UNIVERSITÀ DEGLI STUDI DI BERGAMO

Scuola di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Classe N. LM-32 – Classe delle lauree magistrali in Ingegneria Informatica

Robustness evaluation of the estimation provided by a Multilayer Perceptron

Relatore:

Chiar.mo Prof. Angelo Michele Gargantini

Correlatori:

Chiar.ma Dott.ssa Silvia Bonfanti

Chiar.mo Dott. Andrea Bombarda

Tesi di Laurea Magistrale

Rita PEDERCINI

Matricola n. 1041193

ANNO ACCADEMICO 2019 / 2020

Abstract

Machine learning is a set of methods and algorithms that allows us to learn models starting from data, without deep knowledge about the domain. It allows solving complex tasks in many different fields such as autonomous driving, image recognition, speech recognition. Nowadays, machine learning is being more and more important for the healthcare sector as well. It supports humans for many applications, including the development of new medical systems, the treatment of chronic diseases, and the handling of patient data. For the deployment of a machine learning model, especially for critical systems, its robustness is an important aspect to take into account. The robustness concerns the ability of a system to maintain its grade of performance when dealing with unseen data. The performances of a model depend on the training data, therefore if it is not representative of the real world, the model can have anomalous behaviors with new samples. This issue can happen in many real applications and in general in the healthcare sector because of changes in the environment or system, but also for issues in data acquisition. Nowadays, the most approaches to evaluate robustness refer to classification problems, while few works are focus on regression tasks. This work describes a new interpretation of some approaches, created for computer vision problems, to evaluate the robustness of regression models. In particular, this thesis presents a framework to compute the robustness of a regression model. In fact, the case study is based on a multilayer perceptron trained to predict the blood pO2 values, based on the curve obtained in response to a bright pulse. The robustness evaluation highlights the main vulnerabilities of the model. To overcome the model weaknesses, the classical solution is the data augmentation method. This thesis describes some new interpretations of data augmentation techniques and a novel approach of incremental learning methods to improve the performance of regression models such as this case study. The robustness evaluation has been performed not only using natural inputs, inputs data which could occur in practical scenarios, but also adversarial examples, input data designed to cause the model to make a mistake. Furthermore, for healthcare systems, the awareness of the uncertainty in predictions is another important property. The determination of the output confidence can be performed through Bayesian neural

networks, which model uncertainty both on the model and on data. This thesis presents the comparison of performances and robustness between Bayesian neural networks and the classical multilayer perceptron model. This project is realized in partnership with AISent team and refers to a real case study.

Acknowledgements

I'd like to thank my supervisor, Professor Angelo Michele Gargantini, for helping me during all this work.

Special thanks to Silvia Bonfanti and Andrea Bombarda, for all precious advice and help.

I wish to offer a word of thanks to all the AISent team that provided this opportunity and supported in this work.

Lastly, I'd like to say special thanks to my family and my friends for their support in this period of my life.

Contents

Document Outline

This chapter describes the structure of the document providing briefly the concepts discussed in each section.

- **Chapter 1** introduces the reader to the case study describing the context, dataset, network and metrics

- **Chapter 2** provides basic knowledge about neural networks, focusing on the neural network used in the case study, Multilayer Perceptron. It mainly describes its architecture, the processing of the dataset and the training phase.

- **Chapter 3** describes machine learning testing, focusing on the robustness metric. It introduces novel robustness metrics that can be applied in regression models. Furthermore, it provides some advice on the choice of the best robustness metric based on the problem analyzed.

- **Chapter 4** describes a testing framework to determine the model's robustness in perturbed environments. It also reports an example of its application, step by step, on the case study.

- **Chapter 5** provides an exploration of data augmentation techniques and incremental learning method. The chapter introduces also novel methods to enrich the input dataset for regression problems using both the data augmentation and incremental learning techniques. Furthermore, you can see all the results obtained applying these methods on the original dataset and a comparison of robustness analysis for the new models re-trained.

- **Chapter 6** introduces the reader to the Bayesian neural network, a different type of neural network that models uncertainty. It presents the main approaches to realize these networks and the different types of uncertainty to model. Furthermore, it reports the implementation and evaluation of Bayesian neural networks using the variational inference approach through Tensorflow Probability.

- **Chapter 7** presents the robustness analysis of Bayesian neural networks, using the same framework and functions implemented for MLP models. Moreover, it shows how data augmentation can be applied to improve the robustness for this type of network. In this chapter we will present also novel metrics to evaluate Bayesian neural networks, considering the output distribution completely.
- **Chapter 8** presents the main methods provided by the literature to evaluate adversarial robustness. It describes the implementation of the Fast Gradient Sign Method for the evaluation of this case study, both for the multilayer perceptron model and the Bayesian neural networks.
- **Chapter 9** summarizes the results obtained in this work and presents some future works on this topic.

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning is a field of computer science based on the idea that systems can learn from data and improve their performance through statistical and modeling techniques.

This approach involves building and adapting models through specific algorithms, using the experience achieved from data, in order to improve their ability to make predictions. This feature allows researchers and data scientists to solve complex problems without having complete knowledge of the domain.

Despite the term "machine learning" was coined in 1959, only in recent years these methods are growing so fast. This is due to many factors: faster computational speed, in particular thanks to the development of specific hardware called Graphic Processing Unit, cheap storage of data, big data availability, multi-modal and heterogeneous source of data. Furthermore, this subject includes many algorithms, which differ in the learning style and their function, resulting in different output models. Thanks to this variability, machine learning can be applied in many industrial applications to perform different tasks and work with large amounts of data. For example, it can be used for natural language processing, bioinformatics and medical diagnosis, image processing and pattern recognition, search engines, financial market analysis.

One of the main challenges of machine learning models is testing, which can be described as any activity aimed at detecting differences between existing and required behaviors of machine learning systems [27]. In fact, machine learning systems are based on different structures and methods compared to traditional systems. In particular, especially for complex types of neural networks, data scientists and machine learning users are not able to understand exactly how the model makes predictions/decisions; it is not possible to express analytically the function used by the model. For these reasons, machine learning systems are thus sometimes regarded as "black-box" models.

To be more confident in the network behavior, the literature provides different test-

ing workflows and properties (such as correctness, robustness, fairness, efficiency) to test a machine learning model.

In this project, we formalize a black-box definition of robustness for regression problems. We use it to test and improve the robustness of a simple neural network: a Multilayer Perceptron. Furthermore, we analyze a different approach to define the uncertainty of predictions using a Bayesian neural network.

1.1 Case study

This work is focused on the robustness analysis of a neural network model to predict the pO2 values of the blood flowing in a probe, based on its fluorescence. The pO2 is the partial pressure of oxygen in the blood, it reflects the amount of oxygen dissolved in the blood and it is measured in millimeter of mercury (mmHg). This metric is important to measure the effectiveness of the lungs in the uptake of oxygen from the atmosphere to spread it in the bloodstream.

This developing system consists of a probe, where the blood flows, and a spotlight to light up the blood. Through the response to the bright pulse, a micro-controller predicts the pO2 at the current temperature and at 37°C. This method allows us to simplify the control of pO2 of a patient, especially during a medical operation.

The input data is the raw curve obtained in response to the bright pulse, an example in Figure 1.1. Each curve can be described through two exponential functions:

$$y_1 = A_1 * \exp^{-B_1}$$

$$y_2 = A_2 * \exp^{-B_2}$$

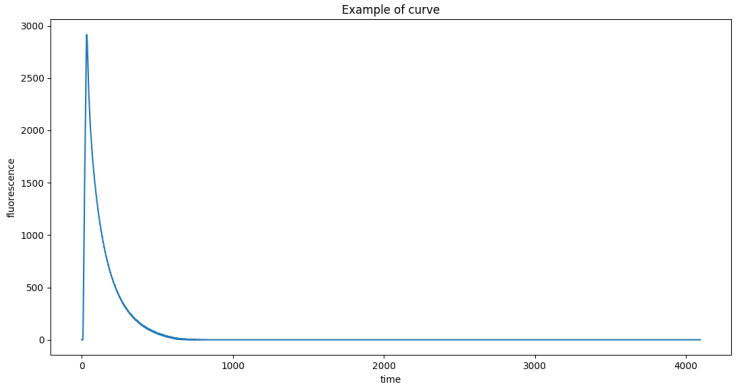


Figure 1.1: Example of curve obtained in response to a spotlight pulse

The prediction of pO2 values has been implemented through a standard neural

network: a multilayer perceptron (MLP). The first approach realized consists in using the four parameters of the curve (A_1, B_1, A_2, B_2) and the blood temperature as features of the multilayer perceptron. However, this method is worse performing than using the means of some sampled values of the curve. The following analysis is based on the best model obtained using the means of some sampled values and the blood temperature as features of a MLP.

1.1.1 Network

The system realized is based on a multilayer perceptron, a class of feed-forward neural networks: a network wherein connections between the nodes do not form cycles. The model is composed of three "Dense" layers (a regular fully connected neural network layer) :

- input layer, composed of 12 neurons;
- hidden layer, composed of 10 neurons;
- output layer, composed of 2 neurons, to predict the current pO2 value and the pO2 at 37°C.

All layers use a sigmoid activation function.

1.1.2 Dataset

The dataset used for this work is composed of 21650 curves showing the blood fluorescence in response to a spotlight pulse.

Each curve is processed in order to get the input, which will be passed through the network. The input is based on the mean of some values, sampled in different points of the curve. This process reduces the curve quantity stored.

In particular, the input data is based on 6 parameters:

- mean of the curve from 50 to 60;
- mean of the curve from 90 to 110;
- mean of the curve from 190 to 210;
- mean of the curve from 340 to 360;
- mean of the curve from 620 to 640;
- the current temperature of the blood.

Below the code implemented for this purpose.

```
def compute_params(curve):
  val_0 = np.mean(curve[50:60])
  val_1 = np.mean(curve[90:110])
  val_2 = np.mean(curve[190:210])
  val_3 = np.mean(curve[340:360])
  val_4 = np.mean(curve[620:640])

  return val_0, val_1, val_2, val_3, val_4
```

Each input vector is related to an output vector, containing the current pO2 and the pO2 at 37°C. The output vector has been determined using two different precision measurement instruments, specialized in the measure of blood pO2. The best model was trained using as true values the estimations provided by the first measurement instrument. The following analyses use as output vectors the same estimations.

In order to speed the neural network training, all input features and all outputs have been pre-processed using the scaling normalization technique.

1.1.3 Metrics

In the state of the art, the main approaches to evaluate the performance of neural networks for regression problems are:

- Mean Squared Error (MSE): it is the average of the squared difference between the target value and the value predicted by the model. Since it squares the residuals, it penalizes even small errors, leading to over-estimation of how bad the model is.
$$MSE = \frac{\sum (y - \hat{y})^2}{n}$$
- Mean Absolute Error (MAE): it is the absolute difference between the target value and the value predicted by the model. This metric is more robust to outliers and does not penalize the errors as extremely as MSE.
$$MAE = \frac{\sum |y - \hat{y}|}{n}$$

Many different methods derive from them. In particular, to measure the goodness of this model, we used the Mean Absolute Percentage Error (Figure 1.2). It is the percentage equivalent of MAE. Its equation is similar to that of MAE with the addition of percentages:

$$MAPE = \frac{100\%}{n} \sum \frac{|y - \hat{y}|}{y}$$

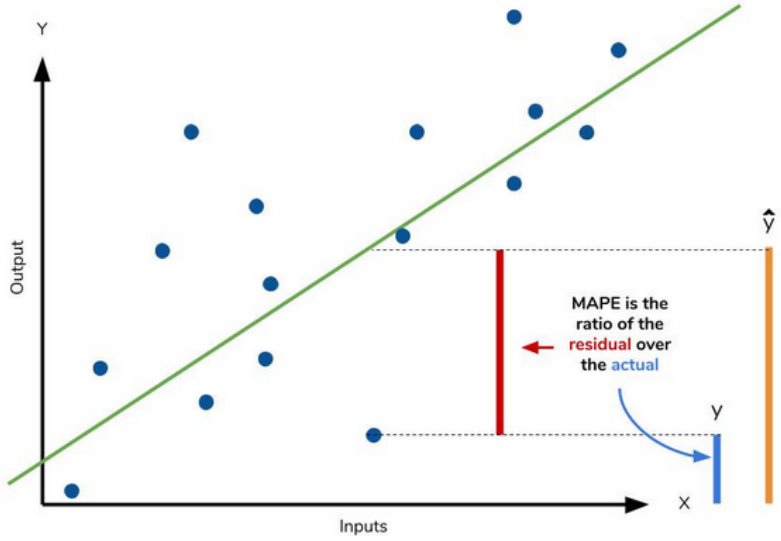


Figure 1.2: Example of Mean Absolute Percentage Error [18]

This metric has some critical issues which cause its uselessness for many domains. The main problems concern:

- the failure of MAPE when some target values are equal to zero, because of the division for zero problem;
- the heavier penalty on small target values: if some target values are very close to zero, the corresponding absolute percentage errors will be really high and therefore they bias the correctness of MAPE.

To overcome these issues, in literature there are some different alternatives, mainly used for time series forecasting, such as Mean Absolute Scaled Error (MASE), Symmetric Mean Absolute Percentage Error (sMAPE) or Mean Directional Accuracy (MDA) [12][17].

However, for this specific case study, we can use the standard MAPE metric since its main issues do not affect this domain. Indeed, pO2 target values will not be equal to zero: in particular, the dataset used for the model training, which represents a well-formed population, has current pO2 values in the range between 39.1 mmHg and 433 mmHg, while pO2 values at 37°C are between 39.1 mmHg and 581 mmHg. Moreover, to overcome the problem of the heavier penalty on small target values, the model

has been trained with a custom loss function which weighs more all samples with low pO2 values, so that the model will make fewer mistakes on these critical targets. More details about this function will be presented in Section 2.1.3. Besides, for this case study, we use another metric informally called "Out of Threshold %" (OT): it is the percentage of the number of predictions out of +/-10 % threshold, so with an absolute percentage error greater than 10%. Indeed, the pO2 prediction can be considered confident only if the difference between the target value and the value predicted by the model is less than +/-10%.

Furthermore, as explained before, the "true" values of pO2 used to evaluate the error of each prediction are the estimations provided by a precision measurement instrument. However, these measures are affected by errors. In fact, using two different measurement instruments we can obtain different measures for the same sample of blood. For this reason, we can define an error lower bound for the model based on the difference between the estimations provided by two precision measurement instruments.

The results provided by the original model on the test dataset (composed of 4330 samples) are the following:

	MAPE [%]	**Lower Bound MAPE [%]**	**OT [%]**	**Lower Bound OT [%]**
current pO2	3.70	2.48	5.17	0.96
pO2 37°C	3.35	1.94	3.60	0.0

Table 1.1: Lower bounds and results generated through original model

Chapter 2

Theoretical background

This chapter provides some basic information about neural networks and in particular about Multilayer Perceptron, which is one of the most typical neural network models to solve both regression and classification tasks. We will present a mathematical explanation of Multilayer Perceptron neural networks and, in order to clarify how this network works, we will describe the training process of these models and how to adapt it to solve a specific task.

