Instituto Superior Técnico

2023/2024 - 4th Period

Computational Intelligence for the Internet of Things

1st project
Fuzzy Systems and Neural Networks
Part I & II

Prof: Joao Paulo Carvalho

Group 3

1. **No:** 94230 **Name:** Samuel Barata
2. **No:** 96765 **Name:** Sandra Castilho

May 20, 2024

1 Introduction

The explosive growth of Internet of Things (IoT) devices has resulted in a high volume of data being generated, but most of it is not exploited or used at all, since sending all that device-generated data to a centralized data center or to the cloud causes bandwidth and latency issues.

One solution to this problem is to take advantage of the edge devices, using their spare computing capabilities to enable faster and more comprehensive data analysis, without exchange the data over to the cloud (which also reduces the network load).

However, one should be aware not to exhaust the edge resources, to not compromise the normal operation of the edge devices or the network bandwidth.

The objective of this project is to build an intelligent system that manages the computing load assigned for Edge computing tasks on an Edge device (CLP), without compromising the main role of the device. This consists on using edge devices to compute as much as it possibly can without affecting its own edge functionality and without increasing congestion on the network.

2 Fuzzy System

2.1 Introduction

The objective is to create a Fuzzy Inference System (FIS) capable of controlling the variation of the Computing Load Percentage (CLPVariation), given some system parameters as input (such as memory, processor, network, etc...). It is assumed that when the device is operating as a pure Edge device (i.e., CLP = 0), only networking duties (including routing data to the cloud) are performed.

2.2 Architecture and Decisions

To help construct the fuzzy system, we have the following system features:

- Memory Usage
- Processor Load
- Input network throughput
- Output network throughput
- Available output bandwidth
- Latency

Each feature is a one-minute average, and for each one, the variation rate is also available.

2.3 Variables

Because Fuzzy systems' rule base size increases exponentially with the number of inputs, we began by selecting the important features of the system to use, and also how to combine/group them in smaller fuzzy systems.

We decided not to use the Variation values, as it would overcomplicate the system, and the insights that we would get from them would not be very useful. For example, it was hypothesized to group the Processor Load and its Variation, considering that a positive variation for a "long" period of time would mean a high load on the device and the CLP should decrease. However, we do not have a way to record the state, and knowing the current state of the processor load of the device is enough, as we want to keep its usage around a certain value (70%-80%).

Because the *Memory Usage* and *Processor Load* are a good indicative of the current performance of the system, and since either of them having extremely high usage will have a great impact on the device performance, we joined these two features to be the inputs of a fuzzy system (FS_HDR) that have as output *Hardware Resources*.

On the same train of thought, we tried to combine the network features (*Output Throughput, Available Output Bandwidth, Input Throughput* and *Latency*) on a fuzzy system whose output was *Network Usage*, to have an idea of how congested the network could be.

Finally, we computed the value for the CLP based on the *Hardware Resources* and *Network Usage*.

The original architecture is presented in figure 1.

