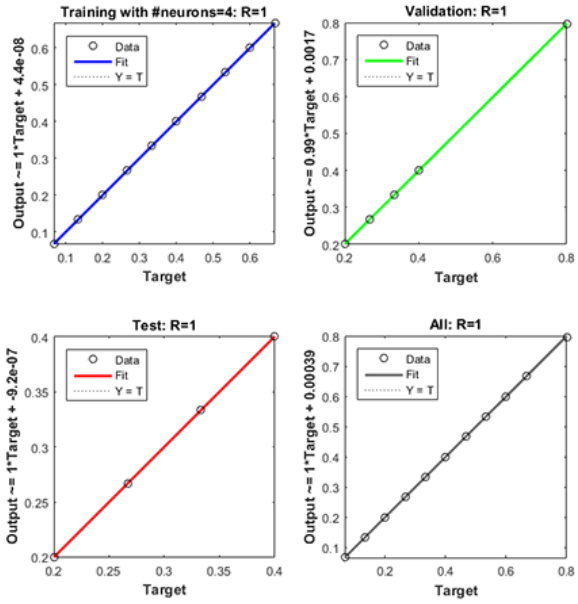


Figure 2.3: R1 Error Means

The networks with two neurons in the hidden layer has the best performance, but it is, in our opinion, too small, since it is smaller than the number of inputs given to the networks (3). Thus, we consider the networks with 3, 4 and 7 hidden units. The network with 7 hidden units is discarded, since it appears to be too large a size and the regression graphs are not perfect, as opposed to the other configurations. The error for the 4-unit configuration is closer to zero than the the 3-unit one, but since the regression graphs are exactly the same and the **3-unit hidden layer network** is smaller, we picked the latter.



3 Hidden Neurons 4 Hidden Neurons



4 Hidden Neurons

Figure 2.4: R1 Histograms and Regression

2.3 Risk Caution 2 (R2)

With the same procedure as for R1, we generated the input data for R2, using *risk_counting* on columns A21 through A27. Then, as we did in the previous section, we created a pool of MLP networks with different configurations of hidden neurons and chose the best one.

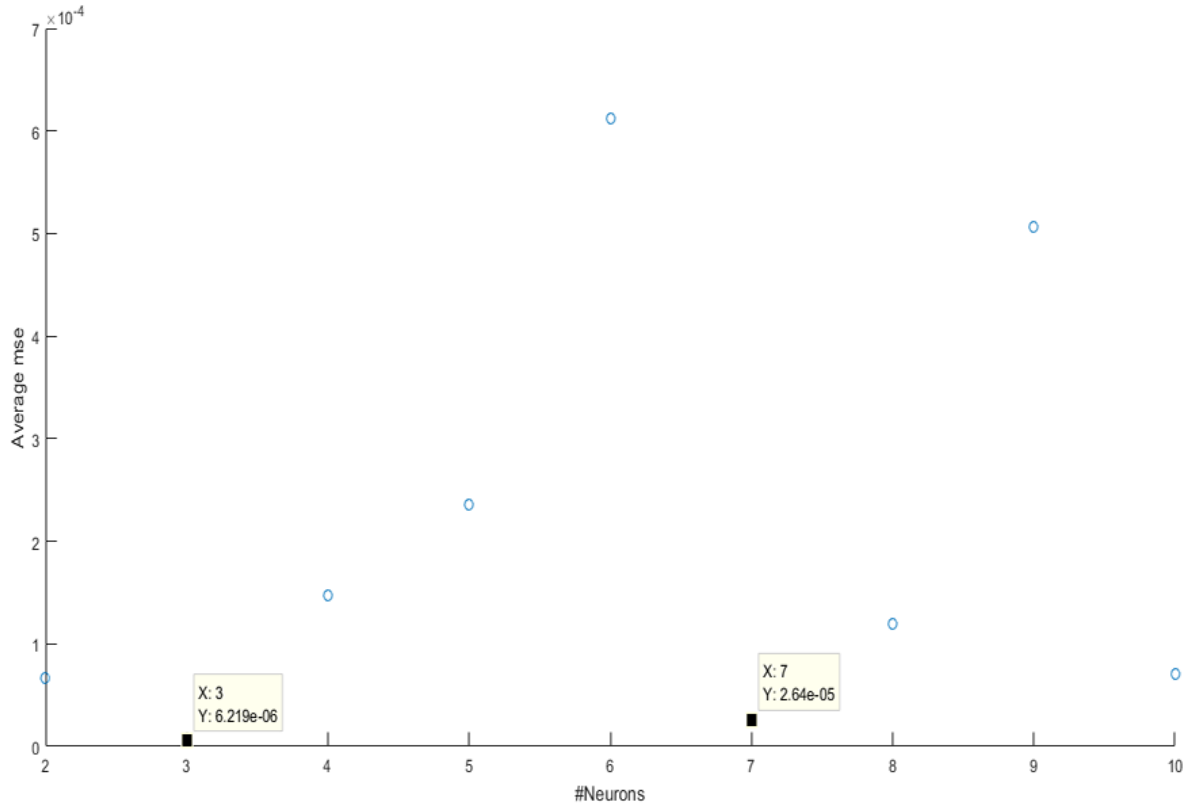


Figure 2.5: R2 Error Means

The best results are given by the networks with three and seven neurons in the hidden layer.

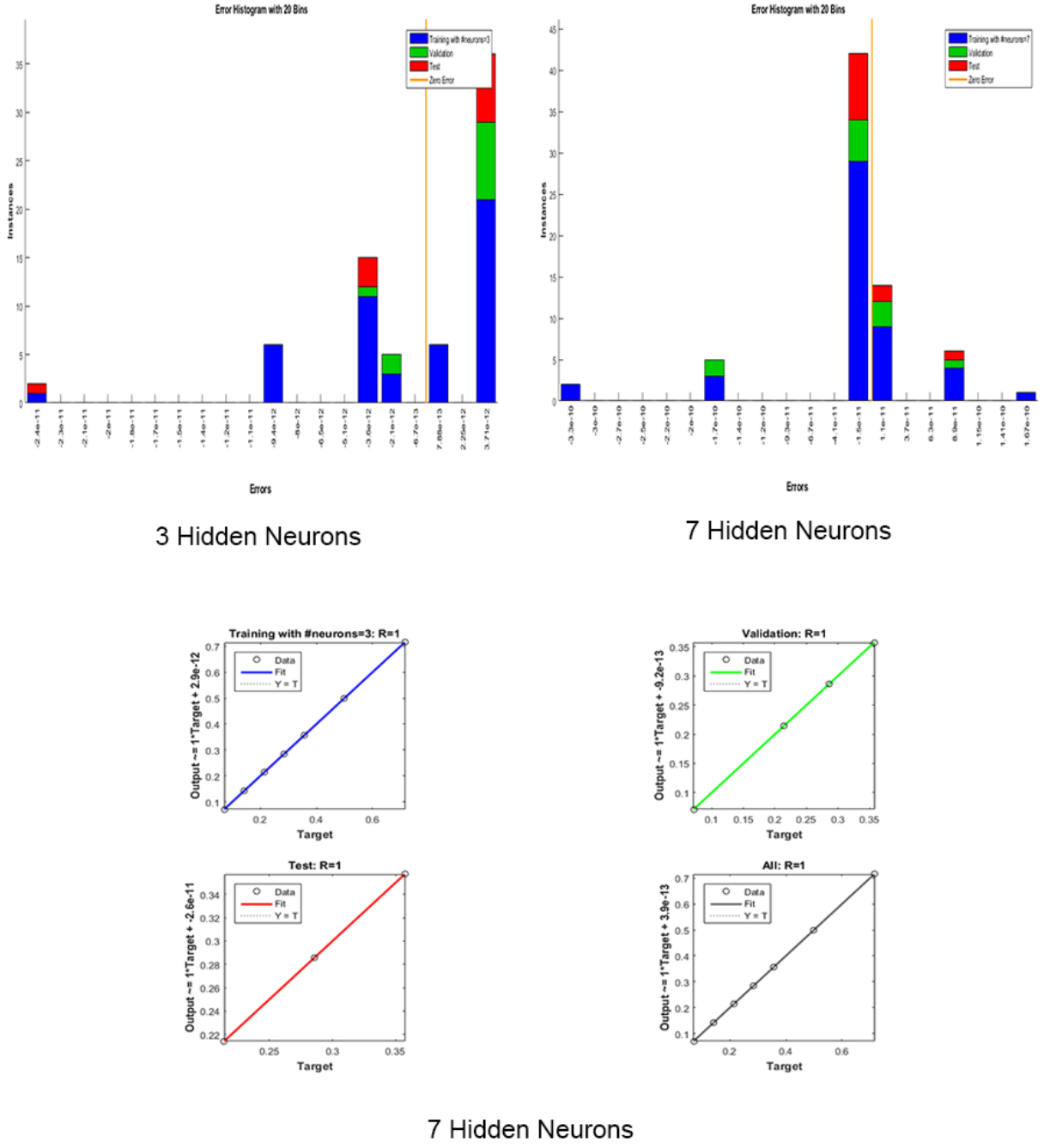


Figure 2.6: R2 Histograms and Regression

We eventually picked the three-hidden-neuron network, for the same reasons as R1.