2.1 Neural Network and Multilayer Perceptron

An artificial neural network is a type of machine learning model. It is based on a collection of nodes called artificial neurons that are connected and cooperate to compute the output for a specific task. Both its structure and its behavior are inspired by the human brain.

The basic type of artificial neural network was developed in the 1950s and it was composed of a single neuron called "Perceptron".

A perceptron is a basic model of a human neuron. It performs a computation on the input received and its output is computed through an activation function. In particular, for each connection, there is a weight ω, representing the importance of the related input to the output.

The neuron's output is determined by applying an activation function σ on the weighted inputs z, where inputs x are linearly combined with the weights and bias:

$$\sigma(z) = \sigma(\sum \omega_j * x_j + bias)$$

The first perceptron was based on a simple step function: if the weighted sum is less than a threshold value the output is 0, otherwise, the output is 1. This function can

be simplified using the bias instead of the threshold, in this way the threshold becomes a variable to learn like all the other weights:

$$if(\omega * x + bias < 0) : output = 0$$

$$if(\omega * x + bias > 0) : output = 1$$

However, the output computed by this activation function is binary, therefore it can be used only to solve simple problems. For this reason, many different functions have been developed. Nowadays, neural network models use non-linear activation functions. These functions allow solving complex tasks, learning and modeling complex data such as images, video, audio and high dimensional dataset. The most common activation functions are sigmoid function, tanh function, RELU function.

The activation function used in this case study is the sigmoid activation function (Figure 2.1):

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$

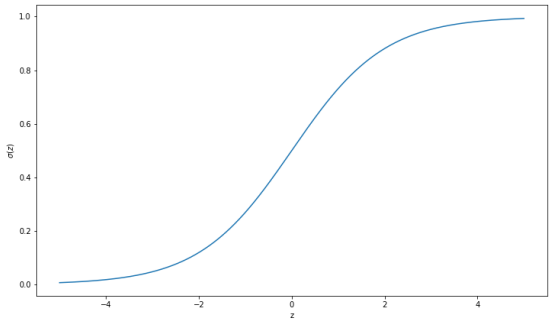


Figure 2.1: Sigmoid function

The main advantages of this function concern the smooth gradient, which prevents fast-changing in output values and speeds the algorithm convergence, and the normalization of the output, its values are bound between 0 and 1. The main disadvantages are the vanishing gradient problem and its computational cost.

A single neuron cannot recognize complicated patterns on its own, for this reason, a neural network is composed of many neurons grouped into layers and connected to each other to handle even complex inputs. In particular, each neuron is connected to another layers' neuron through a weighted connection. A network can have multiple layers which can be classified into three categories:

- Input layer: the first layer, it takes input from the dataset, therefore it is the exposed part of the network for this reason it is also called the visible layer;

- Hidden layer: it is a layer after the input one, it is not directly connected to input data. A neural network can contain many hidden layers;

- Output layer: the final layer is called the output layer. It is responsible for computing the output values required by the problem. The choice of structure and activation function for this layer is strongly constrained to the type of problem. For example, regression problems may have a single output neuron, binary classification can be implemented using a single neuron with sigmoid activation function, while multi-class classification problems may have many neurons in this layer [5].

The most typical artificial neural network is the Feed-forward Neural Network. It is composed of artificial neurons organized in layers, where each neuron in a layer is connected with all the neurons in the previous layer. The input data pass through the network, layer by layer until it reaches the output. During ordinary operation, when it acts as a prediction system, there is no feedback between layers: the flow of information takes place only in the forward direction.

The main problem of this type of neural network is the parameters explosion: each neuron in a layer is linked to all the neurons of the following layer. The number of parameters increases by adding several hidden layers, for example, given two layers i and $i + 1$ with respectively n_i and n_{i+1} number of neurons, the layer i has the number of parameters equal to $n_i * n_{i+1} + n_i$. To overcome this problem, there are specific models used to solve different tasks, based on the generic Feed-forward Neural Network: Convolutional Neural Network, Recursive Neural Network, Multilayer Perceptron.

Multilayer Perceptron is a class of feed-forward neural networks, it can be considered as a subset of deep neural networks. In literature, there are different definitions of Multilayer Perceptron. The most strict definition represents Multilayer Perceptron as a three-layer network: composed of the input layer, a single hidden layer and the output layer; in Figure 2.2 an example of this network. However, in many other definitions, networks with more than one hidden layers are considered Multilayer Perceptron as well.

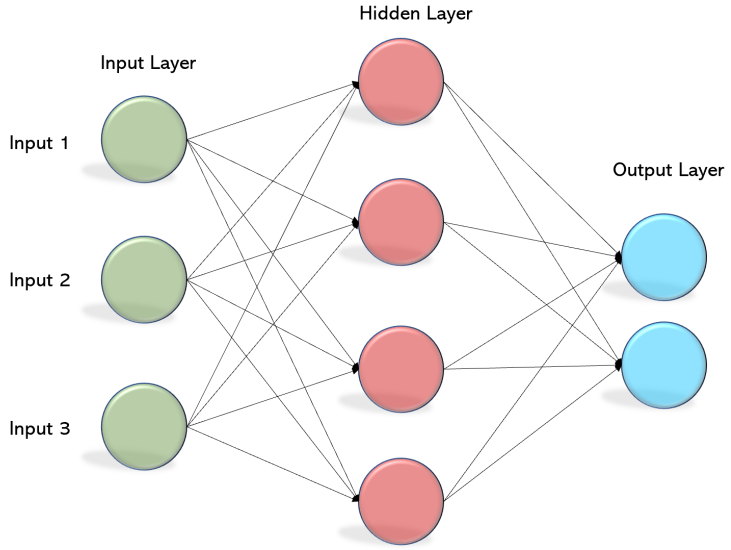


Figure 2.2: Example of Multilayer Perceptron

2.1.1 Data pre-processing

Data pre-processing aims to facilitate the training process by transforming and scaling the entire dataset appropriately. Pre-processing is an important procedure before training the machine learning models, in order to remove outliers and to scale the features to an equivalent range. There are two main types of pre-processing: scaling/normalization and standardization methods. In both cases, the values of input features are transformed and they acquire specific helpful properties.

Feature scaling (also known as normalization) is used to standardize the range of input features. Indeed, the range of input data may vary widely, therefore a feature could influence the output more than the others due to its larger values even though it is not more important for the prediction. The normalization technique brings all features to the same range, between 0 and 1. The normalization formula, also called min-max normalization is:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

You can use this technique when the distribution of your data is unknown or it is not Gaussian and it is useful, especially, when data have varying scales.

Standardization is another scaling technique where the features are centered around the mean with a standard deviation of 1. Its formula is:

$$x' = \frac{x - \mu}{\sigma}$$

This method assumes that data have a Gaussian distribution. It is useful when data

have varying scales and the algorithm used makes assumptions about data having a Gaussian distribution [13].

For this case study, we pre-processed our data using the min-max normalization, indeed the features have different scales, but their distributions are not Gaussian.

2.1.2 Training

The advantage of machine learning models is to learn from data. This means that data-scientists do not have to define explicitly the model and its behavior. Neural networks learn their parameters during the training phase, which is one of the main steps in machine learning modeling since it allows us to determine the behavior of the model. In this phase, the algorithm adjusts the network parameters comparing the predicted output of a set of samples with their target values. Based on the training data, there are three different types of training:

- Supervised learning: a type of machine learning where training data are provided with labels as targets. It is the most widely used type of machine learning especially to work on regression and classification problems. It maps the labeled inputs to the known outputs;

- Unsupervised learning: a learning method based on training data without labels. It allows us to understand patterns and discover outputs, therefore it is used for clustering and association tasks;

- Reinforcement learning: based on data that are in the form of sequences of actions, observations, and rewards. The model learns how to take actions to interact in a specific environment to maximize the specified rewards.

In particular, the Multilayer Perceptron training phase is based on a supervised learning technique called back-propagation. This method works by approximating the relationship between the input data and the output, by adjusting the weights in order to minimize a loss function. This function, usually called cost function, is non-linear and it calculates the distance between the predicted output and the target value. There are many different loss functions, the choice of the best loss function depends on the characteristics of the task to solve.

The back-propagation algorithm involves two phases:

- Forward phase: in this phase all network parameters are fixed and the input vector is propagated, layer by layer, through the network. We calculate the output of the current model in response to a specific input: the prediction of the model.

Using the labeled data we can compute the error, the difference between the predicted output and the target value, evaluated through a specific loss function;

- Backward phase: the loss is propagated through the network in the backward direction in order to determine for each node its contribution to the overall error. The error of each node is used to update network parameters. The weights are updated based on the gradient computed on the loss function. Indeed, the gradient shows how much each parameter needs to change in order to minimize the loss function [14].

In Figure 2.3 a basic example of the back-propagation process.

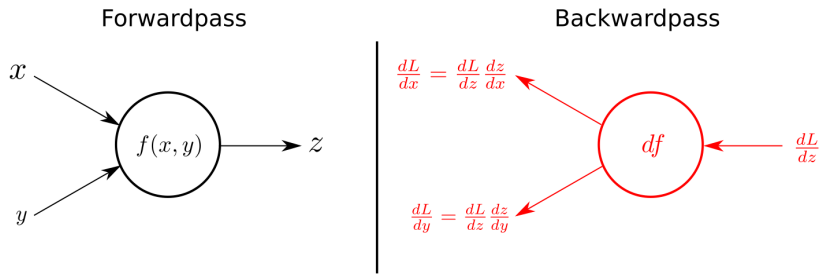


Figure 2.3: Example of back-propagation

The training phase is an iterative process divided into epochs: a series of steps where all the training dataset is used. Each iteration performs the back-propagation algorithm which can be implemented in two basic ways:

- Sequential mode: the network parameters are updated on a sample-by-sample basis. It is used mainly for classification tasks;

- Batch mode: the network parameters are updated on a batch-by-batch basis, where each batch consists of a subset of data sampled from the training dataset. The batch size is really important since it affects the training speed and the quality of the final model. It is best suited for regression problems.

The choice of loss function and optimizer, which define how to use the gradient in order to update the weights and biases, is really important to obtain optimum and faster results. Another important aspect of the training phase is the dataset. For the state of the art of machine learning, the dataset has to be divided into three different sets: training, validation and test set. These sets should be independent of each other and they should have the same properties. In general, the training set is bigger so that the model has more information to determine its behavior. Indeed, the training set is used during the training phase to update the network weights, while the validation set

evaluates the ability of the network to generalize the problem. Instead, the test set allows us to compute the metrics of the final model. A summary of the training phase can be seen in Figure 2.4.

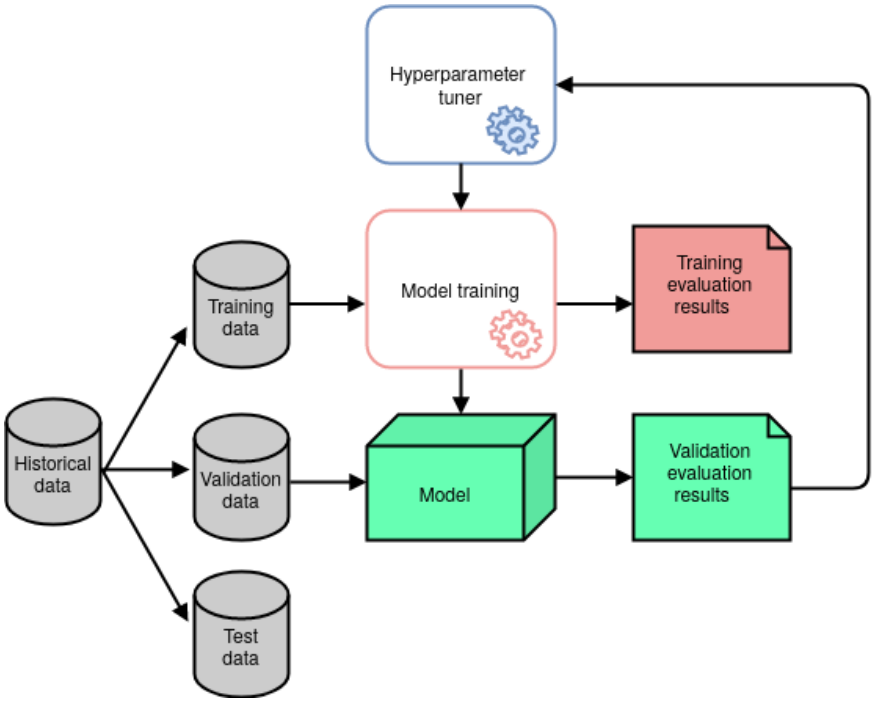


Figure 2.4: Example of the training phase [28]

In this case study, the original dataset has been divided into these sets with the following proportions: 60% training set, 20% validation set and 20% test set.

2.1.3 Loss, Optimizer and Hyperparameters

In order to get optimum and faster results, it is important to choose properly some functions and parameters, such as loss function, optimizer, batch size, epoch, learning rate [3].

Usually, the error is computed as the difference between the target value and the predicted output. The function used to calculate this error is known as "Loss function". There are different loss functions, which provide different errors for the same prediction influencing the performance of the model. Based on the type of task we can choose between different loss functions:

- Regression loss functions: for regression problems, wherein the target variable is continuous. The most widely used are Mean Squared Error and Mean Absolute Error.

- Classification loss functions: when the output variables are probability values related to a specific classification. They represent the confidence of the prediction. The most used classification algorithms are Binary Cross-Entropy, Negative Log-Likelihood, Margin Classifier and Soft Margin Classifier;

- Embedding loss functions: used for problems where we have to measure whether two inputs are similar or dissimilar. Some algorithms are: L1 Hinge Error and Cosine Error.

For this case study we used a modified version of Mean Absolute Error, in fact, we are dealing with a regression problem. Our loss function, which can be called "Weighted MAE", weighs more all samples with low pO2 values, since they suffer from a higher percentage error.

Another important choice to make in order to improve the performance of our model is the optimizer. Optimizers are algorithms used to update the weights and biases of the model to reduce the error. These algorithms can be divided into two categories:

- Constant Learning Rate Algorithms: in this case, we have to choose the learning rate η used to update weights and biases.

$$\omega^{k+1} = \omega^k + \eta * \Delta J(\omega)$$

  The choice of this hyperparameter can be difficult: a learning rate too small leads to slow convergence, affecting the overall training time which gets too large; instead a learning rate too large can prevent the convergence causing the loss function to fluctuate around the minimum or even to diverge. One of the most used optimizers is Stochastic Gradient Descent, which derives from the gradient descent algorithm.

- Adaptive Learning Algorithms: the main challenge of using gradient descent is the choice of its learning rate which depends both on the model and on the problem. Furthermore, in gradient descent, the same hyperparameter is applied to all weight updates. Adaptive gradient descent algorithms provide an alternative to classical gradient descent. They include learning rate methods, which provide a heuristic approach to determine the optimum learning rate for each parameter which is adjusted over time. The most used algorithms are Adagrad, Adadelta, RMSprop, Adam [11].