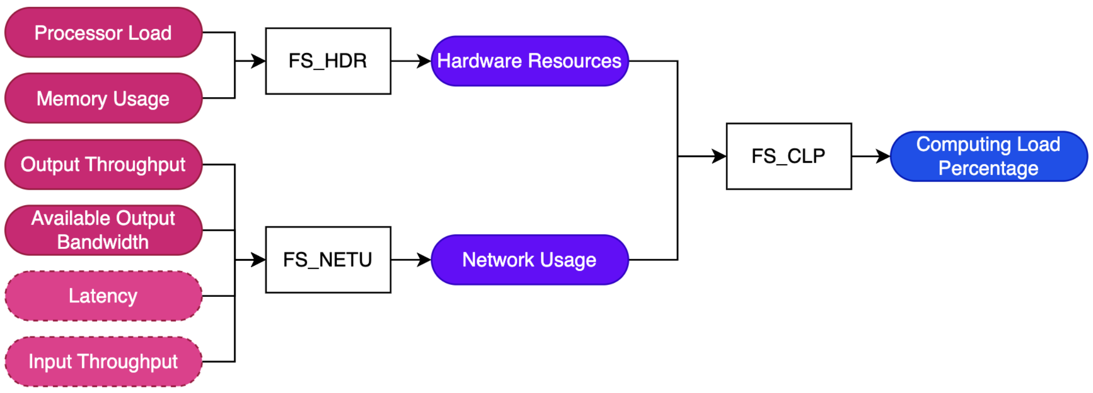

Figure 1: Original Fuzzy systems architecture

We experimented with combining the features of the *Network Usage* and reached the following conclusions:

- Because the input network and the output network are not necessarily correlated, we shouldn't join them in an "outer" fuzzy system, but instead consider the *Input Throughput* in the final Fuzzy System (the one that computes the CLP), as a "tiebreaker" of Hardware and Network load, as high *Input Throughput* implies more data to be processed - and thus an incentive to process it locally. However, while constructing the FAM (Fuzzy Associative Memory) for it, we concluded that its weight was meaningless, and discarded it.
- *Latency* was discarded from the *Network Usage*, considering that the packets should reach their destination anyway with high latency, but a congested link will mean loss of packets, and that is worse than a slow link. However, the values that we were getting for the CLP value when considering the *Latency* as an input of the final fuzzy system were not satisfactory, and decided to discard it as well.

In the end, we ended up with a fuzzy system that computes the *Output Congestion* (FS_OUTC), using as input the *Output Throughput* and *Available Output Bandwidth*. The value for the CLP is then based on the *Hardware Resources* and *Output Congestion*.

The final architecture of the project is presented in figure 2.

2.3.1 Linguistic Terms & Membership Functions (Fuzzy Sets)

We chose the Linguistic Terms "High", "Medium" and "Low" for all the input Fuzzy Sets, and "Increase", "Maintain" and "Decreased" for the output *CLP Variation*, and to allow a better categorization of the fuzzy sets, we decided to use trapezoids over triangles. After testing the fuzzy systems, it was necessary to add more members to some fuzzy systems, as shown in the membership functions below (for example, adding a "very high" member to the *Processor Load*, or adding "decrease much" and "increase much" members to the *CLP Variation*).

The Membership Functions for our fuzzy systems are presented in the figures 3, 4 and 5.