2.4 Risk Caution 3 (R3)

The same procedure used for R1 and R2 has been used for R3. The inputs come from *risk_counting* applied to columns A31 through A37. The Performances are the following:

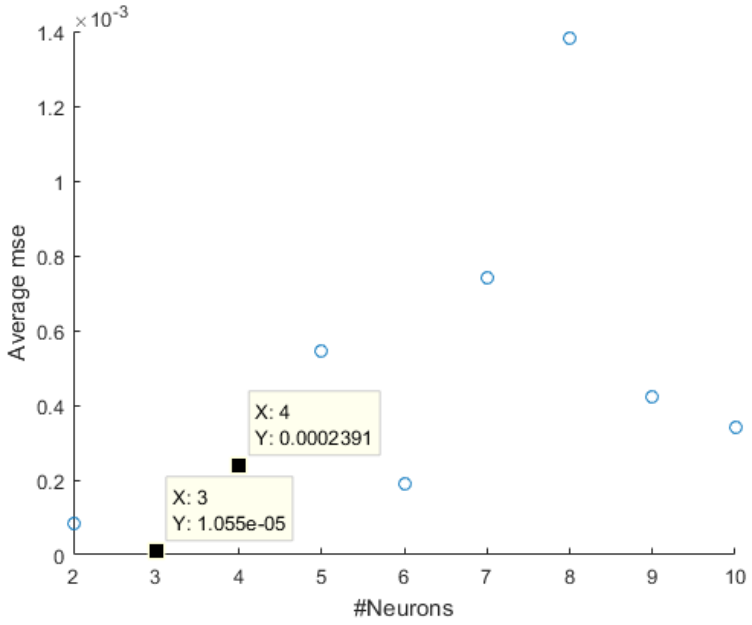


Figure 2.7: R3 Error Means

The best results are obtained with the 2-, 3- and 4-unit configurations. Once again, we deemed the two-hidden-neuron network too small for our purpose.
Furthermore, there are two orders of magnitude between the error of the three- and four-unit configurations, so we discarded the larger one and picked the network with a hidden layer of **three neurons**.

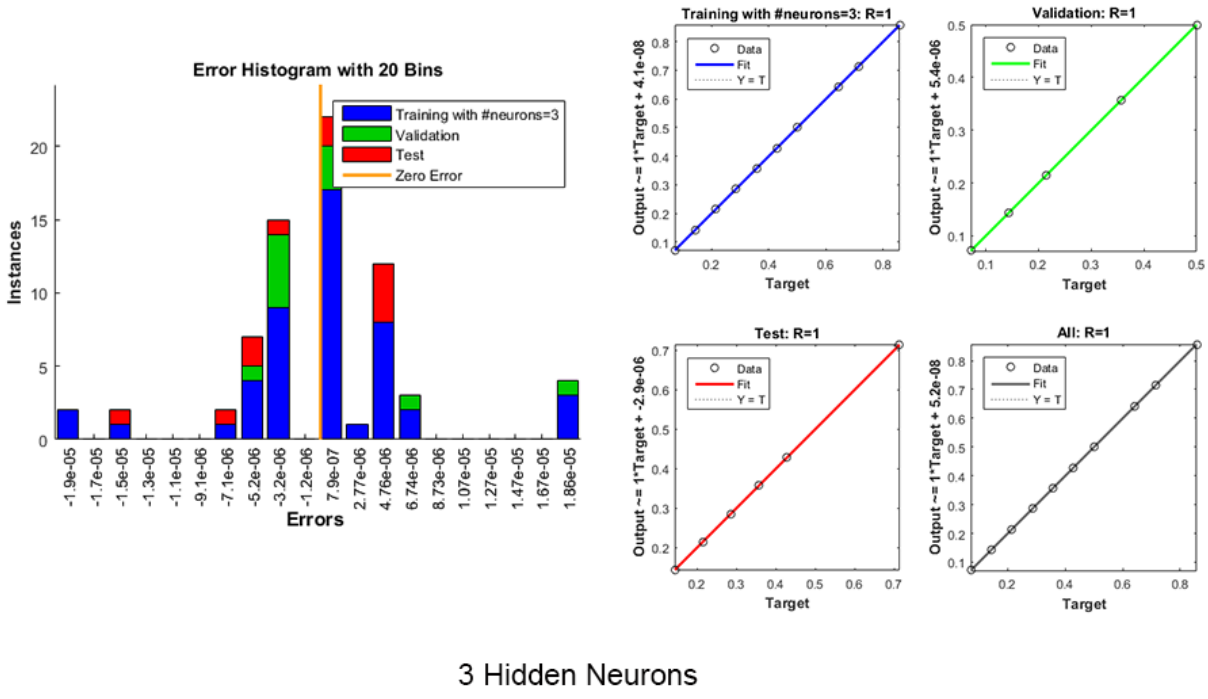


Figure 2.8: R3 Histogram and Regression

3 Part II: Classifiers

3.1 Classifier 1

The second part of the project consists in the development of two Classifiers: each classifier must be implemented as an MLP neural network, a RBF neural network and an ANFIS system. The first step was to create an MLP.

3.1.1 MLP 1

To create the MLP network of the first classifier we used the same technique used to create the previous MLP networks: the input of the network is obtained concatenating *gen_perc* with ***f5*** and ***f6*** (task-related factors in risk perception).The target output is the ***task_perc***. The Performances graph generated by **create_mlp.m** is as follows:

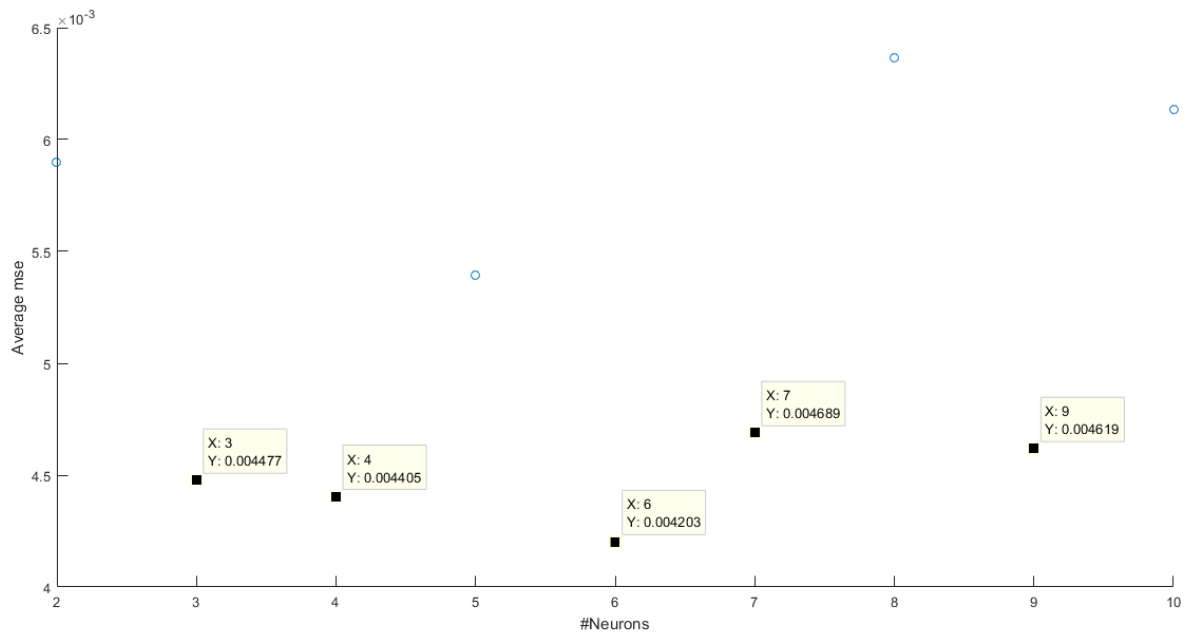
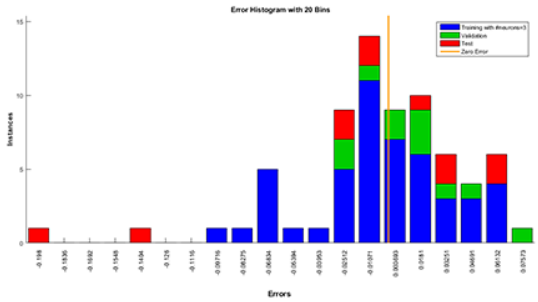
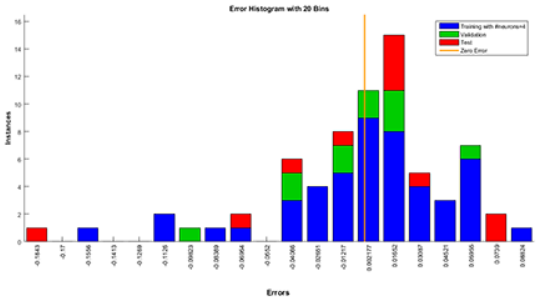


Figure 3.1: CL1 Error Means

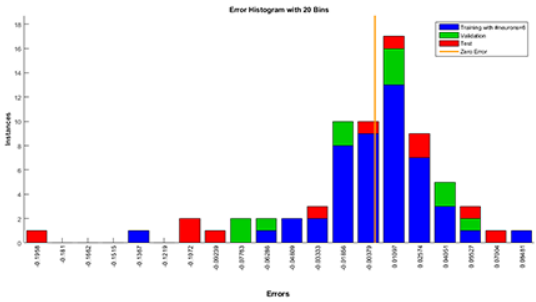
Five networks (3-, 4-, 6-, 7-, and 9-neuron networks) gave very similar results, so we decided to analyze them all. Below are the graphs obtained with *choose_net.m*.