For this case study, we use Adam optimizer. This optimizer has the benefits of both AdaGrad and RMSProp. Instead of adapting the learning rates based on the average

first moment as in RMSProp, Adam also uses the average of the second moments of the gradients. It is a popular algorithm in deep learning networks because it achieves good results fast.

Other important hyperparameters to choose in order to improve the computational time and the convergence are:

- Epochs: iterations of the training phase. In order to get an appropriate model and avoid overfitting, we trained the model for around 3000 epochs.
- Batch size: the size of the training dataset subset used in each training iteration to adjust weights and biases. In this case study, we set 1024 steps per epoch, which is equivalent to a batch size of almost 12.

To improve the model and especially to avoid the overfitting problem it could be possible to use some regularization techniques, such as the dropout method where randomly selected neurons are ignored during the training phase, so they do not contribute to the updating of weights. The network becomes less sensitive to the specific weights of neurons and it has more capability of generalization.

Chapter 3

Robustness

This chapter presents the main approaches in machine learning testing. Since this work is focused on the robustness analysis of a multilayer perceptron, later in this chapter we present a novel approach to quantify the robustness of this machine learning model. This property is really important in systems used for medical treatments where the correctness of the prediction can be decisive.

3.1 Machine Learning Testing

Software testing aims to detect the differences between current and required behavior. Testing is an effective way to detect problems and to improve the reliability of machine learning systems [27].

In general, a software bug is an imperfection in a computer program that causes a discordance between the current and the required behaviors. In machine learning, we can refer to the term bug in a similar way: it is any imperfection in a machine learning system that causes a difference between the existing and the required conditions. Machine learning testing consists of activities aimed to reveal machine learning bugs. Each machine learning system may have different types of required conditions: correctness, robustness, security, fairness and privacy.

Machine learning testing can be executed in two modes: offline and online testing. Offline testing represents all activities performed after the training of a model through historical data, but before its deployment. For example, cross-validation procedure which ensures the achievement of required properties. Differently, online testing includes analyses to evaluate the model response to user inputs in real-time. Online testing can be really important for many reasons: offline testing is based on test data, therefore it takes into account only a small dataset that does not represent completely future data. In addition, it can be complicated to test some problematic situations that

could happen in reality. In Figure 3.1 a schema of the two different executions of machine learning testing.

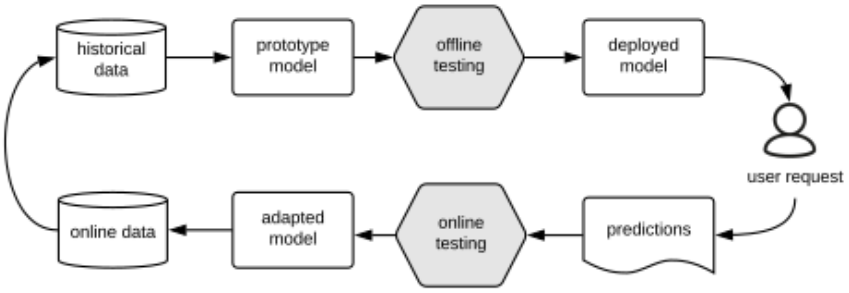


Figure 3.1: Machine Learning Testing in machine learning system development [27]

Furthermore, machine learning testing includes different methods to test both the model, training and test datasets. In particular, to obtain a model with high performances it is important to use an appropriate dataset for the training phase. Therefore, it is important to check if the dataset is correctly generated to achieve the required behavior. The main tests related to the dataset include checking the correctness of labels/target values, checking the completeness of the dataset (the dataset has to be representative of the population and its size has to be appropriate, indeed small dataset can generate only poor models) and checking the correct data pre-processing and the well-formed dataset splitting into training, validation and test set. In addition to dataset tests, there are tests to verify the correctness of the software implemented, allowing to test all the code that developers write to implement, deploy or configure the algorithm. However, the most interesting test class is the functionality test. It includes all methods to evaluate the behavior of the network checking that the model guarantees some properties. The main properties defined by the state of the art are correctness, robustness, privacy, security, efficiency and fairness.

For this work, we focus on offline testing in particular to measure the robustness property. We define a specific framework to evaluate the robustness of our model and to improve it as well.

3.2 Robustness

In software engineering, robustness is defined as “The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions” [1] [21].

For machine learning, robustness can be defined in a similar way. However, in order to properly specify robustness, it is important to define another machine learning

testing property: Correctness.

Correctness measures the probability that the machine learning system predicts output accurately. Let x be a sample in the distribution D of future unknown data and let h be the machine learning model to test. $h(x)$ is the predicted output of x, while $c(x)$ is the target value. In general, the model correctness can be defined as the probability that $h(x)$ and $c(x)$ are identical:

$$E(h) = Pr[h(x) = c(x)]$$

This property is a fundamental requirement of a machine learning system. Since future data are usually not available, correctness is evaluated by splitting the data in training, validation and test set and using the test set as future data. For regression problems, where the equality between the predicted value and the target value is improbable, there are different metrics to evaluate correctness, which measures the error between predictions and targets. A model with high correctness has a low error.

Robustness can be defined using the previous definition of correctness. Let S be the machine learning system and $E(S)$ be its correctness, while $\delta(S)$ is the machine learning system with perturbations. The robustness of a machine learning system is the difference between $E(S)$ and $E(\delta(S))$:

$$r = E(S) - E(\delta(S))$$

Therefore, state of the art defines robustness as the property to maintain the resilience of a machine learning system towards perturbations: the behavior of the machine learning system remains close to the nominal one even with perturbed inputs [27].

As defined by the state of the art, the robustness definition depends on the correctness of the model. Since this case study is a regression problem, we cannot measure the correctness as a probability as defined in the literature, which could be done in classification problem using accuracy, precision and recall metrics. For this reason, to measure the correctness we will use the same metrics used to evaluate the model, described in Section 1.1.3: Mean Absolute Percentage Error.

We represent each test with a graph showing the variation of MAPE with the increase of a specific alteration/perturbation of the input data. Each alteration is associated with an interval that represents the area under analysis for that perturbation. In general, it is not possible to maintain a behavior close to the nominal one for each perturbation, where for nominal behavior we consider the results obtained with the original dataset without any perturbation. However, since predictions can be considered confident when the MAPE is less than 10%, we want to maintain the MAPE error

below the reference line of 10% which represents the acceptable bound. To determine the robustness we compute the area included between the MAPE limit and the alteration graph. Therefore, we get a useful value that can be compared with the ideal situation, where the MAPE is equal to the nominal value for each grade of alteration, but also with other network models. Using this basic approach, the area is calculated through a definite integral of the MAPE values in the interval specified:

$$\int_{n}^{m} max(10\% - \textbf{\textit{MAPE}}(x), 0) * dx$$

Where m represents the maximum alteration and n the nominal case.

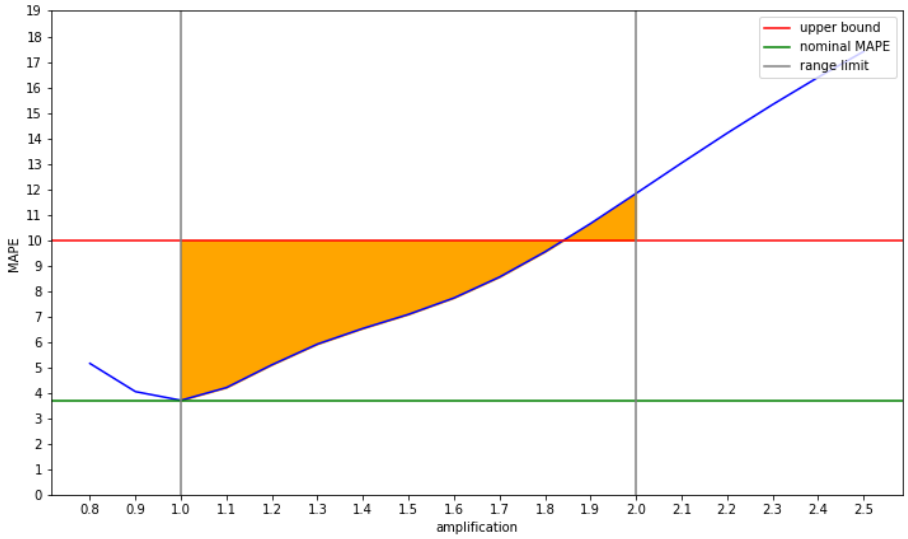


Figure 3.2: Example of the significant area to evaluate the robustness (orange area)

In Figure 3.2, you can see an example of an alteration graph. In this case, the perturbation is the amplification of the raw curve used to compute the network input parameters. The range used to evaluate the robustness is between 1 and 2, as shown by the grey lines; the blue line represents the alteration curve for current pO2 MAPE, while the green one is the ideal curve. The orange area represents the significant area used in order to evaluate the robustness for this perturbation.

Robustness can be computed also through other functions or weighing the difference between the acceptable limit and the MAPE for each alteration grade. Indeed, in many domains, alterations close to the nominal value are more probable than the others, therefore we compute the robustness considering the probability associated with each alteration. For this purpose, we define a distribution that favors values close to the nominal case and we weigh each MAPE value with its probability. The area is

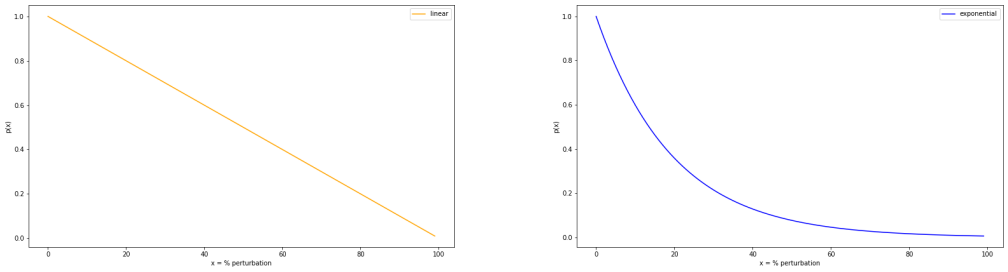
computed as follow:

$$\int_n^m max(10\% - \textit{MAPE}(x), 0) * p(x)\, dx$$

Where $p(x)$ is the probability associated with a specific alteration. Since we do not have the correct distribution of the alterations we used two basic probability distributions:

- Linear: the slope is automatically defined from the data.
  In this case $p(x) = \frac{m-x}{m}$. In Figure 3.3a there is an example of the linear distribution.

- Exponential: the slope is defined by a parameter α to choose based on the importance of the alteration grades. In this case: $p(x) = \alpha^x$. In Figure 3.3b you can see an example of an exponential distribution.



(a) Linear distribution (b) Exponential distribution

Figure 3.3: Example of distributions to weigh the error in the robustness evaluation

In order to evaluate the robustness, we use the Heaviside Step function as well. The Heaviside step function $H(y)$, also called the unit step function, is a discontinuous function, whose output value is zero for negative arguments $y < 0$ and one for positive arguments $y > 0$, as shown in Figure 3.4 [2].
In this case, the area is calculated with the following formula:

$$\int_n^m H(10\% - \textit{MAPE}(x)) * dx$$

where H is the Heaviside Step function, while its argument is the difference between the MAPE upper bound (10%) and the MAPE for a specific alteration grade.

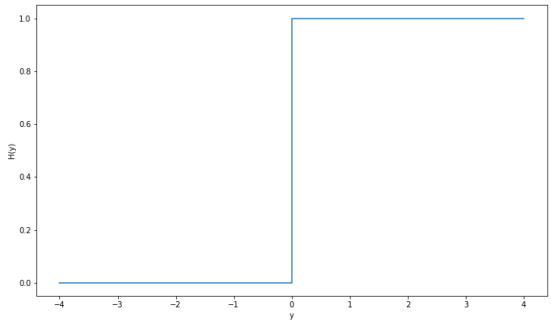


Figure 3.4: Heaviside Step function

The Heaviside Step function considers each alteration, which satisfies the acceptable limit of 10%, in the same way without distinguishing how close to the ideal situation (nominal results) it is. Indeed, the output for all positive arguments is equal to one. Therefore, an alteration which causes a MAPE near 10% and another one with MAPE close to the nominal value are considered equivalent: both are counted in the same way for the calculus of the area since the difference with the upper bound is positive.

After having computed the area for a type of alteration, we can determine the robustness of the network on that specific perturbation by it comparing with the area obtained in the ideal situation, where for each grade of alteration the MAPE is equal to the nominal value. Therefore, we can compute the robustness percentage for each type of perturbation with the following formula:

$$Robustness_{\%} = \frac{area}{ideal\_area} * 100$$

This method allows comparing the network robustness on different perturbations. In addition, it is possible to summarize with a single value the global robustness on all perturbations by computing the mean of all single robustness values.

3.3 How to choose the right robustness function

Based on the function used to compute the area we obtain different robustness results. In general, there is not an explicit rule to choose the right function for the robustness measurement, suitable for any case study. However, based on the specific task to evaluate, we could determine functions or probability distributions that better satisfy the required conditions. Indeed, for tasks where the only important requirement is the respect of a condition, such as an upper bound, the robustness can be computed using the Heaviside Step function, or another function with similar behavior. Instead, for

problems where all grades of perturbations have the same importance, but it is also important to consider the difference between the metrics computed on altered data and on the original dataset, the best solution is to evaluate the robustness with the basic approach. In this case, the error between altered and nominal metrics is included directly in the formula. Differently, when the alterations have different importance based on their grade, the best practice consists of weighing the difference between metrics in altered and nominal conditions through a probability distribution. For example, in many applications small alterations are more probable and therefore we can assign a higher weight for these perturbations. The choice of the probability distribution depends on how fast the probability decreases to zero when the alteration grade increases; in general the most common probability distributions are the linear (Figure 3.3a) and the exponential distribution (Figure 3.3b).

In Chapter 4, we will use this definition to evaluate the robustness of the multilayer perceptron model trained for this case study. In particular, we will compute the robustness using the four approaches presented in this chapter (basic approach with MAPE error, weighing MAPE with both linear and exponential distribution and the Heaviside Step function), in order to compare the different results.

Chapter 4

Robustness evaluation of MLP

This chapter describes a testing framework to evaluate the robustness of machine learning models. For health-care systems, evaluating the model safety is an important task since we want to maintain a level of confidence in the machine learning model predictions even with perturbed input data. For this reason, we implement a framework to analyze the model behavior outside nominal conditions measuring its robustness on predictions computed from perturbed data. We defined tests to reproduce all possible situations which perturb the input data and could cause an anomalous behavior of the machine learning system.