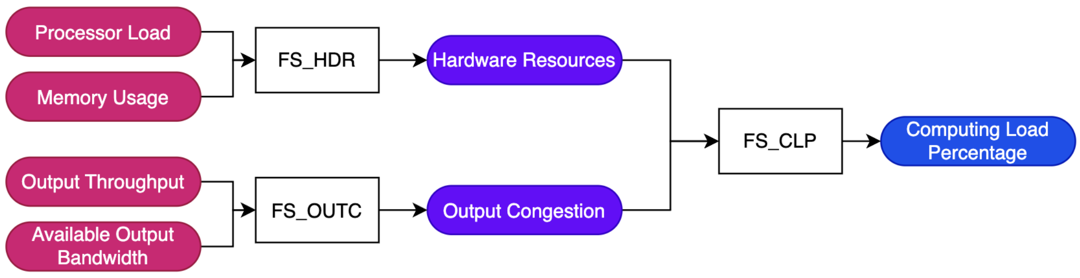

Figure 2: Final Fuzzy systems architecture

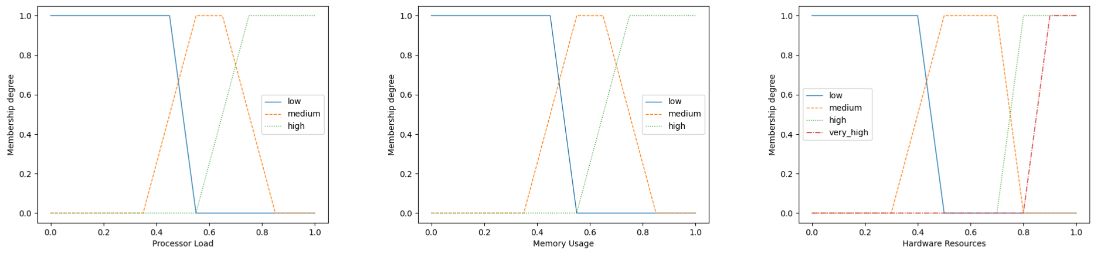

Figure 3: Membership functions for the "Hardware Resources" Fuzzy System

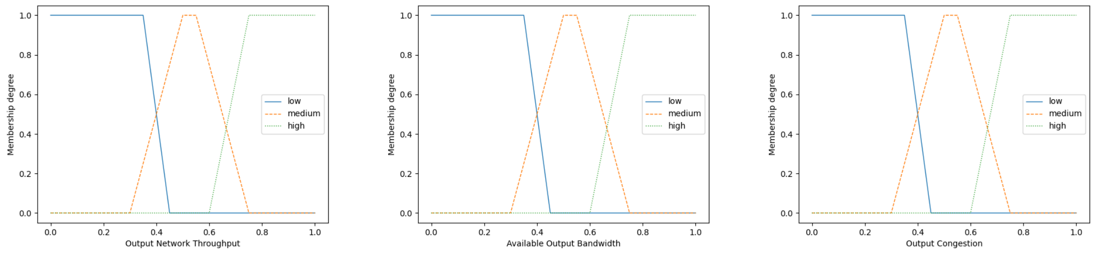

Figure 4: Membership functions for the "Output Congestion" Fuzzy System

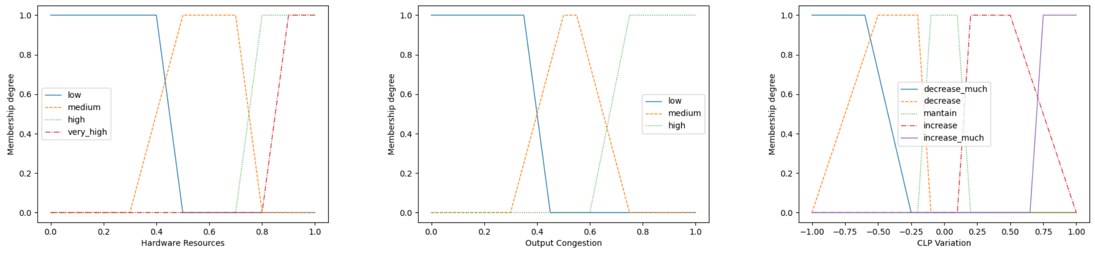

Figure 5: Membership functions for the "CLP Variation" Fuzzy System

2.3.2 Fuzzy Rules

Because the output of our final Fuzzy System is a Fuzzy Set, we decided to use a Mamdani Fuzzy Model, with the fuzzy rules shown in the FAMs 1, 2 and 3.

- **Hardware Resources Fuzzy System**

		Processor Load		
		Low	Medium	High
Memory	Low	L	M	H
	Medium	M	M	H
	High	H	H	VH

Table 1: FAM for Hardware Resources Fuzzy Rules

- **Output Congestion Fuzzy System**

		Output Bandwidth Available		
		Low	Medium	High
Network Through-put	Low	M	L	L
	Medium	H	M	M
	High	H	H	H

Table 2: FAM for Output Congestion Fuzzy Rules

- **CLP Variation Fuzzy System**

		Output Congestion		
		Low	Medium	High
Hardware Resources	Low	↑↑	↑↑	↑↑
	Medium	0	↑	↓
	High	↓↓	↓	↓
	Very High	↓↓	↓↓	↓↓

Table 3: FAM for CLP Variation Fuzzy Rules

In figures 6, 7 and 8 are the surfaces induced by the rules for the 3 considered fuzzy systems.