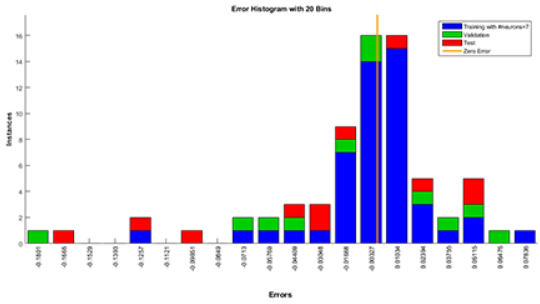
(a) 3 Hidden Neurons



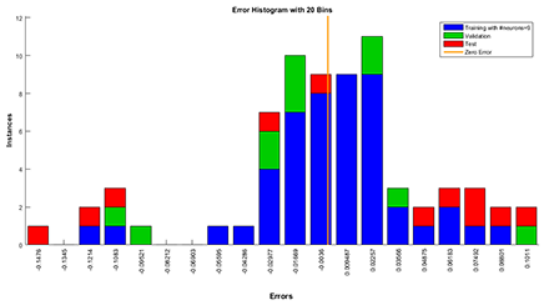
(b) 4 Hidden Neurons



(c) 6 Hidden Neurons

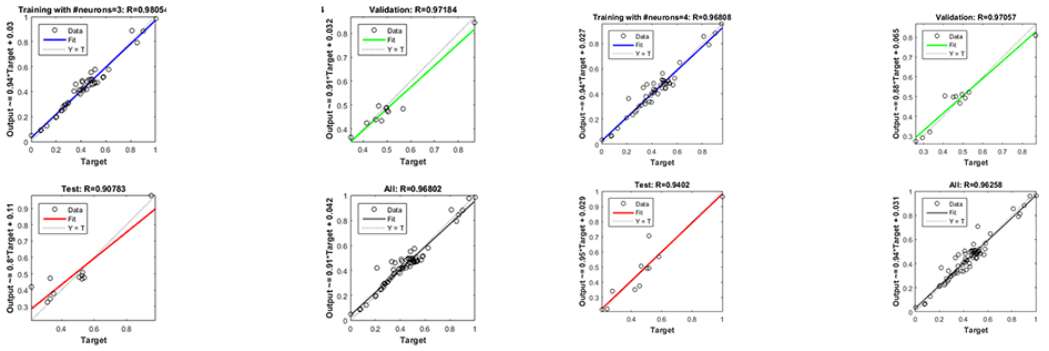


(d) 7 Hidden Neurons



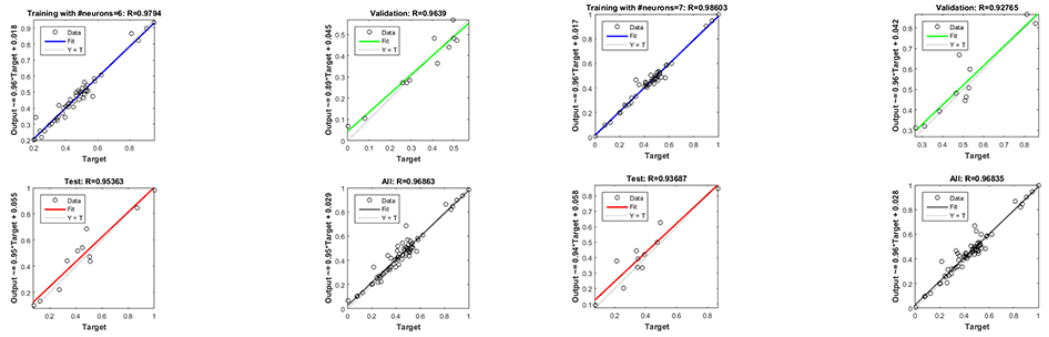
(e) 9 Hidden Neurons

.



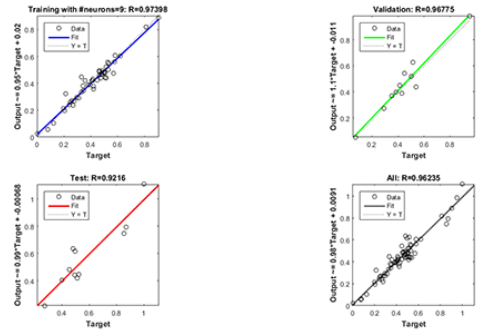
(a) 3 Reg

(b) 4 Reg



(c) 6 Reg

(d) 7 Reg



(e) 9 Reg

As seen in the figures, the histograms turned out to be very similar. We eventually picked the network with **six neurons** in the hidden layer, as the validation and test data are closer to the center of the Gaussian bell in the histogram. The regression graph looks the nicest, as the Validation and Test graphs, the most significant, give the highest accuracy.

3.1.2 RBF 1

In order to create a pool of RBF networks, we created the **create_rbf** function, conceptually similar to the aforementioned *create_mlp*. *create_rbf* creates an RBF network for each amount of neurons in the hidden layer that goes from 3 to 13 (for a total of 11 configurations) and for each spread value going from 0.1 to 0.9 (step equal to 0.1, for a total of 9 spread values - that makes a total of 99 RBF networks).
80% of the input data is used for the training set, while the remaining 20% forms the test set. The sets are randomly generated from the input data for each RBF network, so that each network has different training and test sets.
The function returns the pool of networks, along with the performance (mean error) and the data on the performance resulted from the training. Particularly, the average performance for each configuration of neurons in the hidden layer is displayed in a graph (a total of 11 points are shown in the graph).
A second function, analogous to *choose_mlp*, **choose_rbf**, takes the networks with the best performance and generates its histogram and regression graphs.

The input data and target output for Classifier 1 are the same as the one used previously in the MLP.

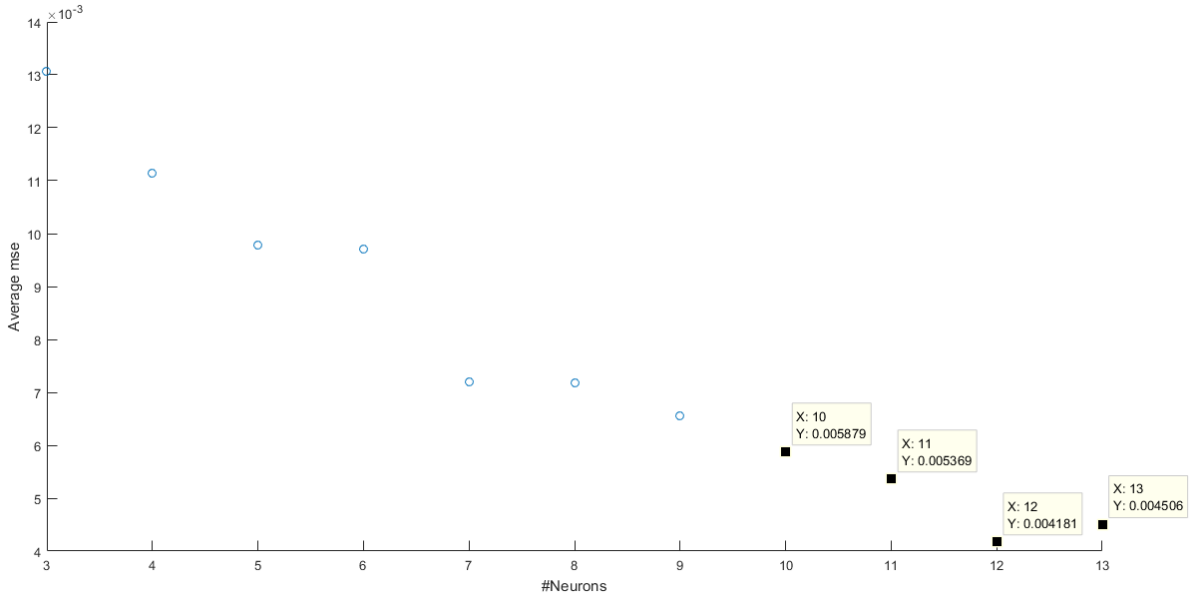
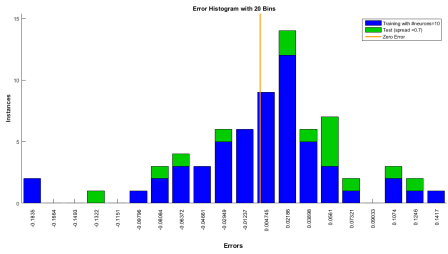
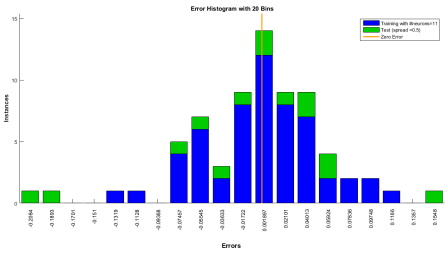


Figure 3.4: RBF1 Error Means

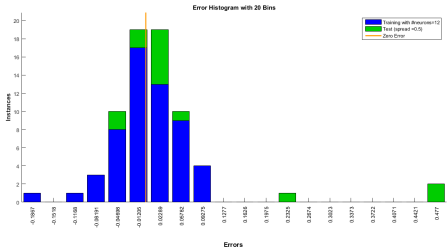
As we can see from the graph, the best configurations of neurons in the hidden layer are 10, 11, 12 and 13. The histogram and regression graphs are the following:



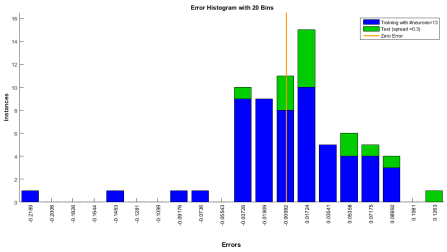
(a) 10 Hidden Neurons



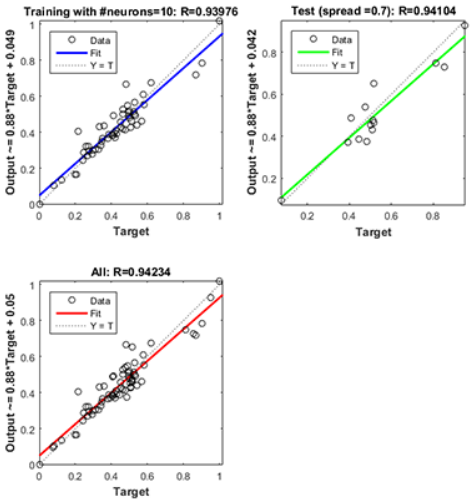
(b) 11 Hidden Neurons



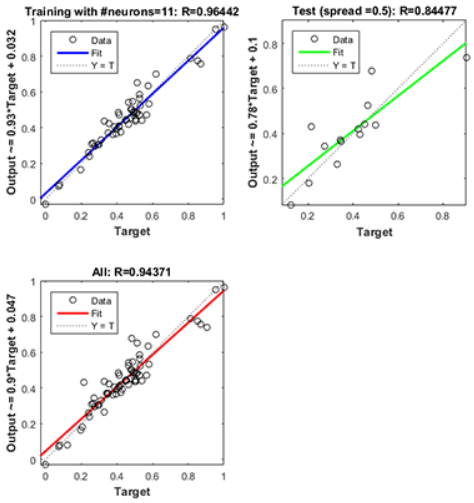
(c) 12 Hidden Neurons



(d) 13 Hidden Neurons

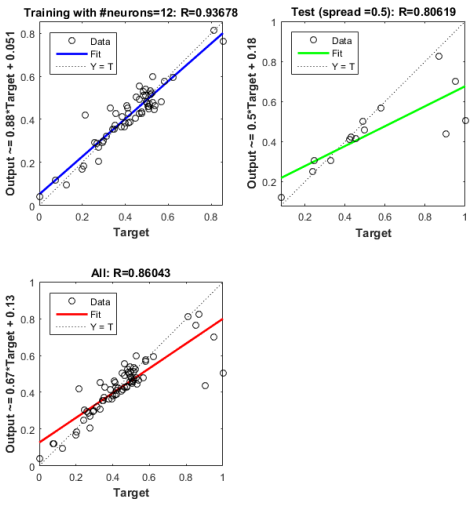


(a) 10 Reg

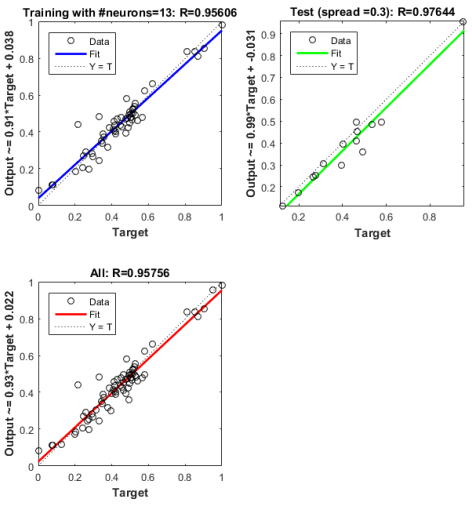


(b) 11 Reg

!ht



(c) 12 Reg



(d) 13 Reg

In the end, the chosen RBF network is the one with **12 hidden neurons** and **spread** equal to 0.5. The criteria followed are the ones explained before.

3.1.3 ANFIS 1

As for the MLP and RBF networks, two functions have been created to generate a pool of ANFIS systems and pick the best one.
The first function is **create_anfis**, which creates ANFIS systems with a number of membership function between 3 and 6 (4 configurations) and trains each of them ten times, saving each resulting system, with a different training set randomly generated from 80% of the input data (the remaining 20% is used as the test set). A total of 40 ANFIS systems is returned by the function, along with the graph of the minimum error given by each configuration of membership functions (a total of 4 points are represented in the graph).

The second function, **choose_anfis**, is applied to the systems with the lowest error and displays the regression and histogram graphs thanks to which the best system is chosen.

The input data and target outputs are the same as the ones given to the neural networks described before.

As shown in the figure below, the systems with four and five membership functions obtained the lowest errors.

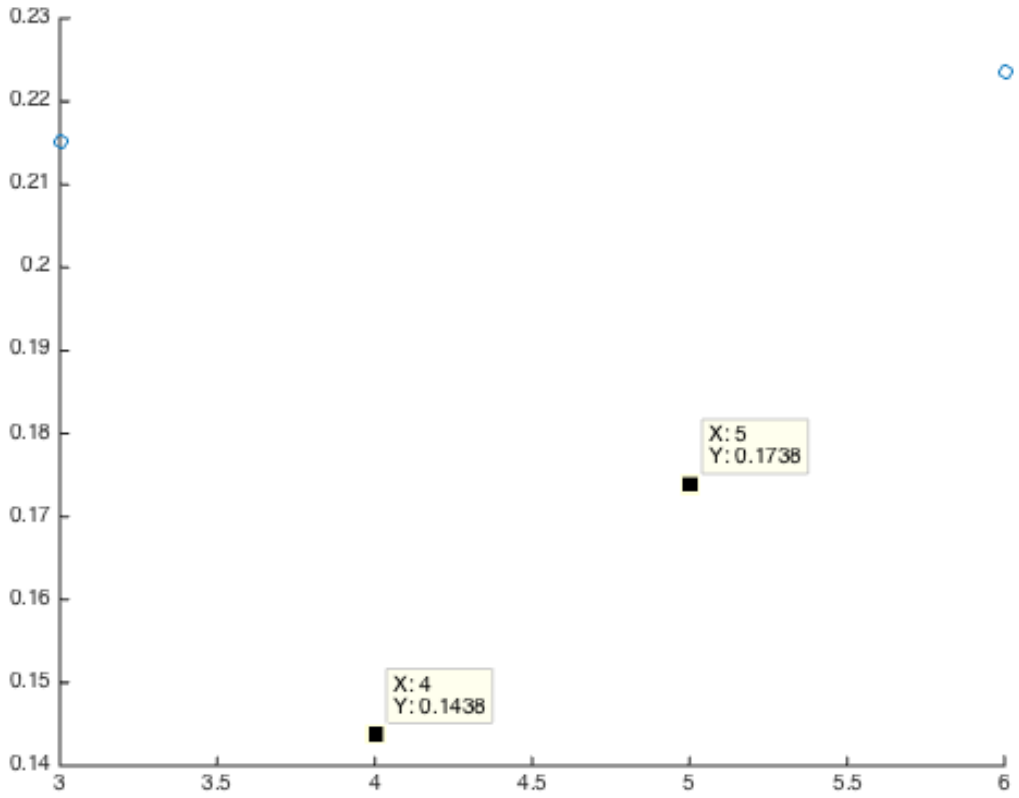
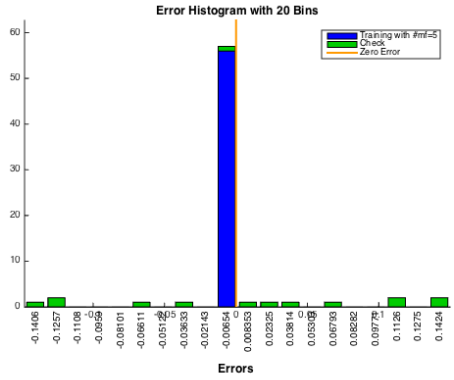


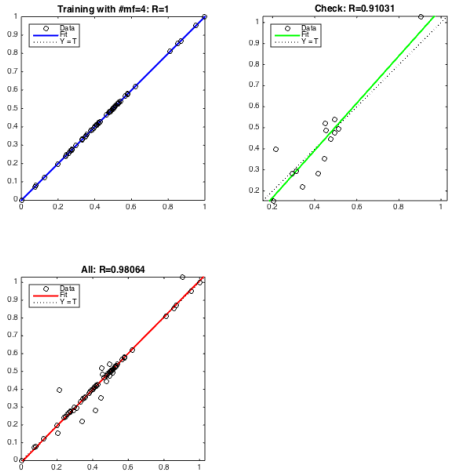
Figure 3.7: ANFIS1 Error Means

The regression and histogram graphs are the following.

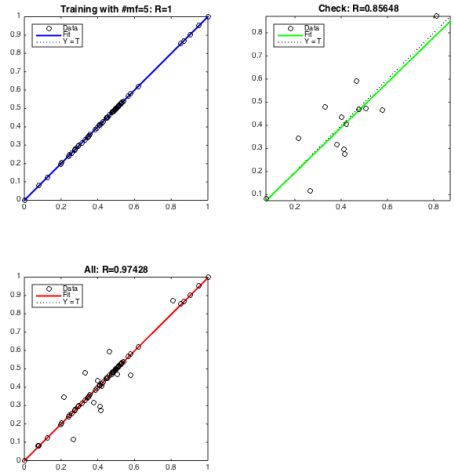
(a) 4 err



(b) 5 err



(a) 4 reg



(b) 5 reg

The chosen ANFIS has **five membership functions**.

3.2 Classifier 2

3.2.1 MLP 2

As previously described, we used the **create_mlp** using 70x3 matrix ***risk_caution*** as input, and 70x1 vector ***task_caution*** as target output. **choose_mlp.m** has then been used to pick the best network in the pool.