4.1 Test Framework

The first step to evaluate the robustness of a model consists of generating some test inputs: samples never seen before by the network, useful to check the correctness of the model predictions on different possible data. The state of the art classifies test inputs of machine learning testing into two categories: adversarial inputs and natural inputs. Adversarial inputs are perturbed input data based on the original inputs and model as well. They may not belong to normal data distribution and they are designed in order to cause the model to make a mistake, therefore they are useful to evaluate the robustness on attacks. Natural inputs, instead, are inputs that belong to the data distribution of a practical application scenario. They are created in order to test the robustness of the model on different kinds of inputs which could occur in practical situations. In this test framework, we evaluate the robustness using natural inputs. This framework aims to generate some samples by altering the original dataset with some perturbations which could occur in real-world settings. In general, it is complicated to reproduce accurately some specific conditions in the reality, it needs a high quantity of time and costs. A more common solution consists of reproducing artificially these conditions

and using them to test and improve the model. The test framework implemented can be considered as a black-box approach: it is not based on the network knowledge, but only on the generation of some perturbed input data by applying some alterations. These altered data are then used to compute the network robustness. This framework can be summarized in a list of steps:

1. Define the robustness function used to evaluate this property;
2. Define the metrics and the benchmarks to use;
3. Analyze the domain of the case study, determine all the alterations, which could occur in the real world, and select the most important to reproduce;
4. Determine for each alteration the appropriate range of modification;
5. Compute all metrics on the new dataset obtained for each alteration;
6. Analyze the results and compute the robustness metric to highlight critical situations.

To test the robustness of our MLP neural network we use the robustness functions described in Chapter 3: basic MAPE error, weighted MAPE with linear distribution, weighted MAPE with exponential distribution and Heaviside Step function. Therefore, the robustness is determined by computing the MAPE on altered data and considering as nominal behavior the MAPE obtained using the original test set. To generate the altered dataset we used the complete dataset. The choice of all possible alterations is an important step since it is important to determine significant perturbations that could occur in the real world; this is the first condition to get consistent results. In our case, we determined five alterations that could happen on the raw curve used to compute the input parameters: the cut of the curve end, caused by an out of memory situation, clock offset, the cut of the peak due to saturation, amplification and attenuation of the curve. The choice of these alterations is the result of the domain analysis which will be presented in Section 4.2. After that, we defined the range for each alteration. This is a complicated task since it is important to select a range capable to generate all critical and possible conditions for each alteration. The alterations chosen are then applied to the dataset in order to generate new input data. Applying the altered dataset to our model, we can compute all the metrics needed to compute the robustness of the network. Therefore, we compared the results on the perturbed input dataset with the nominal behavior. The robustness obtained highlights some critical situations and some weakness of the model.

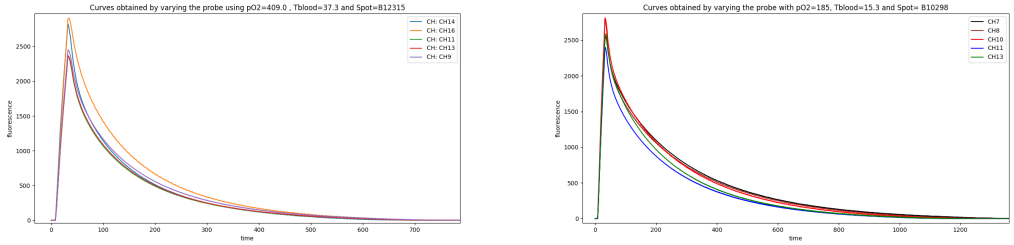
4.2 Domain Analysis

After having defined the robustness function and the metrics to compute it, we analyze the case study domain in order to determine all the possible alterations. As seen in Chapter 1, input parameters are computed from the curve showing the blood fluorescence in response to a spotlight bright pulse.

The aim of this analysis is to determine all perturbations which could be applied to the curve without changing the pO2 associated with it. Because of the different probe and/or spotlight used, the curves obtained from this system vary on the amplitude and on the response time, even when they represent the same pO2 values. In particular, the training dataset contains samples from 16 different types of probe and 178 different spotlights.

In order to determine proper alterations, perturbations that do not affect the pO2 values, we compare the curves' shapes by varying one at the time the spotlight, the probe and the pO2 values, while the other features are fixed.

The first analysis consists of visualizing the behavior of the curve by varying the probe, while pO2 values, blood temperature and spotlight are fixed. In particular, we perform this analysis in two cases: choosing the pO2 values, blood temperature and spotlight more frequent in the dataset, in order to have more data to compare (results in Figure 4.1a), and choosing the mean values of those features (results in Figure 4.1b).



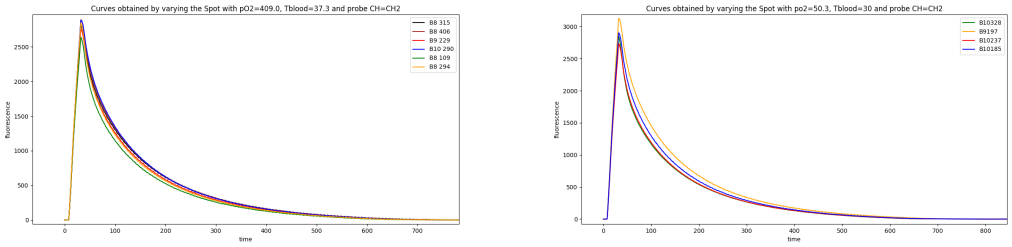
(a) (b)

Figure 4.1: Curves behavior by varying the probe

Observing Figure 4.1, you can see that the probe variation causes amplitude variation and a less significant change in the response time.

The second approach consists of analyzing the curve behavior due to spotlight variation, while pO2 values, blood temperature and probe are fixed. We are using the most common types/values and mean values for the features as before. Different spotlights cause a different amplitude, but this variation is less significant than the amplitude change obtained in the previous analysis. Instead, the change in the response time is not really significant as in the probe variation. These results can be observed in

Figure 4.2.



(a) (b)

Figure 4.2: Curves behavior by varying the spot

A similar test is computed to determine the curve changes when varying the pO2 values associated with them. As before the blood temperature, spotlight and probe are fixed. The results, available in Figure 4.3, show that we can obtain a faster time of response with higher pO2 values, while it decreases with lower ones.

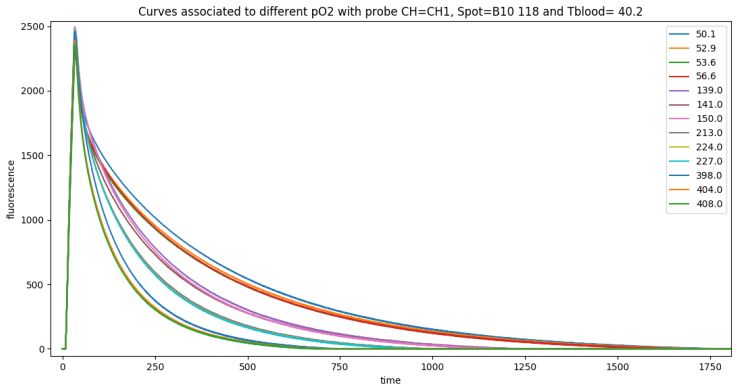


Figure 4.3: Curves behavior by varying the pO2 values

Another analysis consists of comparing the shape of three curves: two curves associated with some pO2 values and the third one associated approximately to the average pO2 values of the previous ones. This operation aims to determine a linear relationship between curve shapes and their pO2 values.

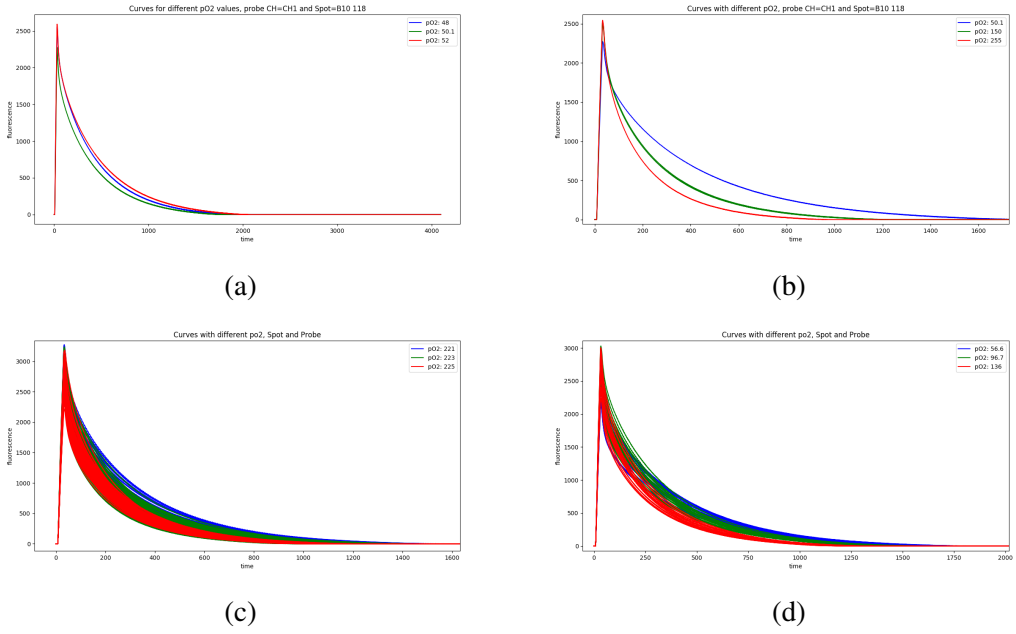


Figure 4.4: Analysis of curves with different pO2 values

However, Figure 4.4 shows that there is not a linear relationship between curve shape and pO2 values. Therefore, we cannot generate new samples data by computing the mean of two curves with different pO2 values, since it is not possible to estimate the correct pO2 target values. Furthermore, this last analysis exhibits that predictions of curves related to similar pO2 values are more complicated: in this case curves are approximately the same even though they represent different pO2 values. In addition, the plot of curves with different spotlight and probe highlights the complexity of this prediction problem. Indeed, as you can see in figures 4.4c and 4.4d, there could be overlapping of curves with different pO2 values.

Domain analysis is an essential phase to determine the correct alterations to perform. For this case study, this procedure highlights how the amplitude of each curve depends on the probes and spotlights used, while the changes in the response time derive from the different pO2 values associated with the curve. Furthermore, we conclude that we cannot generate artificial data by averaging curves with different pO2 values, since it is not possible to estimate their labels, the correct pO2 values that the network should predict. Based on the domain knowledge achieved with these analyses we can define all alterations to apply in order to generate proper artificial samples.

4.3 Alteration tests

Using the knowledge achieved through the domain analysis we define five different perturbations to apply on the complete dataset in order to evaluate the model robust-

ness: cut of the curve end, clock offset, cut of the peak, amplification and attenuation of the curve. For each alteration we choose an appropriate range and we compute the robustness using the four functions defined in Section 3.2. In particular, the robustness evaluation for each alteration depends on the following algorithm.

```
Input
curve_raw : curve obtained in response to the bright pulse
targets : true values of pO2
model : model trained to predict pO2 values
perturbation_range : range to apply the chosen perturbation

Output
robustness_result : robustness value for a specific function

Procedure
mape = array(size(perturbation_range))
ideal_mape = array(size(perturbation_range))
// For each grade of perturbation
for p in range(perturbation_range):
  // Apply the alteration to the original curves
  curve_new = alteration_method(curve_raw, p)
  // Compute the parameters from each curve
  params = compute_params(curve_new)
  // Normalization (min-max scaling)
  input_data = normalize(params)
  // Compute predictions on new dataset
  predictions = denormalize(model.predict(input_data))
  // Compute errors for the current perturbation grade and
     store it
  mape_p = mean_absolute_percentage_error(predictions,
     targets)
  mape.add(mape_p)

// Create the ideal_mape array, composed of the nominal
   metrics for each perturbation grade
ideal_mape = insert_in_each_array_index(mape[0])
// Compute robustness
robustness_function(ideal_mape, mape)
```

4.3.1 Cut of the curve end

This alteration consists of "cutting" the end of the curve obtained in response to the bright pulse. The idea is that in the real world because of some disruptions or failures or anomalous system behaviors in general only a portion of the curve could be memorized.

The parameters used as input of the model, except for the blood temperature, coincide with the mean of some values of the curve; in particular, the last parameter is computed by averaging the values from 620 to 640. Therefore, the cut of curve values, that follow the 640^{th} instant, does not affect the network robustness since these values are not considered into the computation of parameters. For this reason, the range used for this alteration starts from the 640^{th} instant, considering the following values canceled as well. Furthermore, the end of the range is the 630^{th} instant, indeed a more extended cut of the end of the curve could completely modify or even nullify the last parameter, which becomes useless. Without a parameter, the network cannot perform properly for this reason those cases cannot be considered as new data input. In conclusion, this alteration is applied in the range between 641 (original input vector) and 620 (canceled only half the values to compute the last parameter) with a step of 1.

Below the code implemented to perform this alteration.

```
1 import numpy as np
2
3 def curve_end_cut(curve_raw, start):
4   curve_new = np.copy(curve_raw)
5   curve_new[:,start:]=0.0
6   return curve_new
```

An example of perturbed data can be visualized in Figure 4.5.

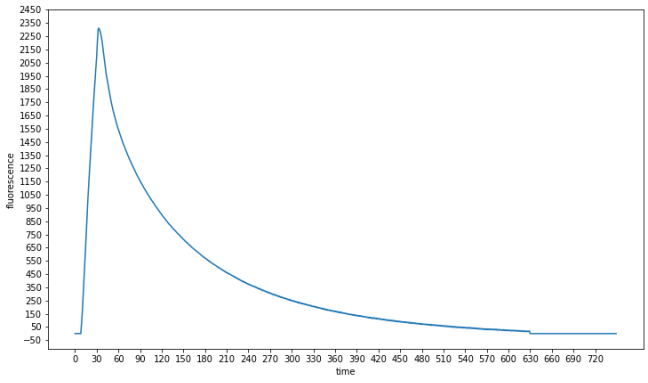
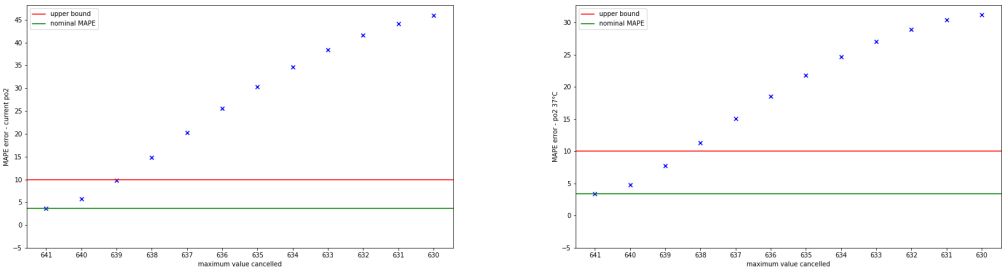


Figure 4.5: Example of the cut of a curve in the range between 640 and 630 instants

Figure 4.6 shows the results obtained using the perturbed data to predict both current pO2 and pO2 at 37°C. You can see that the model is sensitive to the cut of the curve end, indeed the cancellation of only two instants (640^{th} and 639^{th}) for current pO2 and only three instants for pO2 at 37°C causes respectively the achievement and the overtaking of the upper bound of 10%.