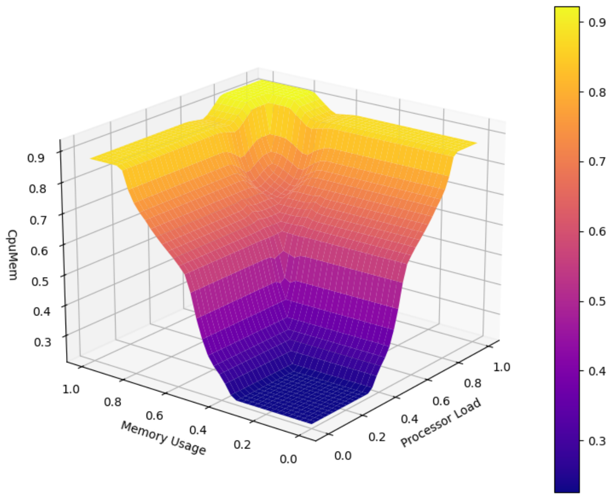

Figure 6: Surface induced by the Hardware Resources Fuzzy System

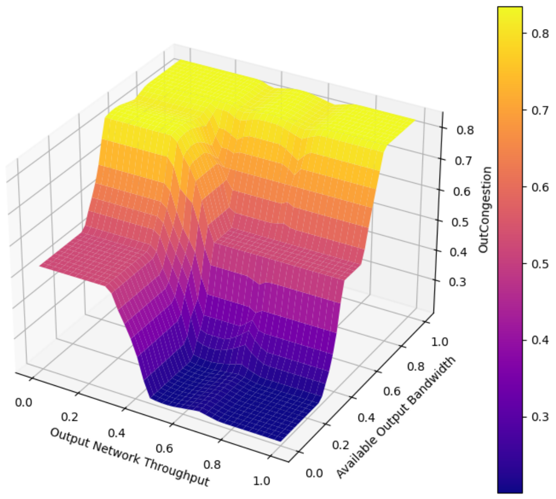

Figure 7: Surface induced by the Output Congestion Fuzzy System

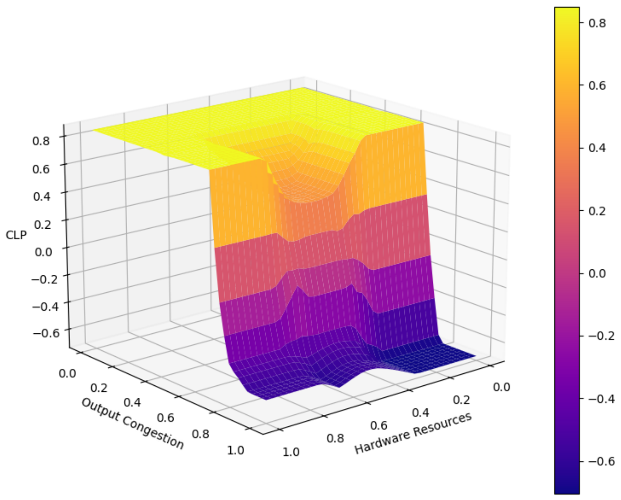

Figure 8: Surface induced by the Computing Load Variation Fuzzy System

2.4 Testing and Results

In refining our fuzzy system, we implemented several strategic adjustments to enhance its performance. Firstly, we meticulously modified the rule set to match better our anticipated output, ensuring a more precise correspondence between input variables and system responses. Additionally, we meticulously fine-tuned the trapezoid points, strategically adjusting them to optimize the system's accuracy and efficacy in generating outputs. As a direct outcome of these refinements, we observed a notable improvement in the system's overall performance. However, it's worth noting a particular consequence of these enhancements: our parameter "decrease much" has now a minimum of at -0.6. Even though the rules are effectively being applied as intended, this intrinsic limitation implies that our system's *CLP Variation* cannot extend beyond this threshold.

For the dataset provided, we obtained the results in table 4.