The first function returned the following performances:
Below are the Regression and Histogram graphs plotted by **choose_mlp**:

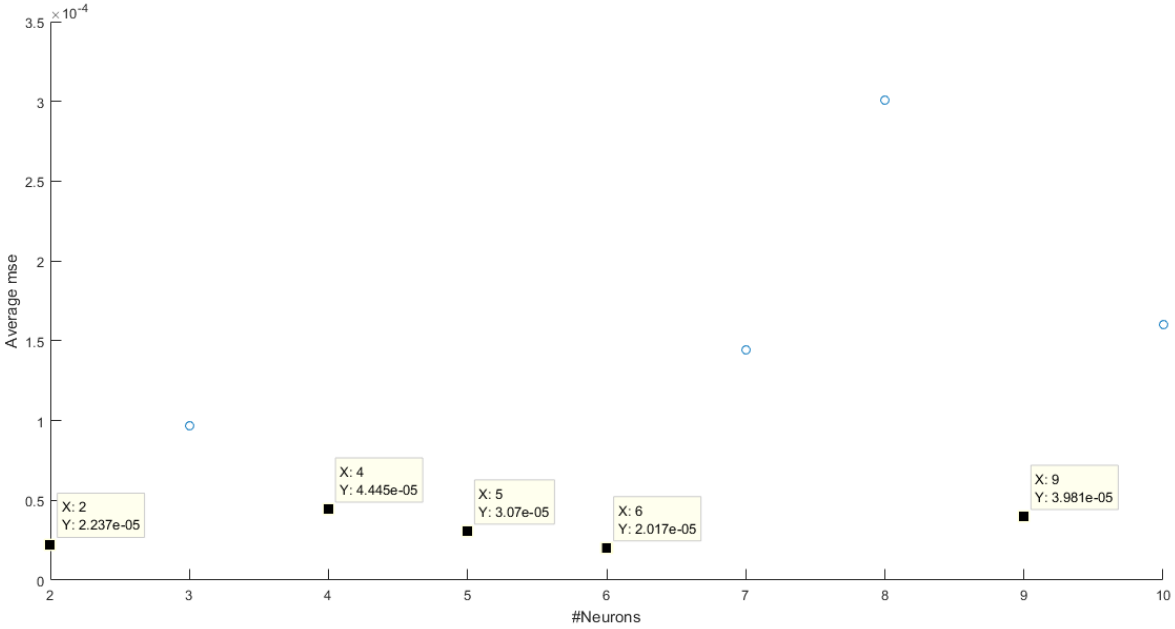
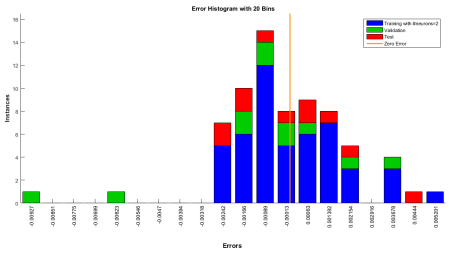
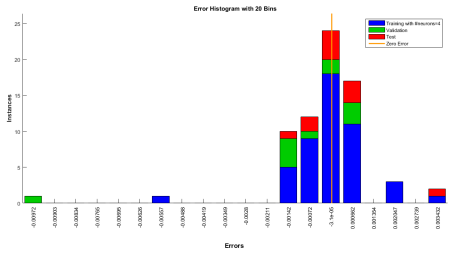


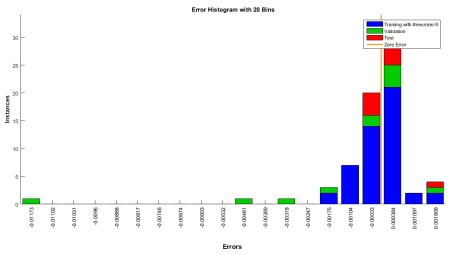
Figure 3.10: CL2 Error Means



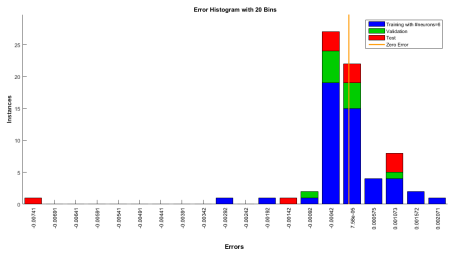
(a) 2 histo



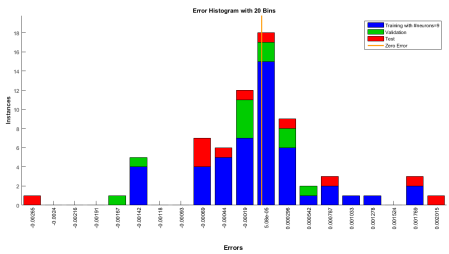
(b) 4 histo



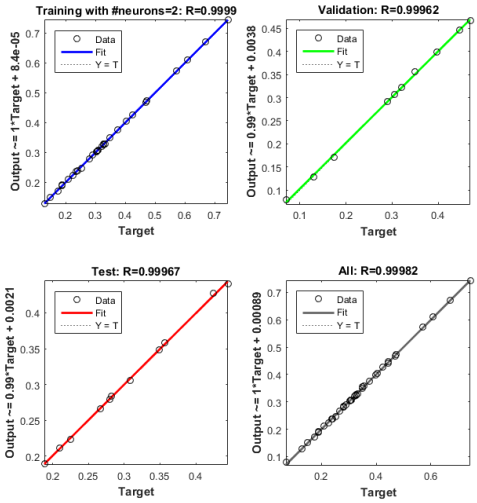
(c) 5 histo



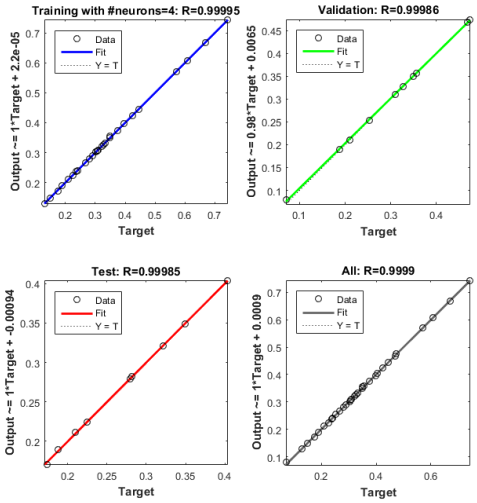
(d) 6 histo



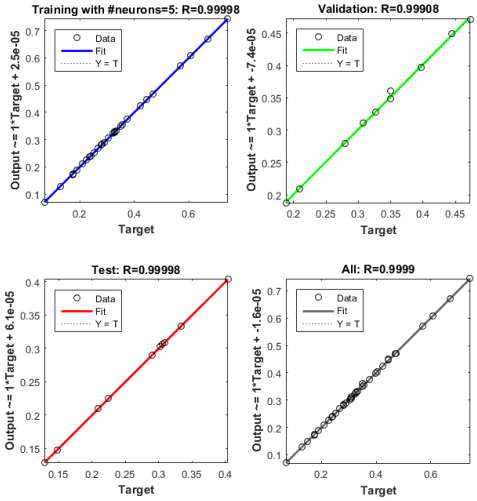
(e) 9 histo



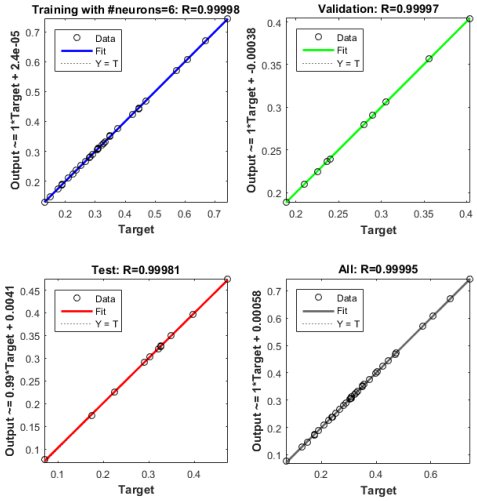
(a) 2 reg



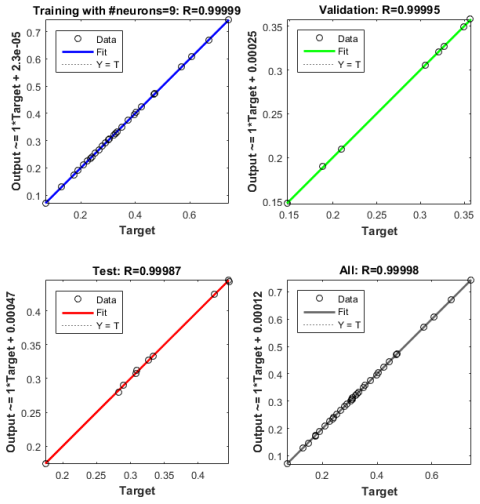
(b) 4 reg



(c) 5 reg



(d) 6 reg



(e) 9 reg

The chosen **MLP2** network has **four neurons** in the hidden layer.

3.2.2 RBF 2

As seen before, the previous function **create_rbf** and **choose_rbf** have been used to create and choose the most suited RBF network configuration.
The input data and desired outputs are the same as in the previous section.

The Performance graph created by the **create_rbf** is the following.

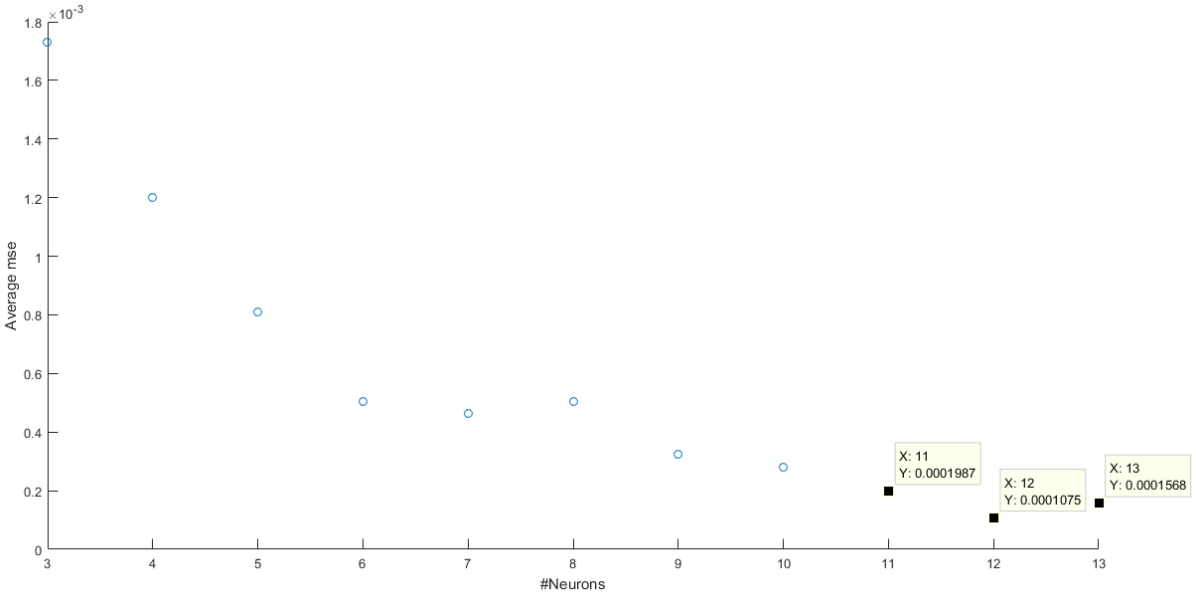
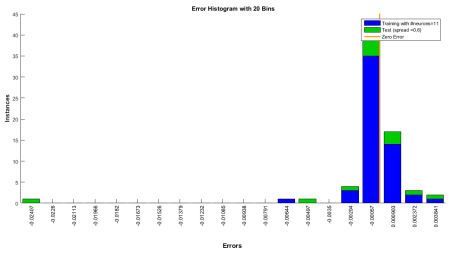
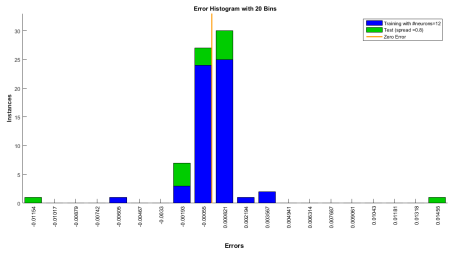