(a) (b)

Figure 4.6: MAPE graphs for the cut alteration

4.3.2 Clock offset

The clock offset alteration aims to represent the difference of time calculation in different machines/systems; in particular between the microprocessor, which will process the curve obtained in response to the pulse, and the signal time. A clock offset can happen when the microprocessor is not correctly set up or there is a delay in the signal generation compared to expectations. In these cases, each curve will be shifted in time and so all parameters will result in different values. Each parameter is computed by averaging some values and each range has a maximum size of 20 values. For this reason, we generate altered input data using a range between 0 (original input vector, without clock offset) and 30, with step 1. Indeed a clock offset of 30 changes all values used to compute network parameters, representing a critical situation. Below the code implemented to execute this alteration.

```
1 import numpy as np
2
3 def clock_offset(curve_raw, offset):
4   curve_new = np.copy(curve_raw)
5   end = curve_new.shape[1]
6   for i in range(end-offset):
7     curve_new[:,end-1-i]=curve_new[:,end-1-i-offset]
8   curve_new[:,:offset]=0
9   return curve_new
```

An example of data with clock offset can be visualized in Figure 4.7.

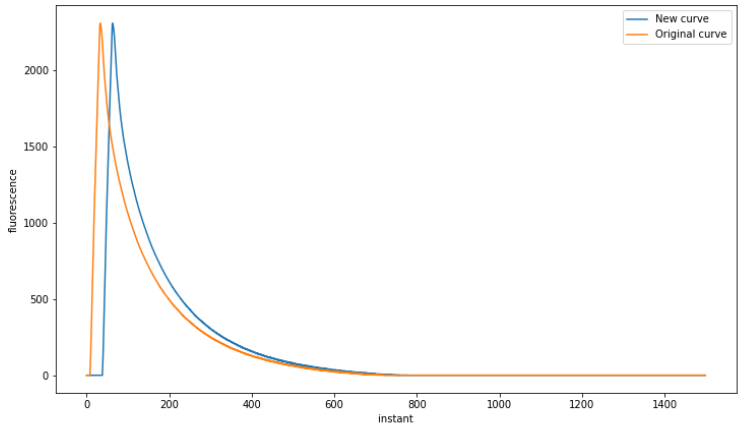
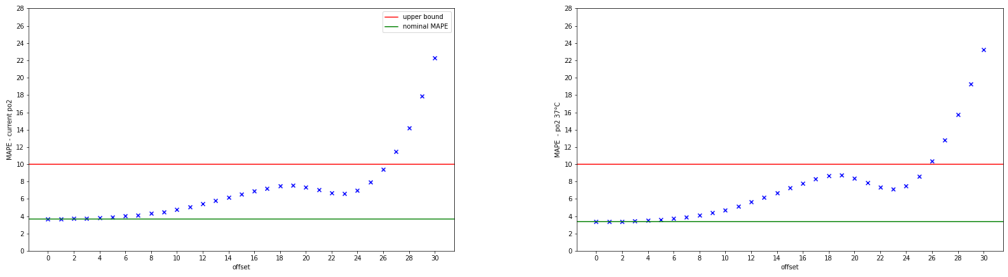


Figure 4.7: Example of a clock offset of 30 instants

Figure 4.8 shows the results obtained using the perturbed data to predict both current pO2 and pO2 at 37°C. You can see that the model is robust to clock offset. In particular applying clock offset up to 10 instants we obtain results equal to the nominal values or with small changes. With higher clock offset, MAPE values increase rapidly, but they remain below the upper bound up to 26 instants.



(a) (b)

Figure 4.8: MAPE graphs for clock offset alteration

4.3.3 Cut of the peak

This alteration consists of "cutting" the peak of the curve to perform saturation events. Saturation is a common phenomenon in electronics where a signal cannot exceed a specific range of values. This situation is really critical since some values of the curve are set equal to a threshold. It could happen when some anomalies occur in the electronic system realized.

In Figure 4.9, you can see a summary of the basic statistical properties of input network parameters.

	TBlood	v0	v1	v2	v3	v4
count	21650.000000	21650.000000	21650.000000	21650.000000	21650.000000	21650.000000
mean	30.520456	1903.909058	1434.737671	942.148621	553.844177	222.749100
std	8.383431	227.108337	235.428589	243.699173	215.425018	138.093033
min	13.600000	647.000000	390.899994	251.649994	107.849998	0.000000
25%	24.299999	1745.624969	1272.099976	778.712509	387.299988	104.962498
50%	32.299999	1878.750000	1420.625000	948.150024	567.324982	223.325005
75%	37.299999	2040.500000	1583.824982	1103.949951	704.650024	323.337502
max	41.099998	2848.100098	2363.600098	1824.099976	1319.699951	744.200012

Figure 4.9: Summary of statistics for input parameters

Based on this information we determined the most appropriate range for this alteration. Since the maximum amplitude of the curves in the available dataset is 4040, we generate altered data executing peak cut from the threshold of 4000. In addition, parameters $v0$ and $v1$ have a mean value higher than 1300 and parameters $v2$ and $v3$ could have a value greater than the 1300 threshold as well. For this reason, we considered this level as the maximum alteration grade. Indeed, a lower level will change drastically the curve, at least half the parameters will change, changing completely the input of the network compared to the original dataset. Therefore, the cut of the peak alteration has been performed in the range between 4000 and 1300 with the step of 10.

Below the code implemented to perform this alteration.

```
1 import numpy as np
2
3 def peak_cut(curve_raw, threshold):
4   curve_new = np.copy(curve_raw)
5   for i in range(curve_new.shape[1]):
6     index = np.where(curve_new[:,i]>threshold)
7     curve_new[index,i] = threshold
8   return curve_new
```

An example of data with the cut of the peak can be visualized in Figure 4.10.

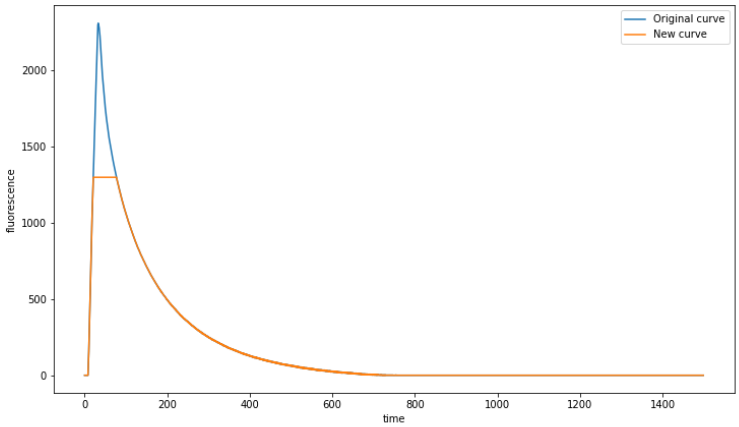
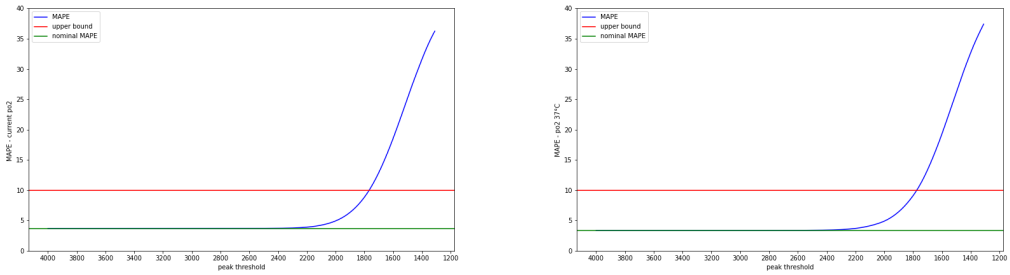


Figure 4.10: Example of the cut of the peak with threshold of 1300

Figure 4.11 shows the results obtained using the perturbed data to predict both current pO2 and pO2 at 37°C. You can see that the model is robust to the peak cut alteration up to the threshold of 2200. After this threshold, the MAPE metrics increase exponentially and already with the threshold of almost 1800 the MAPE reaches the upper bound.



(a) (b)

Figure 4.11: MAPE graphs for the cut of peak alteration

4.3.4 Amplification

The domain analysis highlights how different probes and spotlights modify the curve shape associated with the same pO2 values. To test the robustness on all these variable input data, we generate the altered dataset by amplifying the original curves, performing possible variations caused by probes and/or spotlights. We apply the perturbation in the range between 1 (original curve) and 2 with step of 0.1, indeed redouble the amplitude covers most of the critical situations. An example of data with amplification can be visualized in Figure 4.12.

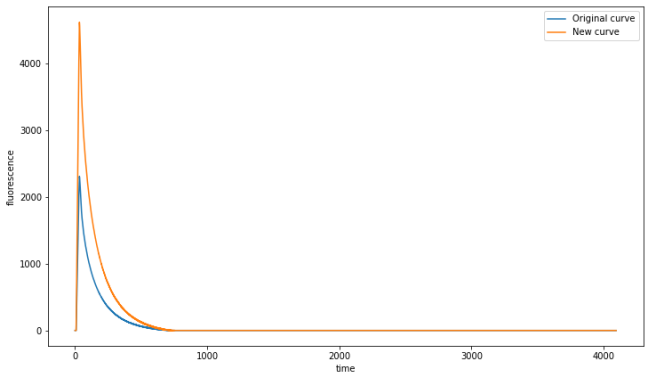


Figure 4.12: Example of the amplification of a curve

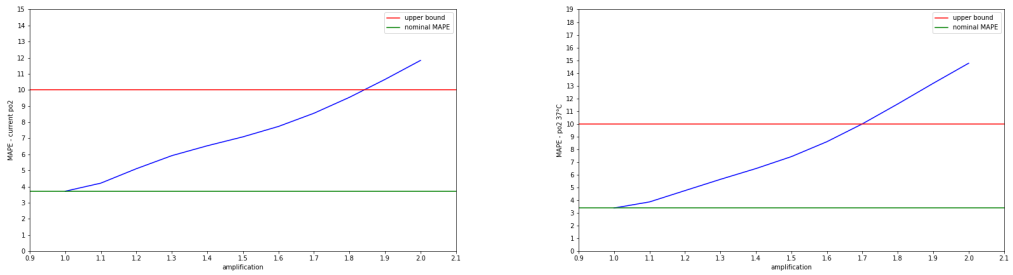
Below the code implemented to execute the amplification alteration.

```
import numpy as np

def amplification_attenuation(curve_raw, amp_att):
    curve_new = np.copy(curve_raw)
    curve_new *= amp_att

    return curve_new
```

Figure 4.13 shows the results obtained using the perturbed data to predict both current pO2 and pO2 at 37°C. You can see that the model has not good robustness on amplification perturbation: the errors increase almost linearly with the increase of the perturbation grade. Furthermore, MAPE values overtake the upper bound for the amplification of almost 1.8.



(a) (b)

Figure 4.13: MAPE graphs for amplification alteration

4.3.5 Attenuation

This alteration is the opposite of the previous one. We decided to evaluate separately these two modifications both related to the amplitude, in order to highlight potential differences in the network behavior after modifications applied to the curve amplitude. Since we want to compare the robustness of these opposite perturbations, we apply the attenuation alteration in the range between 1 (original curve) and 0.5 with the step of 0.1.

An example of data with attenuation can be visualized in Figure 4.14.

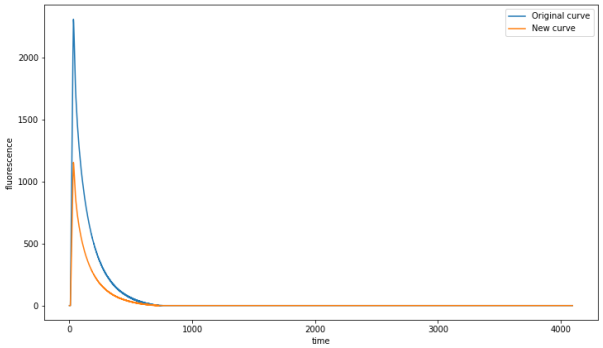
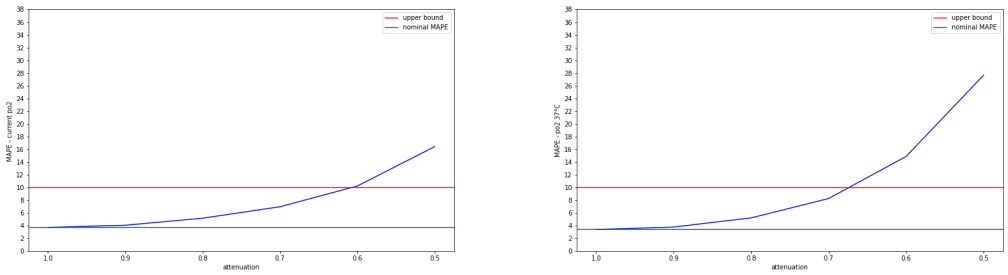


Figure 4.14: Example of the attenuation of a curve

The function implemented to realize this alteration is the same as the amplification method presented in the previous section.

Figure 4.15 shows the results obtained using the perturbed data to predict both current pO2 and pO2 at 37°C. You can see that, in general, the model is less robust to attenuation perturbations than to amplification ones. For small perturbations (attenuation of 0.9) the MAPE values remain close to the nominal values, however, the increase of the grade of alteration causes a rapid rise in the MAPE errors, which overtake the upper bound around the attenuation value of 0.7.



(a) (b)

Figure 4.15: MAPE graphs for attenuation alteration

4.4 Robustness analysis

In the previous section, we defined all alterations to perform on the original dataset, as well as their range, in order to compute the robustness of this model. We have already determined the alteration graph for each perturbation, showing the changing of MAPE based on the alteration grade. The next step of the test framework consists of the robustness analysis. It concerns the computation of the robustness metric for each alteration, as described in Chapter 3. Therefore, we computed the area between the alteration graph and the upper bound and we compare it with the ideal area (the area between the nominal MAPE and the upper bound into the alteration range). In particular, the robustness calculation has been performed using the four functions defined in Section 3.2: classical error function, error with linear weight function, error with exponential weight function (using $\alpha = 0.8$ to give more importance to perturbations close to the original data) and Heaviside Step function. This method generates a percentage for each alteration, representing the model robustness on that alteration, indeed it highlights how close to the ideal case the model behaves with perturbed inputs.

Tables 4.1 and 4.2 summarize the robustness % obtained for each alteration, both for current pO2 prediction and pO2 at 37°C. We can notice that tests got different results. The model has high robustness for clock offset and peak cut alterations, while it is more sensitive to amplification and attenuation. Instead, the worst model robustness concerns the cut of the curve end.

Alteration	**Range**	**Robustness %**			
		Error	**Linear weight**	**Exponential weight**	**Heaviside**
Curve end cut	[641-620]	15.65	28.62	34.38	27.27
Clock offset	[0-30]	61.08	78.55	94.59	89.99
Peak cut	[4000-1300]	77.66	94.80	100	82.96
Amplification	[1-2]	45.68	65.22	67.79	81.81
Attenuation	[1-0.5]	53.30	75.86	67.66	66.66

Table 4.1: Summary of current pO2 robustness for MLP model

Alteration	Range	Robustness %			
		Error	Linear weight	Exponential weight	Heaviside
Curve end cut	[641-620]	19.33	34.48	40.41	27.27
Clock offset	[0-30]	53.90	73.18	93.34	86.66
Peak cut	[4000-1300]	77	94.45	100	82.59
Amplification	[1-2]	41.03	62.07	65.38	72.72
Attenuation	[1-0.5]	48.81	71.82	63.73	66.66

Table 4.2: Summary of pO2 at 37°C robustness for MLP model

4.5 Noise robustness

Another important test to evaluate the performance of a model is its robustness to noise. In electronic systems, there are many noise sources that can perturb the output signal. Many noises can be approximated as white noise. White noise is a stationary Gaussian random process with zero mean and flat power spectral density (Figure 4.16).