Predicted	Expected
84%	85%
85%	85%
85%	80%
81%	73%
59%	50%
52%	12%
-49%	-31%
-49%	-65%
-53%	-82%
-57%	-85%

Table 4: Predicted vs Expected *CLP Variation*

As we can see in Table 4, the last 2 lines of the table have a big discrepancy because our system cannot output values below -0.6. The only line that requires more attention is the 6th, where our output is 52% and the expected value is 12%. After analyzing the data, we concluded that with the variables we chose to use for our system, we are unable to provide a different output. Only a system with more inputs could produce a closer output to the expected values.

3 Neural Networks

3.1 Introduction

The objective is to implement a model based on a Multilayer Neural Network (NN) to control the variation of the Computing Load Percentage (CLPVariation), given some system parameters as input (such as memory, processor, network, etc...). This neural network should model the Fuzzy System developed in section 2.

3.2 Datasets

The datasets used to build the model were generated using a randomized script and the Fuzzy System previously mentioned.

First, we developed a script to randomize the inputs of the fuzzy system, and then feed them to the fuzzy system to obtain the expected output of the neural network.

To avoid overfitting the model, we must use 3 datasets: Training, Validation and Test. Instead of generate a mega-dataset and split it, because we wanted to cover the input space uniformly, we decided to generate the datasets independently, knowing that that the points shouldn't overlap in the datasets (that is, there shouldn't be the same inputs in the 3 datasets).

With "cover the input space uniformly" in mind, we decided to generate the points sequentially for the training dataset, to avoid clusters and obtain a balanced dataset. First, we generated numbers using 11 training points for each of the 4 used features *(Processor Load, Memory Usage, Output Throughput and Output Bandwidth)*, generating numbers with a 0.1 interval (between 0 and 1), and randomized the values for the variations and the unused features, as these values wouldn't be used anyway as an input of the model. This created a dataset with $11^4 = 14641$ examples.

To validate and test the model, we decided to use truly random values for all the features, so we generated a data set with 10 000 000 points, and split it 50/50, to originate our Validation and Test Datasets.

3.3 Regression problem

The neural network is expected to replicate the results of the Fuzzy System. This is a regression problem, as the goal is to predict a quantitative value.

3.3.1 NN model

To begin training the model, we used a MLPRegressor model, with the *logistic* activation function and the *sgd* (stochastic gradient descent) solver. Next, it was needed to tune the parameters and hyperparameters of the model (number of inputs, number of hidden layers, etc), so we developed a script to iterate over them and find the best values. We started by testing our model by hand for a single hidden layer of 1 to 8 neurons and concluded that 5 neurons gave the lowest *mean squared error* as we can see in Table 5.

	MSE
Validate	0.0697423
Test	0.07008434

Table 5: Mean Square Error for 5 neurons with logistic activation and sgd solver

We then used *Grid Search* to optimize the parameters and discovered that this approach yielded improved results. Specifically, we achieved better performance with *activation='relu' and solver='adam'* compared to the original settings of *activation='logistic' and solver='sgd'*.

After that, we tested for combinations of 1 and 2 hidden layers and tested from 1->20 neurons on a single or 2 layers, and we reached the lowest error at 8 neurons on the first layer and 3 neurons on the second.

```
INFO: Best parameters found: {'activation': 'relu', 'alpha': 0.0001,
'hidden_layer_sizes': (8, 3), 'learning_rate': 'constant', 'max_iter': 2000,
'solver': 'adam'}
INFO: Mean Squared Error with best parameters: 0.016218705321498234
```

From the output of the Grid Search function above, we obtain the final results: 2 hidden layers, the first with 8 nodes, the second with 3, *relu* activation function and the *adam* solver.

We trained our final model (illustrated in Figure 9) using these parameters and the training dataset. With this model, the Mean Squared Error for the validation dataset is 0.0162187.