Figure 3.13: RBF2 Error Means

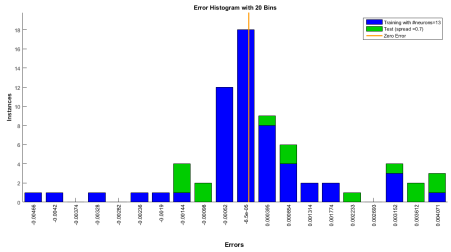
The best candidates are the 11-, 12- and 13-neuron configurations. The corresponding histogram and regression graphs are below:



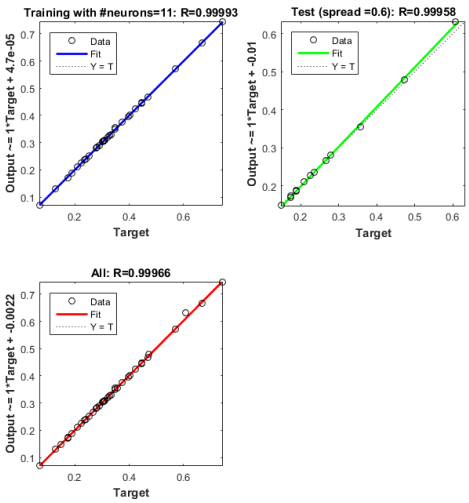
(a) 11 Hidden Neurons



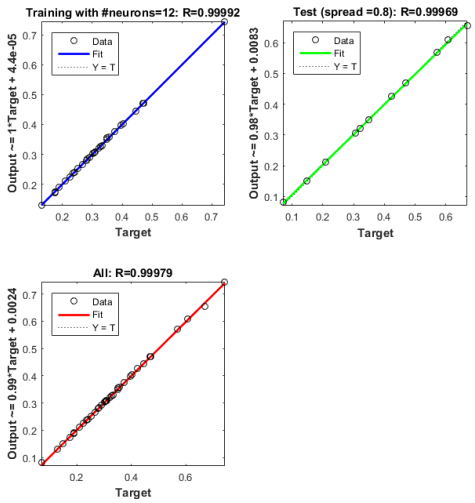
(b) 12 Hidden Neurons



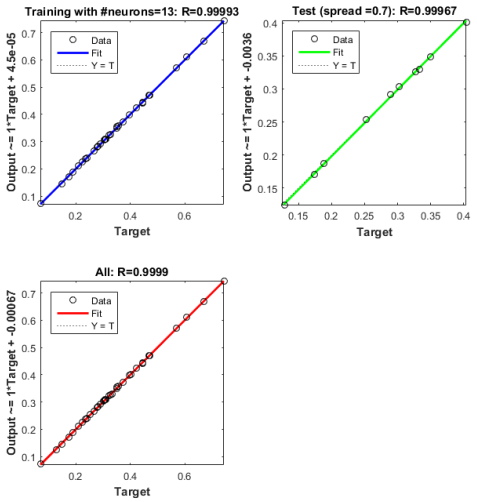
(c) 13 Hidden Neurons



(a) 11 Reg



(b) 12 Reg



(c) 13 Reg

The chosen network has **11 hidden neurons**, and the **spread** value is **0.6**.

3.2.3 ANFIS 2

The already-described *create_anfis* and *choose_anfis* have been used with the same input data and targets used for *MLP2* and *RBF2*. The Performance graph is:

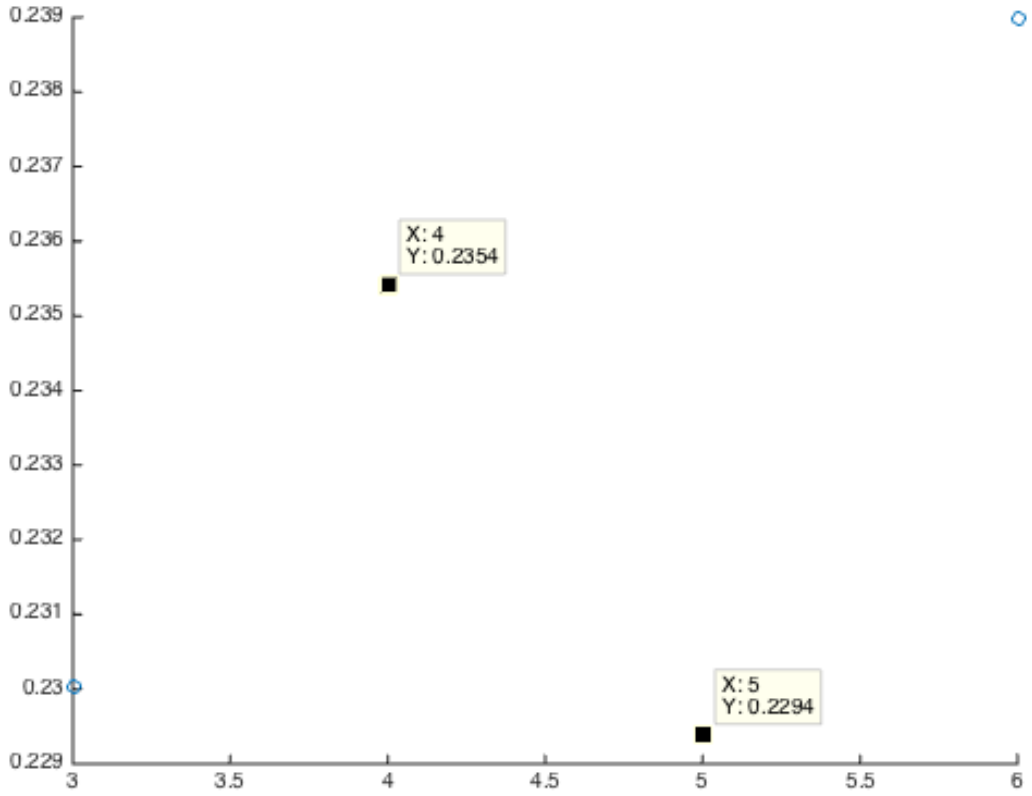
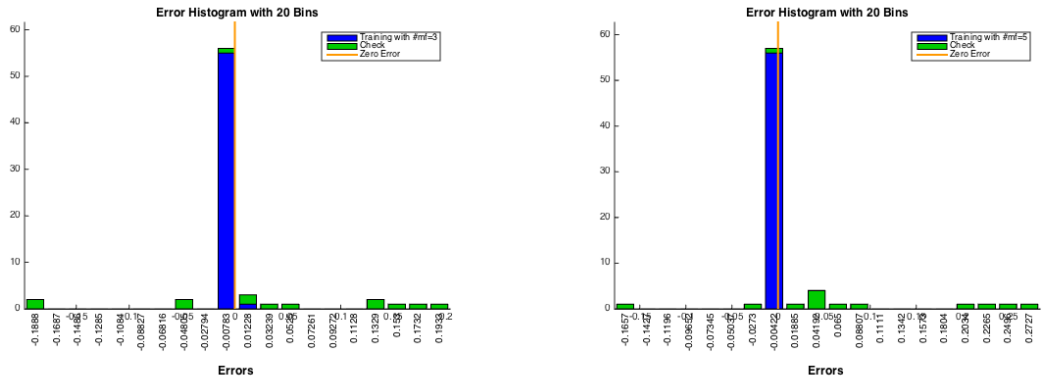


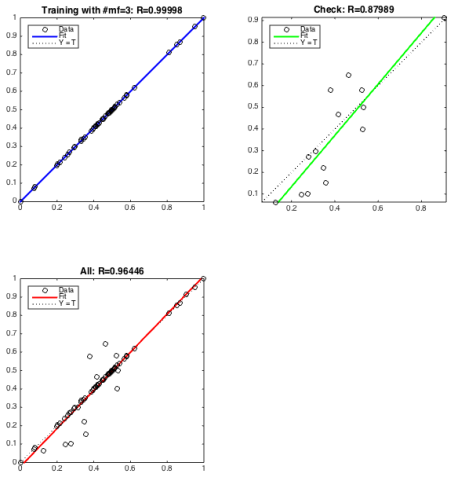
Figure 3.16: ANFIS2 Error Means

The networks with 3 and 5 membership functions are the best, and the corresponding graphs are as follows.