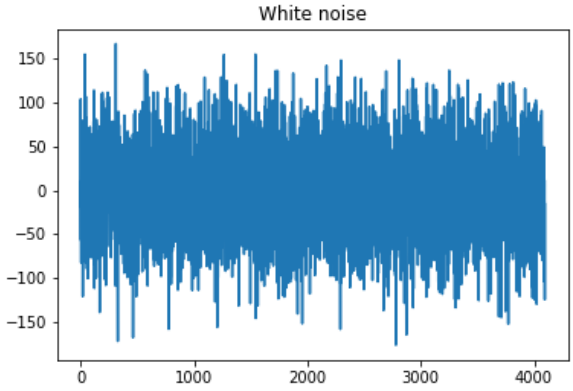


Figure 4.16: Example of white noise

As all electronic signals, the data available are already affected by noises. To verify the model robustness on additive noises, which could affect the electronic system in situations different from those represented by the available data, we add white Gaussian noises to the original dataset.

Indeed, to perform a more complete analysis we generate white noises with different standard deviations into the range between 1 (the standard deviation usually utilized) and 50. We do not consider values greater than 50 since they will cause a signal to noise ratio too low which is not a realistic situation: the white noise with the

standard deviation of 50 causes a signal to noise ratio already close to the acceptable limit of $10dB$. In Figure 4.17 it is possible to compare an original curve and the new curve obtained by adding a white noise with the standard deviation of 50, you can notice that the noise causes a significant modification of the original curve.

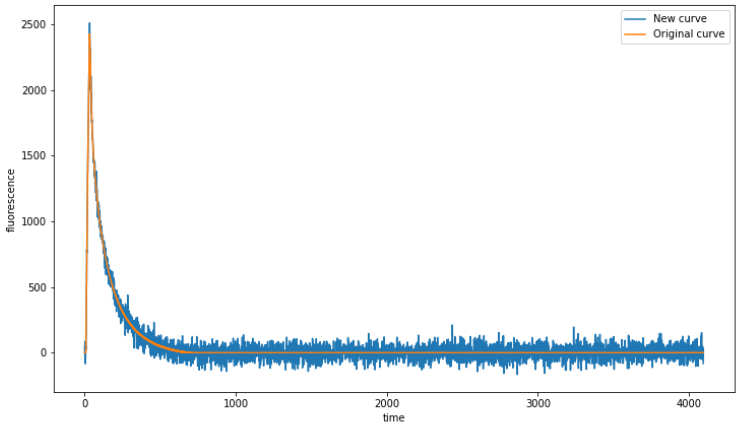


Figure 4.17: Comparison between the original curve and curve with white noise with std=50

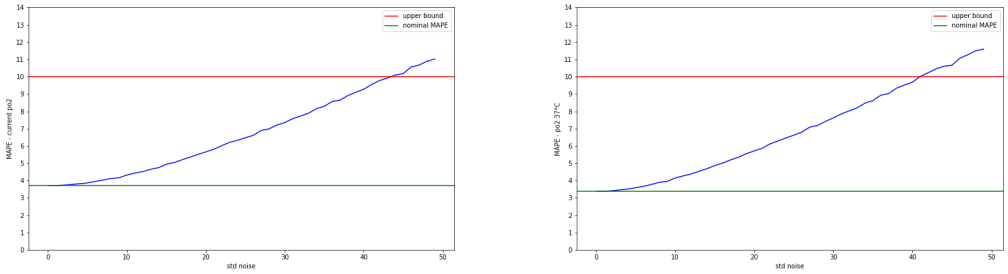
Below the code to add white noise to the original data.

```
import numpy as np

def add_noise(curve_raw, noise_std):
   curve_new = np.copy(curve_raw)
   mean = 0
   noise = np.random.normal(mean, noise_std,
      size=curve_raw.shape[1])
   curve_new += noise
   return curve_new
```

In Figure 4.18, you can see the results obtained by using standard deviation in the range between 0 and 50. In general, the model is robust to white noise injection. In particular, its metrics are not subjected to several changes by adding white noise with low values of standard deviation. White noise standard deviations close to the nominal case are the most common since they represent a better signal to noise ratio which is an important requisite for these applications. In addition, we can notice that only white noises with high values of standard deviation, so with a signal to noise ratio close to the acceptable SNR limit of 10 dB, generate a high MAPE. However, these noises can be considered rare situations, as you can see from Figure 4.17 they cause a significant

change in the curve, therefore this could happen for anomalies in the system or critical cases.



(a) (b)

Figure 4.18: MAPE graphs for additive white noise alteration

The robustness metrics obtained for this test are summarized in Table 4.3.

	Robustness %			
	Error	**Linear weight**	**Exponential weight**	**Heaviside**
current pO2	53.34	73.37	96.83	88
pO2 37°C	46.55	67.57	95.77	77.99

Table 4.3: Summary of noise robustness for MLP model

In general, we can conclude that this model is robust to additive noise, this could be attributed to the processing of the curve in order to get the input parameters. Indeed, the input parameters are means of some range of values, this mitigates the curve noises, both intrinsic and additive white noises.

Chapter 5

Data augmentation

In the previous chapter, we introduced a test framework for a machine learning model for regression problems and we presented its implementation through a multilayer perceptron neural network. Most neural network models are not robust against all possible perturbations. In this chapter, we will analyze two methods to improve the robustness of a multilayer perceptron neural network. These methods consist of enriching the training dataset by adding perturbed data. In this way, we can improve the training phase with non-nominal conditions. In order to improve the model robustness, we implement "data augmentation", the classical method described by the state of the art, and a modified version of the "incremental learning" method.

5.1 State of the art

As explained in previous chapters the basic testing procedures concern the check of the dataset. Indeed, an appropriate training dataset is a crucial aspect in order to obtain models that perform well. This dataset has to provide a good representation of the population, therefore its size is an important property too. The variability of the dataset is crucial because of the training phase procedure. The training procedure is based on the tuning of the network weights such that a specific input can be associated with an output; in particular, the optimization goal is to determine the best model (the right weights tuning) which minimize the loss function, the error between prediction and target values. The choice of the network structure is an important step in the model definition since the numbers of network parameters (weights and biases) are directly related to the dataset size. Therefore, a small dataset does not produce a generic model and could fall into the overfitting problem. This problem is very typical in machine learning neural networks where only small datasets are available; in this case, the model gets good results with the training dataset, while it obtains bad

performances, high errors, with new input data: the model cannot generalize well the problem. Furthermore, the number of network parameters depends on the complexity of the task to perform. For this reason, for many machine learning applications, a big dataset is a requirement to get good performance results. The main problem is that in many applications an appropriate dataset, in terms of size and variety, is not available. The typical solution to overcome this issue is the implementation of the data augmentation technique.

Data augmentation includes a suite of techniques that increase the size and quality of training datasets, it can act as a regularizer to make a better-generalized model. This technique is mostly implemented for deep learning image classification. In computer vision tasks, data augmentation can be implemented through various algorithms: geometric transformations, color space augmentations, kernel filters, mixing images, random erasing, feature space augmentation, adversarial training, generative adversarial networks [23].

Since our case study is a regression problem based on a multilayer perceptron model, we implemented data augmentation using a modified version of some classical data augmentation methods for classification tasks. The first data augmentation approach applied is a new interpretation of the mixing images technique. The basic idea of this method is to create new input data by recombining existing ones [6].

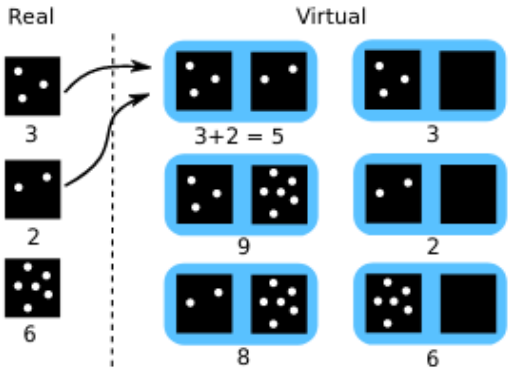


Figure 5.1: Example of mixture data augmentation for images classification

For this case study, we implement a similar approach. Because of various types of probes and/or spotlight, there are many different curves associated with the same pO2 values. For this reason, we determine new samples by averaging the parameters of two curves with the same pO2 targets. In particular, we executed an analysis to determine how the model performances change by adding into the original dataset new samples computed in three different modes: using the mean of two curves with low error predictions and same pO2 target values, using the mean of two curves with out of range prediction errors and same pO2 target values, using the mean of two curves,

one with low error predictions and one with prediction errors out of the acceptable range, with the same pO2 target values. This analysis highlights that the model has good performances with new samples obtained by averaging two curves with low error in predictions. However, averaging two curves with out of range prediction errors cause high prediction errors too. In this case, 76% of new samples have out of range prediction errors. Better results can be observed with the dataset realized applying the third mode, in this case only 20% of the new dataset has out of range prediction errors.

Therefore, in order to generalize the model and improve its performances through the mixture technique, we added new curves to the dataset. In particular, the augmented dataset consists of samples generated through the second and third methods (using curves with out of range prediction errors); in this way, the new samples are associated with some particular situations which the original model was not able to learn properly. Below you can see the pseudo-code to generate the new dataset through the mixture technique.

```
Input
curve_raw : curves obtained in response to a bright pulse
model : model trained to predict pO2 values

Output
new_dataset : dataset generated by means of mixture technique

// Compute mean curves from curves with out of threshold
   prediction errors
Procedure: mean_curves_from_out_threshold
// Select from curve_raw all the curves which have out of
   threshold prediction errors
out_threshold_curves = selection_out_curves(curve_raw, model)

curve_new = array()

// For each pO2 value associated with the out_threshold_curves
for pO2 in pO2(out_threshold_curves):
  // Sample from out_threshold_curves 2 curves with same pO2
     value (if available)
  curve1, curve2 = sample_curves(out_threshold_curves, pO2)
  // Compute the mean of the 2 curves sampled
  mean_curve = mean(curve1, curve2)
  // Insert the new curve in the array
  curve_new.add(mean_curve)
```

```
23 return curve_new
24
25 // Compute mean curves from out and in threshold curves
26 Procedure: mean_curves_from_out_in_threshold
27
28 // Divide curve_raw in curves which have out and in threshold
      prediction errors
29 out_threshold_curves = selection_out_curves(curve_raw, model)
30 in_threshold_curves = selection_in_curves(curve_raw, model)
31
32 curve_new = array()
33
34 // For each curve in out_threshold_curves
35 for out_curve in out_threhold_curves:
36   // Sample from in_threshold_curves 1 curve with same pO2
        value of out_curve
37   in_curve = sample_curves(in_threshold_curves,
        pO2(out_curve))
38   // Compute the mean of the 2 curves sampled
39   mean_curve = mean(in_curve, out_curve)
40   // Insert the new curve in the array
41   curve_new.add(mean_curve)
42 return curve_new
43
44 Procedure: mixture_technique_dataset
45
46 // Combine results from procedures
      mean_curves_from_out_in_threshold and
      mean_curves_from_out_threshold
47
48 new_dataset = mean_curves_from_out_in_threshold() +
      mean_curves_from_out_in_threshold()
49
50 return new_dataset
```

The second approach implemented for the data augmentation technique is similar to basic manipulation methods for image classification, which include geometric transformations, color space augmentations, kernel filters, random erasing, feature space augmentation. The idea is to duplicate an input sample by applying some kind of alterations so the model can learn from more various samples. A crucial requirement

is to augment the input vector in a way that preserves the features needed to make the prediction. Therefore we want to apply modifications to the original input vector which do not affect the pO2 values associated. In Section 4.3, we described all perturbations which could occur in the real-world system and we defined the appropriate ranges to consider. This second data augmentation approach has been performed by building an augmented dataset (in addition to the new dataset obtained with mixture technique) through these five alterations: the cut of the curve, clock offset, the cut of the peak, amplification and attenuation. Below you can see the pseudo-code to generate the new curves to augment the dataset. This procedure selects a specific number of samples from the set of new curves generated by applying a perturbation. All the samples related to different perturbations will be combined to integrate the original dataset.

```
Input
curve_raw : curves obtained in response to a bright pulse
model : model trained to predict pO2 values
n_samples : number of samples for each alteration

Output
new_curves: samples of curves generated by applying an
   alteration to augment the original dataset

// Compute the curves applying an alteration
Procedure: alteration_new_curves
new_curves = array()

// Compute new curves by applying any alteration in the
   chosen range
for p in range(perturbation_range):
  altered_curves = alteration_method(curve_raw, p)
  new_curves.add(altered_curves)

// Select from new_curves the number of samples required to
   generate the new dataset
step = int(new_curves.shape[0]/n_samples)
sampled_curves_index = range(0, new_curves.shape[0], step)
return new_curves[sampled_curves_index, :]

// Generate new dataset through basic manipulation technique
Procedure: basic_manipulation_dataset
```

```
25 // For each alteration compute new curves in the chosen
       ranges and combine it
26 for alteration in alterations_chosen:
27   new_curves += alteration_new_curves()
28
29 return new_curves
```

Another approach to perform data augmentation is noise injection [4]. In particular, in this case study, we inject additive Gaussian white noise to the original dataset. This approach has been integrated into the previous ones: we realize an extra data augmentation method based on a new dataset which includes samples generated using both previous approaches, the mixture technique and basic manipulation methods, but also adding white noise on original samples. The generation of the new dataset depends on the same procedure described for the basic manipulations technique.

Data augmentation techniques require to re-train the model each time we want to increase the dataset and it is necessary to have the original dataset since the model will learn both from the original dataset and the augmented dataset.

In order to reduce the learning time and increase the network generalization at the same time, we implemented a different approach, known as incremental learning. This method allows improving the model performances without retraining all the network every time. Incremental learning is a learning process that takes place whenever new samples are available and adjusts what has been learned according to them. Despite incremental learning is an online machine learning technique, we adapt it to realize a similar offline approach. The idea behind this new approach comes from an experimental method used for classification through a convolutional neural network [22]. This method adds classes to an existent CNN in order to improve the domain of the network. For this reason, the network structure is modified. The new architecture has two different branches: one representing the original network and the other the new and non-trained network. Only the new branch will be trained using only the new dataset. Then the outputs of the two branches are used to calculate the loss function and to update the weights of the trainable branch. Therefore, the new branch is trained to classify the new image types using the support of the original model for previous types. This technique modifies an existent network without retraining the whole system. In our case study we do not use a convolutional neural network, but a multilayer perceptron for a regression problem. Therefore, we modified this method in order to be suitable for regression problems and so to improve the model enriching the training dataset without using the original dataset in the re-training phase. In our case, the new branch has been trained separately from the original one, using only the altered dataset.