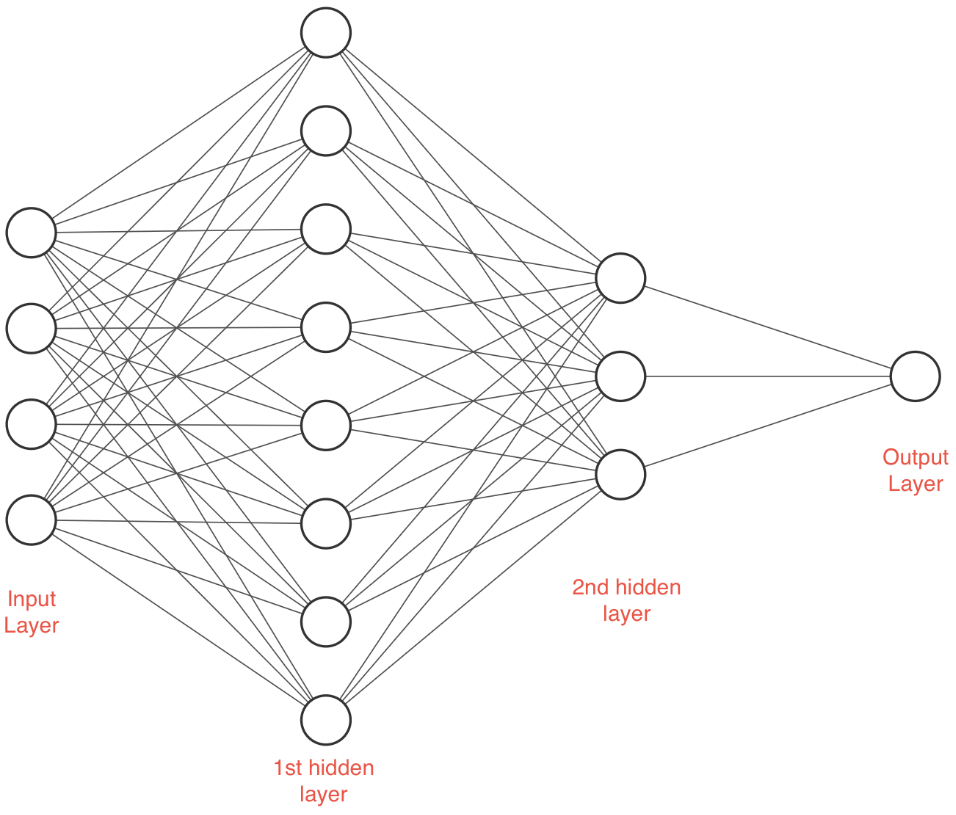

Figure 9: Visual representation of the final Neural Network

3.4 Results

To evaluate our final model, we tested it against our Test dataset. The results are presented in Table 6.

	MSE
Validate	0.0162187
Test	0.0162200

Table 6: Mean Square Error for 8+3 neurons with relu activation and adam solver

The mean squared error (MSE) of the model on the test set is calculated to be 0.0162200. This metric provides insight into the average squared difference between the actual and predicted values. A lower MSE indicates better model performance, with values closer to zero indicating a more accurate prediction. In this case, the MSE suggests that the model's predictions deviate from the actual values by an average squared error of approximately 0.016. We can also see that the 2 values for the errors are very similar, which suggests for once that the two datasets are similarly balanced, and twice that the final model should be a good model (without overfitting), predicting correctly any new value, that it hasn't been "seen" yet.