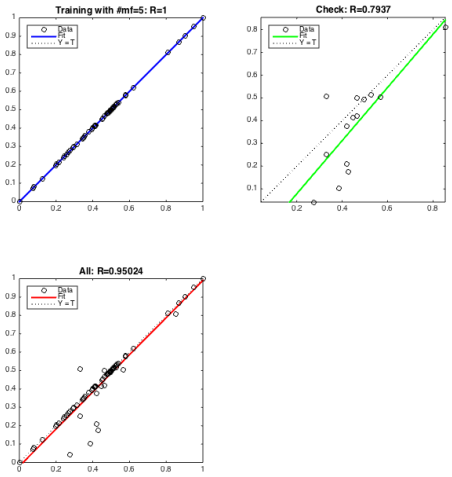


(a) 3 err

(b) 5 err



(a) 3 reg



(b) 5 reg

Finally, the chosen system has 3 membership functions.

3.3 Retrainig the classifiers

After the first part, consisting in the training of the two classifiers using supervised data, the problem required that another round of training be done with ***unsupervised data***.

Determining the thresholds. Before we could find the unsupervised data, we needed to find the thresholds to classify the output space. Analyzing the distribution of the points of the two target outputs ***task_perc*** and ***task_caution***, we decided to use the threshold that gave the lowest number of outliers:

- L1 = 0.23;
- H1 = 0.7;
- L2 = 15;
- H2 = 0.59;

The thresholds are shown in Figure 3.19, where *task_perc* is on the X-axis and *task_caution* on the Y-axis.

Next, a new set of input was created using the **new_input** function.
With this function, the number of actions performed by each worker (that is, the input of networks **R1**, **R2** and **R3**) was perturbed: a random variable *choice*, which can take any integer value between 1 and 4, is used to decide whether to increase or decrease the risk in four levels:

- "much more risk": if possible, a new high-risk task is added;
- "more risk": if possible, a new medium-risk task is added;

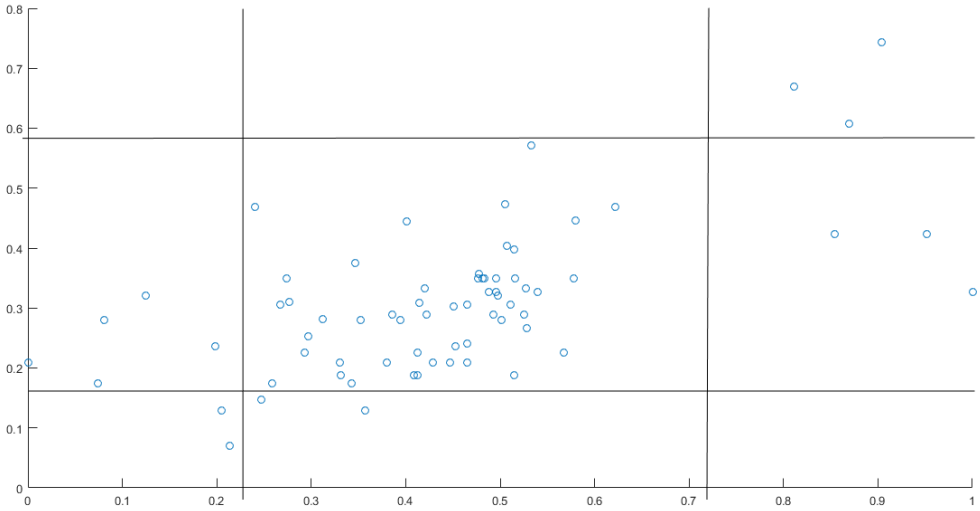


Figure 3.19: Thresholds

- "less risk": if possible, a medium-risk task is deleted;
- "much less risk": if possible, a high-risk task is deleted.

The new output of Classifier2 is then calculated using its corresponding MLP network. The idea is that, for each worker, the input of Classifier1 has to follow the change made on their number of tasks. This is achieved by **modifying all the inputs of GP and Classifier1** (F1, F2, F3, F4, F5 and F6) by adding or subtracting $step1 = 0.01$, or $step25 = 0.25$ (the latter is used for those parameters which seemed to only take the discrete values: 0, 0.25, 0.5, 0.75, 1), **until the output of Classifier1 is coherent to the output of Classifier2** (i.e. low, medium, or high), or until the parameter cannot be modified any further.

The explained procedure is repeated for all the workers.

The next step consists in computing the outputs, for each neural networks and fuzzy systems, corresponding to the perturbed input.

To obtain the new target outputs, a script called **training** is used: the script adapts the output of Classifier1 creating the target of Classifier2, and the other way around. The adaption is made through a function called **transform_output**, which uses the L1, H1, L2, H2 thresholds to project the output of Classifier1 into the same position of the same classification of Classifier2: for instance, if output o_i lies on the average point of the "medium" interval for Classifier1, the function creates target t_i, to be used by Classifier2, which lies on the average point of the "medium" interval of Classifier2.

The *training* script also creates a set of new networks and fuzzy systems, trained and tested on the union of old supervised data and the new, unsupervised, data; the targets are the union of the old target outputs and the new ones, created with the aforementioned *transform_output* function. In the case of MLPs and RBFs, we decided to create new

networks instead of retraining the old ones because our Matlab version does not support the *adapt* function. We then decided to follow the same procedure for the ANFISs.
For the creation of the networks and fuzzy systems, we used the functions described in Sections **Classifier 1** and **Classifier 2**.

3.3.1 Results

In this section, we discuss the results obtained in the process of retraining, making some considerations on the number of mismatches obtained on the classifications by Classifier1 and Classifier2.

The mismatches are computed by the function **classify**, which takes as inputs the outputs of the corresponding networks/systems of Classifier1 and Classifier2, and the four thresholds. The function returns [***[c, m, i]***], where ***m*** is the number of mismatches, **c** is the number of correctly classified samples, and ***i*** is a vector whose elements are set to 1 if the corresponding indexes generated a mismatch, 0 otherwise.

Results after the first training

Below is the number of mismatches obtained with the 70 *supervised* inputs with the networks and fuzzy systems from Sections **Classifier 1** and **Classifier 2**. The desired outputs are obtained using as input the 70 supervised targets.

- MLP: 11
- RBF: 12
- ANFIS: 22

Intermediate results

Below is the number of mismatches obtained with the 70 *unsupervised* inputs (from Section 3.3) with the networks and fuzzy systems from Sections **Classifier 1** and **Classifier 2**. The desired outputs are the ones created in 3.3.

- MLP: 4
- RBF: 19
- ANFIS: 17

The number of mismatches given by the MLP networks is very low, as those are the very networks that have been used to align the results of the two Classifiers in the unsupervised inputs. The four mismatches are given by those workers who have been subjected to the strongest perturbations, or because of their original data: either way, they could not surpass the corresponding threshold and they are probably outliers.

Results after the second training

Below is the number of mismatches obtained with the union of 70 *supervised* input data and 70 *unsupervised* input data, with the new networks and fuzzy systems from Section 3.3. The desired outputs are the union between the supervised targets and the new targets created in Section 3.3.

- MLP: 19
- RBF: 17
- ANFIS: 24

Conclusions

- The MLP networks, after the first training, could not classify correctly 15.7% of the data. With the newly created MLP networks, we found that 13.6% of the input data was mismatched. The second training improved the networks by 2.1%.
- The RBF networks, after the first training, could not classify correctly 17% of mismatches. With the newly created MLP networks, we found that 12% of the input data was mismatched. We saw an improvement of 5%.
- Lastly, the fuzzy systems, after the first training, produced a total of 34.2% mismatches, The newly created ANFIS gave 17.14% mismatches, giving the best improvement of the three systems.

The newly created networks and fuzzy systems have been trained and tested on the very same data that we are now using, so the improvement is justified. Moreover, the method used to construct the new sets of inputs and target outputs made it possible for Classifier 1 to learn what Classifier 2 had learned in the first training; and viceversa. Finally, since the unsupervised input set has been adapted to the "old" MLP networks, which are the best performing networks, the RBF networks and the ANFIS had the chance to learn what the MLP networks had previously learned: this explains the higher improvements in the RBF networks and the fuzzy systems.

4 Genetic Algorithm

The third part of the project consists in creating a population of chromosomes containing a set of MLP, RBF networks and fuzzy systems implementing the classifier described above, and a set of functions to manage the population. A genetic algorithm is, then, applied to the population, in order to minimize the error given by the neural networks and fuzzy systems, to obtain the best configuration for the classifiers.
The functions that we wrote, *NN_initpop*, *NN_mutation* and *NN_fitness*, will be used in the ***globaloptim2*** Matlab environment, implementing a complete genetic algorithm.

We started with the creation of a population of Chromosomes: each chromosome is divided in two parts, each one containing an MLP network, an RBF network, an ANFIS and a selector s in 1, 2, 3. The selector selects the network/system used by that part: 1 corresponds to MLP, 2 to RBF and 3 to ANFIS.

The structure of a single Chromosome is the following: Each chromosome is implemented

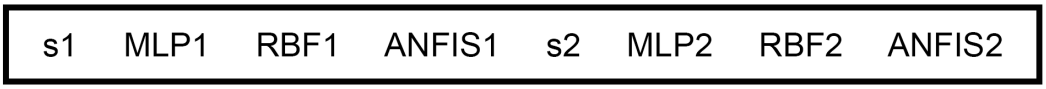


Figure 4.1: chromosome

with a structure containing all the 8 aforementioned fields. The initial population has 50 Chromosomes, arranged in an column array: We then developed the three functions which create and manage the population: *NN_initpop*, *NN_mutation* and *NN_fitness*

4.1 NN_initpop

The first developed function is the *NN_initpop()*, which initializes a population of as many chromosomes as specified by the user. For each chromosome, selectors s1, s2 are randomly initialized and all the networks and fuzzy systems are created with random parameters. Namely, the parameters chosen at random are the number of hidden units in the MLP and RBF networks, the spread in the RBFs and the number of membership functions of the ANFIS. To train and test the networks, we used the union of the supervised input data and the unsupervised input data, with the corresponding targets as explained in Section 3.3.1.