After the training phase, the inference of an input vector uses both branches: for each input we compute its prediction through both models (the original and the new ones), these predictions are then weighted equally to determine the final output, as shown in Figure 5.2.

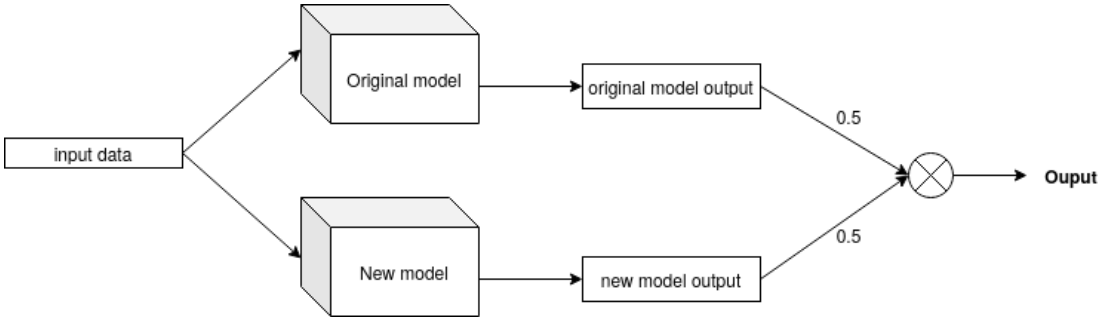


Figure 5.2: Incremental learning technique

In the next section, we will present in more detail all models realized to improve the network performance using both data augmentation and the incremental learning technique with different augmented datasets.

5.2 Models

In this section, we present some models to improve the robustness both using data augmentation and incremental learning techniques. In particular, we use different methods to generate the augmented dataset and we compare the results obtained using:

- mixture technique;
- mixture technique and basic manipulations methods;
- mixture technique, basic manipulation methods and adding white noise on samples.

Except for the mixture technique, for all other methods, the augmented dataset has the size around the 40-50% of the original dataset.

For the first new dataset, we re-train the network using only data augmentation, since its size is really small to implement incremental learning properly. Instead, for the other new datasets, we use both data augmentation and incremental learning for the re-training phase.

With data augmentation, the training is performed using the combination of both the new dataset and the original one. In order to have a correct representation of both the original and new population, both datasets have been split equally between training,

validation and test sets. In particular, the training set is composed of around 70% of the original population and around 30% of new samples.

For the incremental learning technique, the re-training phase has been performed using only the new dataset. Therefore, we have two different networks: one specialized on the original dataset and the other one specialized in the new dataset. In order to get the final output, we combine both predictions with a weighted average.

5.3 Results

In this section, we present the results for both data augmentation method and incremental learning technique, where possible, for all the augmented dataset presented in the previous section. For each dataset, we analyze the metrics obtained on nominal conditions. Indeed, new models must not perform worse on the original dataset, therefore we compute all metrics on the original test dataset (composed of 4330 samples) through the re-trained models. Furthermore, we check the models' behavior on altered conditions by computing their robustness through the same framework described in Section 4.1.

5.3.1 Mixture Technique [MT]

To implement the mixture technique, we generate the new dataset in two ways: we average two curves with out of range prediction errors and we average two curves one with low prediction errors and the other one with errors out of range; in both cases, the curves have the same pO2 target values. Since original samples with out of range errors are 1141, we can generate only 546 new samples with the first averaging method and 1141 with the second one (we chose to determine one new input vector for each sample with out of range errors): the new dataset consists of 1687 samples.

In Table 5.1, you can see the results for the test dataset in nominal conditions using the model re-trained with the augmented dataset generated through the mixture technique. This table shows that the mixture technique improves the performance of the model on the test dataset in nominal conditions.

	MAPE pO2 [%]	**MAPE pO2 37°C [%]**	**OT pO2 [%]**	**OT pO2 37°C [%]**
original model	3.70	3.35	5.17	3.60
augmented model	3.08	3.08	2.12	2.58

Table 5.1: Summary of results in nominal conditions dataset for mixture technique models

Instead, in Section 5.3.4 we will present a summary of the robustness evaluation for the model trained using this augmented dataset. In particular, we will compare the robustness graph for each alteration using all different re-trained models. We will present only the graphs related to current pO2 predictions since we obtained similar results for pO2 at 37°C predictions.

5.3.2 Mixture Technique and Basic Manipulation [+BM]

Another method to perform data augmentation is to generate new samples by applying basic manipulations on the original dataset. We generate a new dataset using all the five alterations determined in Section 4.3 in an equal percentage. We execute different tests to determine the correct range for the alterations. In particular, we tried to use ranges of alterations that cause MAPE up to different limits: alterations with MAPE up to 5% and alterations with MAPE up to 10%. Table 5.2 summarizes all metrics computed on the original test dataset through models re-trained using the dataset generated with different alterations ranges using both data augmentation and incremental learning methods. From this table, we can see that data augmentation is better performing than incremental learning. In addition, the application of small alterations that cause MAPE up to 5% generates higher performances in nominal conditions.

	MAPE pO2 [%]	**MAPE pO2 37°C [%]**	**OT pO2 [%]**	**OT pO2 37°C [%]**
original model	3.70	3.35	5.17	3.60
augmented model 5%	3.13	3.09	2.17	2.79
augmented model 10%	3.22	3.17	2.30	3.37
incremental model 5%	3.43	3.31	3.62	3.62
incremental model 10%	3.62	3.45	4.13	4.08

Table 5.2: Summary of results in nominal conditions using models re-trained with basic manipulations augmentation method

Furthermore, we execute the same tests using both basic manipulations and mixture technique to generate the new dataset. As before we apply all the five alterations determined in Section 4.3. Each alteration contributes to the new dataset for approximately 16%, while the mixture technique for 20%. Table 5.3 summarizes all metrics obtained through the models re-trained using both mixture technique and basic manipulations with different alterations ranges for data augmentation and incremental learning methods.

	MAPE pO2 [%]	MAPE pO2 37°C [%]	OT pO2 [%]	OT pO2 37°C [%]
original model	3.70	3.35	5.17	3.60
augmented model 5%	3.12	3.06	2.21	3.03
augmented model 10%	3.18	3.13	2.54	3.18
incremental model 5%	3.45	3.33	3.62	3.37
incremental model 10%	3.68	3.47	4.64	3.97

Table 5.3: Summary of results in nominal conditions using models re-trained combining mixture technique and basic manipulations

As you can see from Tables 5.2 and 5.3, re-training the model using both mixture technique and basic manipulations allows us to obtain better performances for some metrics than using these methods separated, especially for the data augmentation technique. Therefore, the robustness analysis will be executed only for models re-trained by combining the two techniques. In addition, you can notice that data augmentation has higher performances than incremental learning even when we combine the mixture technique and basic manipulations methods. Moreover, the model has higher improvements when we augment the dataset with perturbations up to 5% of MAPE; for this reason in the following methods, we will consider only alterations that cause up to this MAPE limit. In general, we can conclude that both data augmentation and incremental learning methods, using perturbations up to 5% or 10% of MAPE, improve the model behavior in nominal conditions.

In Section 5.3.4 we will present the robustness evaluation of those models and we will compare it with all other data augmentation techniques.

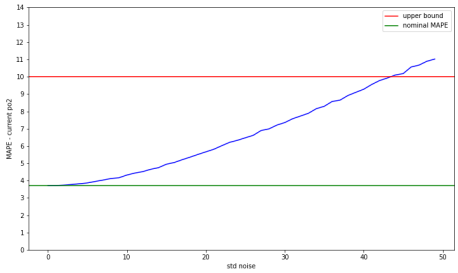
5.3.3 Mixture Technique, Basic Manipulation and Additive White Noise [+AWN]

Data augmentation can be realized also by injecting noises on the dataset. Therefore, we re-trained the model using an augmented dataset by adding white noise to the original samples. Furthermore, we re-train the model generating a new dataset from the combination of additive white noise method, basic manipulations and mixture technique, both with data augmentation and incremental learning method. In both cases, we consider white noises with standard deviations that cause up to 5% of MAPE. Table 5.4 summarizes all metrics computed on the original test dataset with models re-trained using both data augmentation and incremental learning methods.

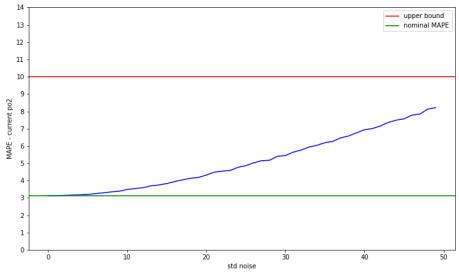
	MAPE pO2 [%]	MAPE pO2 37°C [%]	OT pO2 [%]	OT pO2 37°C [%]
original model	3.70	3.35	5.17	3.60
only awn model	3.12	3.10	2.35	2.60
augmented model	3.11	3.06	2	2.79
incremental model	3.32	3.20	2.86	3.02

Table 5.4: Summary of results in nominal conditions dataset for noise injection models

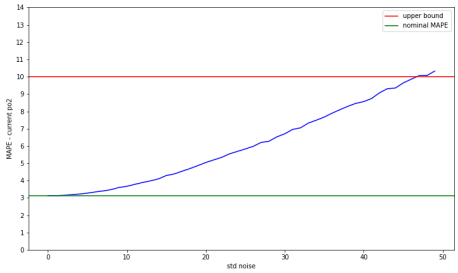
From Table 4.18, we can notice that using the combination of the three data augmentation methods allows us to get better results in nominal conditions than using only the additive white noise method (only awn model), especially for data augmentation technique. In Figure 5.3, we can compare the noise robustness for the different models.



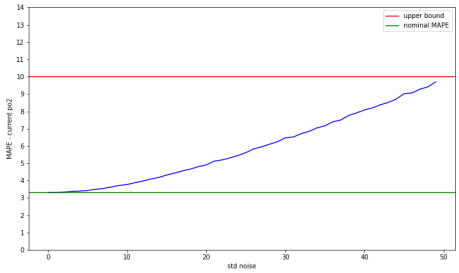
(a) Nominal model



(b) Only awn model



(c) +AWN augmented model



(d) +AWN incremental model

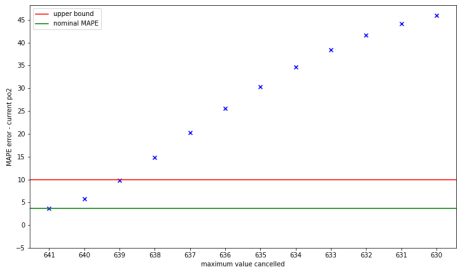
Figure 5.3: MAPE graphs for noise alteration with all different models

You can see that re-training the model using only additive white noise augmentation (Figure 5.3b) produces better results for noise robustness than using the combination of additive white noise, basic manipulations and mixture technique methods (Figure 5.3c).

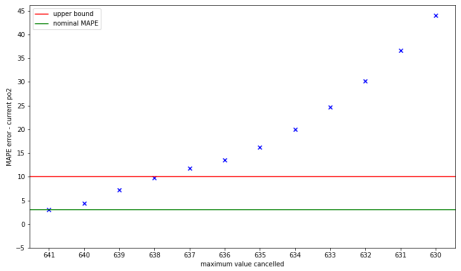
In Section 5.3.4, we will present the robustness of the model trained with the combination of mixture technique, basic manipulations and additive white noise.

5.3.4 Comparison of models robustness

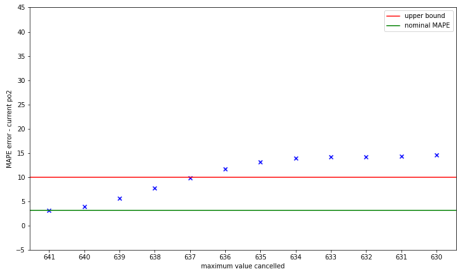
This section presents the robustness evaluation, for each alteration presented in Section 4.3, for models re-trained through data augmentation techniques.



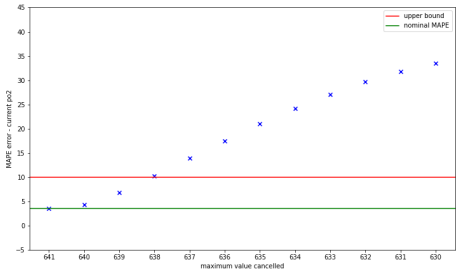
(a) Nominal model



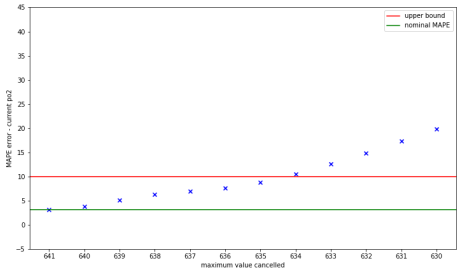
(b) MT model



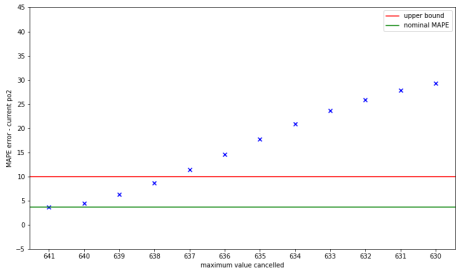
(c) +BM 5% augmented model



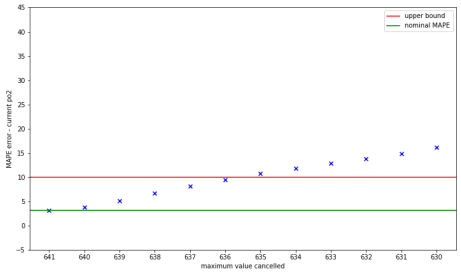
(d) +BM 5% incremental model



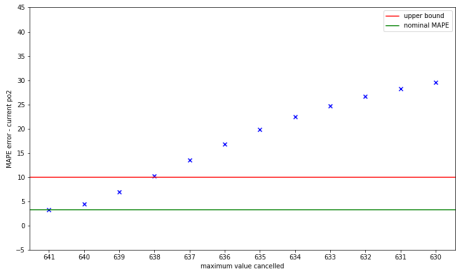
(e) +BM 10% augmented model



(f) +BM 10% incremental model

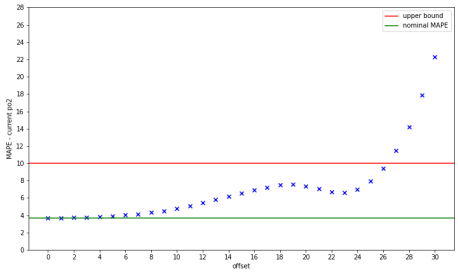


(g) +AWN augmented model

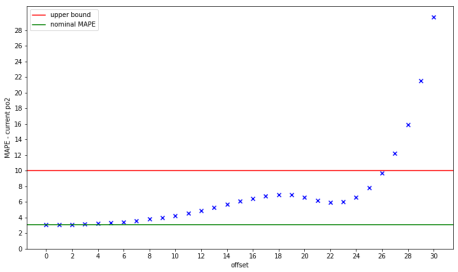


(h) +AWN incremental model

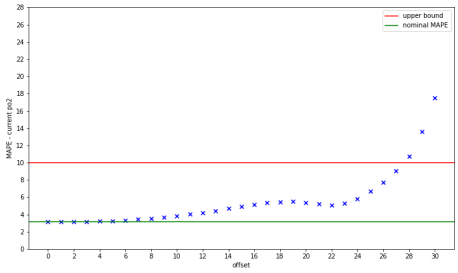
Figure 5.4: MAPE graphs for the cut of the end curve alteration with all different models