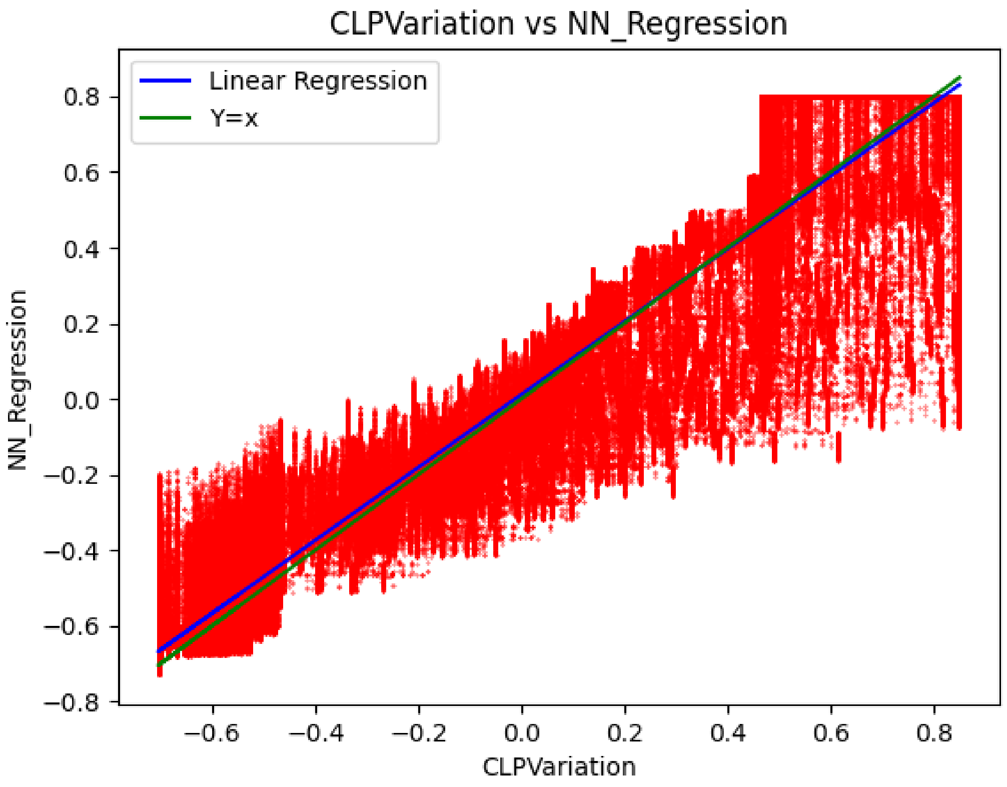

Figure 10: Linear Regression Expected CLP vs Neural Network Prediction

Figure 10 shows the representation of the linear regression of our model, the green line the regression of a perfect model. We expected the red points to fall near the green line. We can see that that is the case for the most points, given for the compact red areas around the line. However, we can also see that there are some outliers present, where some red dots deviate from the line on the right of the graph: where the model should predict a very high CLPVariation (> 0.5), is predicting a moderate increase (between 0 and 0.4). Although this might be a concerning issue, because the MSE that we got is so low, we shouldn't be concerned. In practical terms, this just means that we aren't using the Edge device capabilities to their possible maximum and also aren't contradicting the expected data. This suggests that the edge device should still be able to achieve its purpose effectively. While there may be slight deviations in the predictions, they remain within acceptable bounds, ensuring the continued functionality and reliability of the system.

3.5 Classification problem

We now want to use the NN as a multiclass classifier with the following 3 classes: "Decrease", "Maintain", "Increase".

To achieve that, we created a function (classification function) that maps the CLPVarition values to the 3 classes. The limits for this function are expressed in Table 7.

Class	Lower Bound	Upper Bound
Decrease	$-\infty$	-0.2
Maintain	-0.2	0.2
Increase	0.2	∞

Table 7: Classification Bounds

For this problem, we ran the NN model against the 3 datasets (Training, Validation and Test), and feed the output from the model to our classification function. To evaluate the results, we also feed the "correct" values to the classification function, and computed the classification metrics.

3.5.1 Results

In tables 8 through 13 are the classification reports (precision, recall, F1-score and accuracy metrics) and the confusion matrices for the 3 datasets, providing a detailed insight into the performance of the neural network model across the three defined classes: 'Decrease', 'Maintain', and 'Increase'.