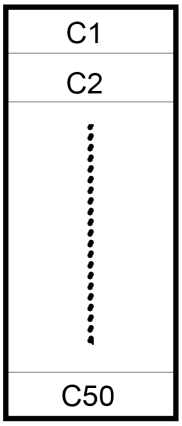


Figure 4.2: popolazione

4.2 NN_mutation

The *NN_mutation* function creates a new population from the current one. The size of the new population, ***mutatedPopulation***, varies depending on the size of *parents*, a vector containing the indices of the chromosomes of the current population selected for mutation.
The mutations performed on the chromosomes to obtain mutatedPopulation depend on the value of the selector in each part (that is, they depend on the kind of neural network or fuzzy system they are applied to). The mutations are described as follows:

- If the selector s selects an MLP network, the weights are perturbed with a given probability: they are multiplied by a random real parameter n in [0.99, 1.01].
- If s selects an RBF network, the centers and spreads of the Gaussian functions, and the output linear coefficients are multiplied by a random real paramenter n in [0.99, 1.01]. The centers and the spreads are maintained within the interval [0, 1].
- if s selects an ANFIS, the centers and spreads of the Gaussian membership functions, and the output linear coefficients are multiplied by a random real paramenter n in [0.99, 1.01].

4.3 NN_fitness

The *NN_fitness* function computes the fitness score of each chromosome in the population. In our case, the fitness function coincides with the two objective functions: the error between the desired output and the actual output of the network/system, and the number of mismatches.

For each chromosome, and for each part of the chromosome, the function computes the outputs of the two neural networks/fuzzy systems selected by s, and then computes the Mean Squared Error between the target output and the actual output. Next, a score is assigned to the chromosome for each mismatch occurred, depending on the kind of mismatch:

- 0 points if no mismatch occurs;
- +1 point if the classifications differ by one class (e.g., Classifier1 predicts 'low' and Classifier2 predicts 'medium');
- +2 points if the classifications differ by two classes (e.g. Classifier1 predicts 'low' and Classifier2 predicts 'high').

Finally, the MSEs of Classifier1 and Classifier2 are summed up to form the first field of *Score* (i.e. the parameters of the objective functions to be minimized), while the second field is the mismatch score of the two classifiers in the chromosome.

4.4 Main

The ***main*** function is used to run the genetic algorithm. The function takes the desired dimension of the initial population of chromosomes and the desired number of generations. An **options** structure has been initialized to specify the parameters to use: *NN_initpop* and *NN_mutation* are used, respectively, as creation function and mutation function. To run the algorithm, the function ***gamultiobj2*** has been called, with *NN_fitness* as fitness function.

4.5 Results

The genetic algorithm has been executed with an initial population of 50 chromosomes and a number of generations equal to 250. The results are as follows.

Pareto Front. The solutions are arranged in the typical hyperbola-shaped graph, as shown in Figure 4.3. In the Figure, the best solutions were highlighted and have been consequently examined for more details (please note that, as explained in Section 4.3, the values on the Y-axis do not correspond to the number of mismatches, while the values on the X-axis correspond to the Mean Squared Error).

The number of mismatches obtained by a given pair of classifiers is **at most** equal to the value found on the Y-axis. Seeing how the thresholds for the classes were constructed in Figure 3.19, we can say that there are **at least 9 workers that are outliers**, and cannot be correctly classified. This means that the two pairs of classifiers **(5.076, 2)** and **(4.506, 5)** *"overclassified"* even those workers who cannot be classified.
Thus, we decided to pick the pair of classifiers **(4.384, 8)** (***Pair 8***)for further examination.

Classifier 1 of *Pair 8* is an RBF network, while Classifier 2 is a MLP network. The inputs which could not be correctly classified by *Pair 8* are the following: 1, 2, 3, 12, 37, 71, 72, 89, which corresponds to the original, supervised, inputs of workers 1, 2, 3, 12 and 37, and the perturbed, unsupervised, inputs of workers 1, 2 and 19.

4.5.1 Conclusions

As shown in Figure 3.19, workers 1, 2 and 3 are not outliers, so the networks could not correctly classify them.
The networks could not classify correctly workers 12 and 37, but they are outliers: the

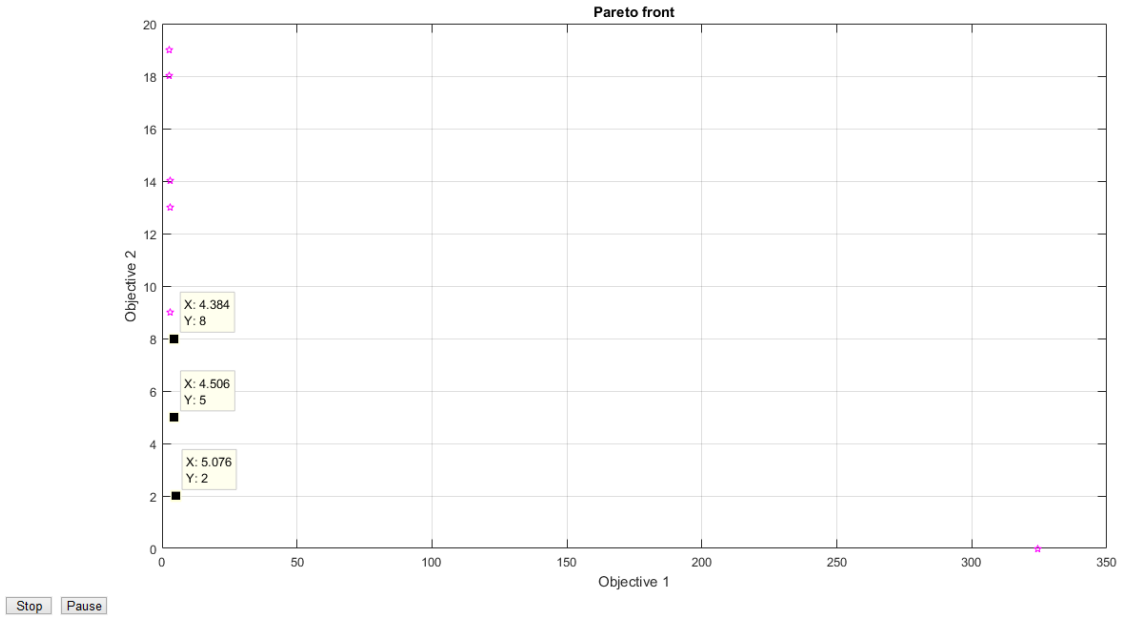


Figure 4.3: Pareto Front

response obtained by the network, is, thus, correct.

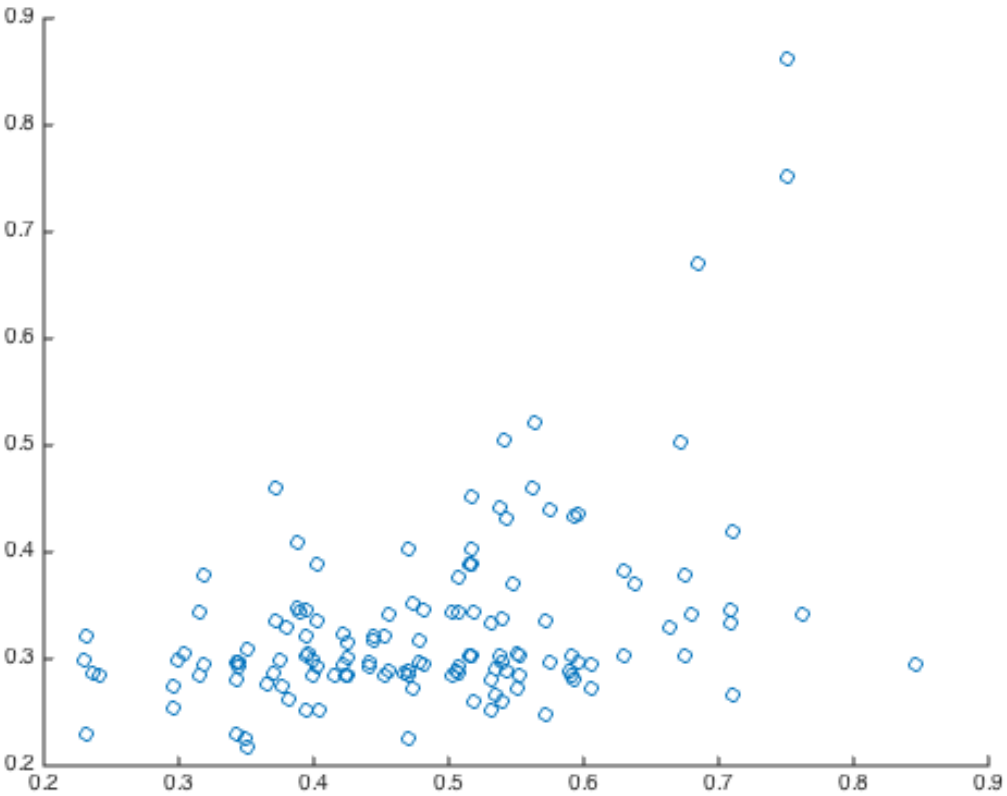


Figure 4.4: Outputs of *Pair 8*

Workers 1 and 2 could not be classified correctly even after being perturbed. The perturbation on workers 3, 12 and 37 allowed the networks to recognize and classify them. After the perturbation, worker 19 could not be correctly classified: we can assume that the perturbation on their data has been too severe for the networks to recognize them.

Of the 9 outliers on the supervised input, *Pair 8* could only recognize 2, while it could not classify 3 workers that are not outliers.