	Precision	Recall	F1-score	support
Decrease	0.92	0.95	0.94	5989
Increase	1.00	1.00	1.00	7744
Maintain	0.60	0.47	0.53	908
accuracy			0.95	14641
macro avg	0.84	0.81	0.82	14641
weighted avg	0.94	0.95	0.95	14641

Table 8: Classification Report for Training dataset

		Predicted		
		Decrease	Increase	Maintain
Actual	Decrease	5703	0	286
	Increase	0	7744	0
	Maintain	478	0	430

Table 9: Confusion Matrix for Training dataset

The classification report for the training set, summarized in table 8, demonstrates exceptional performance in predicting the 'Decrease' and 'Increase' classes, with precision and recall values both around 0.92 and 1.00 respectively. This indicates that the model is highly accurate in identifying instances where the output is either 'very negative' or 'very positive', with zero false-positives and false-negatives in the 'Increase' class. This is expected, as this is the dataset used for training the Neural Network, where the weighs are computed to be the very best to meet the excepted output.

	Precision	Recall	F1-score	support
Decrease	0.98	0.97	0.97	1859972
Increase	0.98	0.98	0.98	2825928
Maintain	0.68	0.74	0.71	314100
accuracy			0.96	5000000
macro avg	0.88	0.90	0.89	5000000
weighted avg	0.96	0.96	0.96	5000000

Table 10: Classification Report for Validation dataset

		Predicted		
		Decrease	Increase	Maintain
Actual	Decrease	1794895	0	65077
	Increase	211	2782793	42924
	Maintain	33844	47120	233136

Table 11: Confusion Matrix for Validation dataset

	Precision	Recall	F1-score	support
Decrease	0.98	0.96	0.97	1858190
Increase	0.98	0.98	0.98	2827993
Maintain	0.68	0.74	0.71	313817
accuracy			0.96	5000000
macro avg	0.88	0.90	0.89	5000000
weighted avg	0.96	0.96	0.96	5000000

Table 12: Classification Report for Test dataset

		Predicted		
		Decrease	Increase	Maintain
Actual	Decrease	1792873	0	65317
	Increase	250	2784752	42991
	Maintain	33923	47101	232793

Table 13: Confusion Matrix for Test dataset

Because the Validation and Test datasets share the same results, they are analysed together. Both for the Validation and Test dataset, it is observed a high precision, recall and F1-score for the "Decrease" and "Increase" classes, with values between 0.96 and 0.98. However, the model's performance for the 'Maintain' class is notably lower, with a precision of 0.68 and recall of 0.74, resulting in an F1-score of 0.71 (for the 2 datasets). This suggests that the model has difficulty distinguishing instances where the output is closer to 0, leading to a higher rate of misclassification in this category. This discrepancy is likely due to the class imbalance in the dataset, where the 'Maintain' class has significantly fewer samples compared to the 'Decrease' and 'Increase' classes.

The overall accuracy of the model stands at 96%, which indicates that the majority of the predictions are correct. The macro average metrics, which treat all classes equally, show slightly lower values (precision: 0.88, recall: 0.90, F1-score: 0.89), reflecting the model's challenges with the 'Maintain' class. In contrast, the weighted average metrics, which consider the number of instances per class, align closely with the high overall accuracy, as the performance on the dominant classes ('Decrease' and 'Increase') heavily influences these averages.

Additionally, the confusion matrix for the Test dataset, as shown in Table 13, further illustrates the model's performance by summarizing the number of correct and incorrect predictions for each class. The confusion matrix reveals that the model accurately predicts the 'Decrease' and 'Increase' classes, with a large number of true positive predictions (1,792,873 and 2,784,752, respectively) and relatively low false positive and false negative predictions. However, the model struggles more with the 'Maintain' class, as evidenced by the higher number of false positive and false negative predictions compared to the other classes. These insights from the confusion matrix help identify areas for improvement and guide future refinements to enhance the model's predictive capabilities.