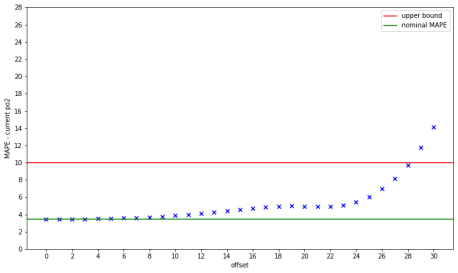
(a) Nominal model



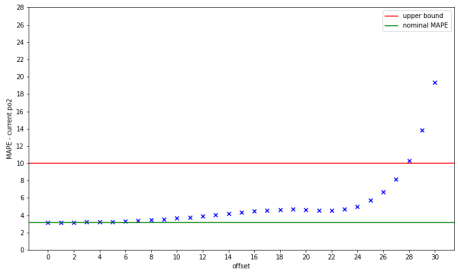
(b) MT model



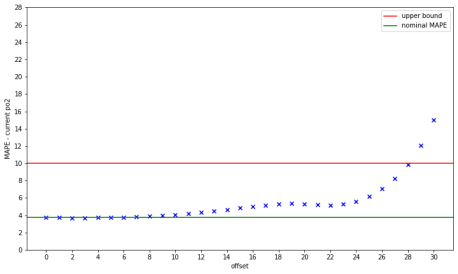
(c) +BM 5% augmented model



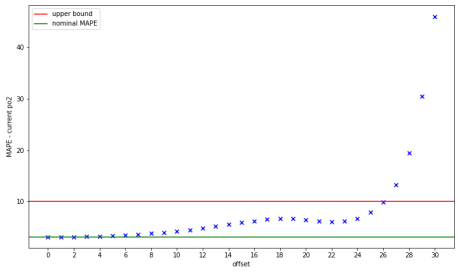
(d) +BM 5% incremental model



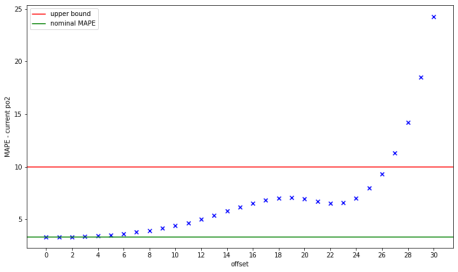
(e) +BM 10% augmented model



(f) +BM 10% incremental model

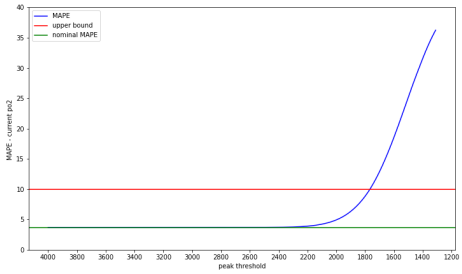


(g) +AWN augmented model

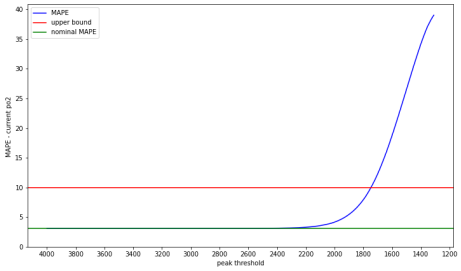


(h) +AWN incremental model

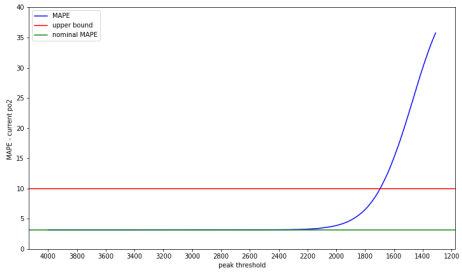
Figure 5.5: MAPE graphs for offset alteration with all different models



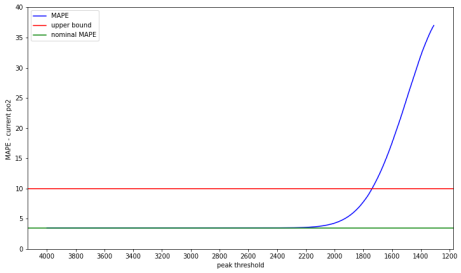
(a) Nominal model



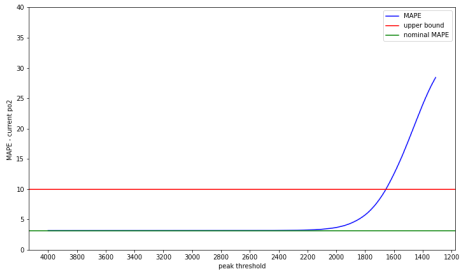
(b) MT model



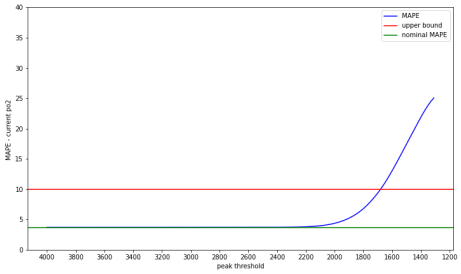
(c) +BM 5% augmented model



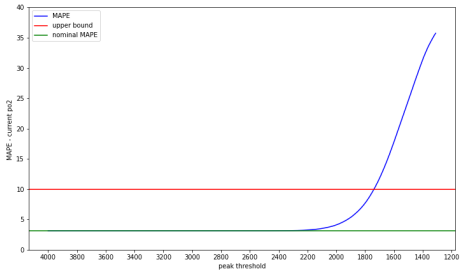
(d) +BM 5% incremental model



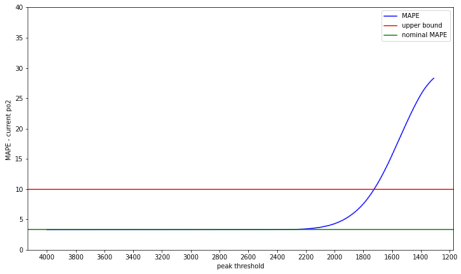
(e) +BM 10% augmented model



(f) +BM 10% incremental model

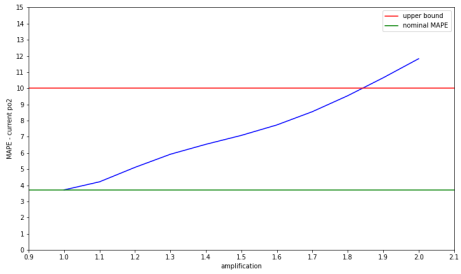


(g) +AWN augmented model

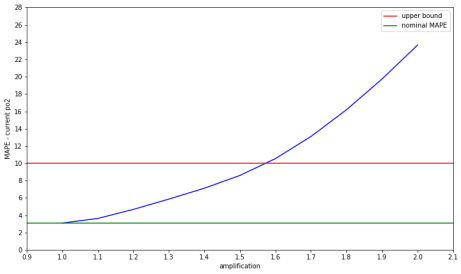


(h) +AWN incremental model

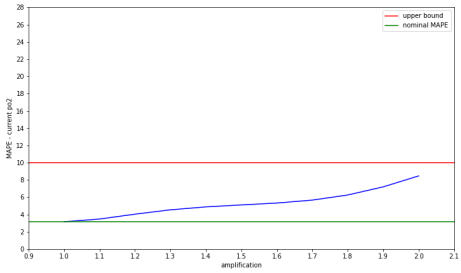
Figure 5.6: MAPE graphs for peak alteration with all different models



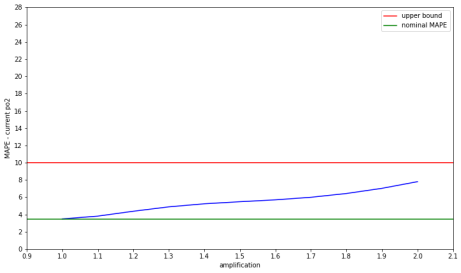
(a) Nominal model



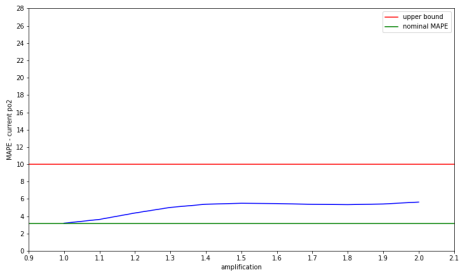
(b) MT model



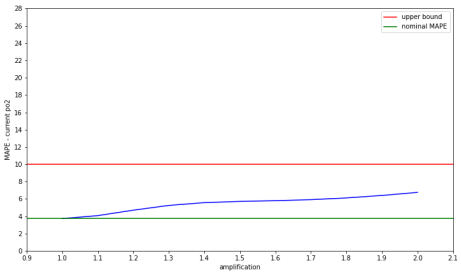
(c) +BM 5% augmented model



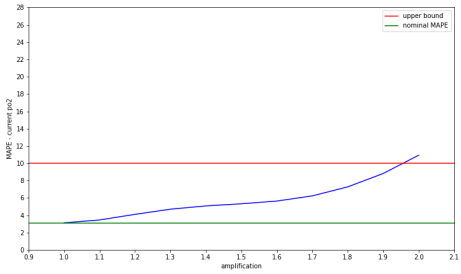
(d) +BM 5% incremental model



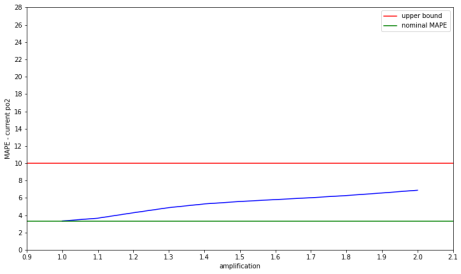
(e) +BM 10% augmented model



(f) +BM 10% incremental model

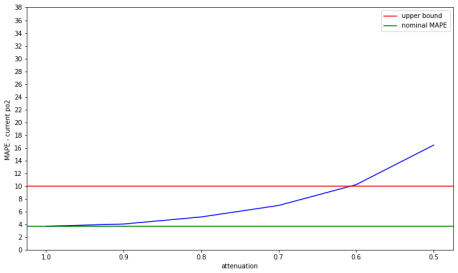


(g) +AWN augmented model

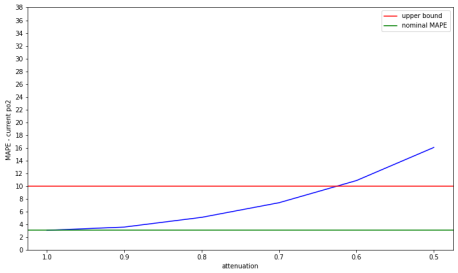


(h) +AWN incremental model

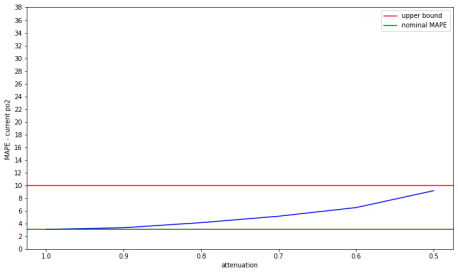
Figure 5.7: MAPE graphs for amplification alteration with all different models



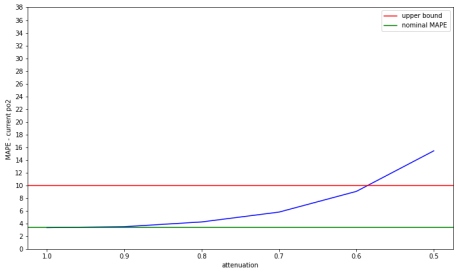
(a) Nominal model



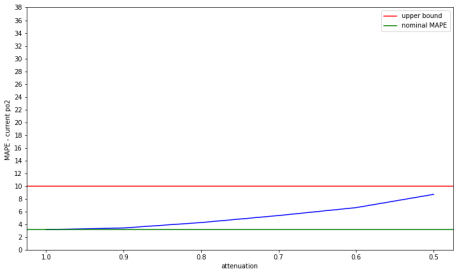
(b) MT model



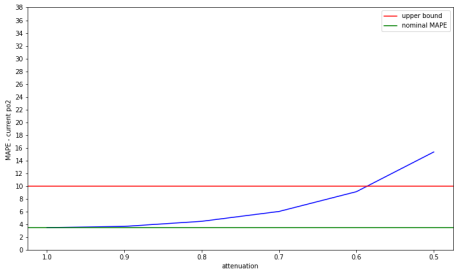
(c) +BM 5% augmented model



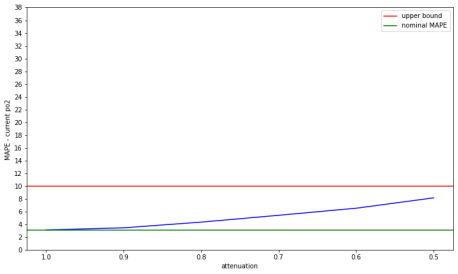
(d) +BM 5% incremental model



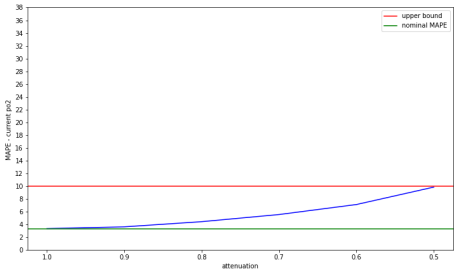
(e) +BM 10% augmented model



(f) +BM 10% incremental model



(g) +AWN augmented model



(h) +AWN incremental model

Figure 5.8: MAPE graphs for attenuation alteration with all different models

From Figures 5.4-5.8, you can see that both data augmentation and incremental learning methods improve the robustness on all the five basic manipulations analyzed: the cut of the curve end, clock offset, the cut of the peak, amplification and attenuation. In general, data augmentation has better results than incremental learning. Even apply-

ing only the mixture technique (+MT model), we can get a model more robust than the original one, especially in the cut of the curve end and in attenuation alterations, even though this model has not been re-trained using the perturbed dataset. While the combination of basic manipulations and mixture techniques (+BM models) generates the most robust models, especially when we augment the dataset through alterations with high errors (MAPE up to 10%). Instead, combining also the noise injection technique (+AWN models), we get small improvements in the robustness of every alteration.

We can conclude that to improve the model performances we can re-train the model by using different techniques to enrich the original dataset: mixture technique, basic manipulations method, additive white noise technique, or their combinations. Indeed, all these techniques provide better results both in nominal and in perturbed conditions. In nominal conditions, the best improvements regard the "Out of Threshold %" metrics, as you can see from Tables 5.1-5.4. While Figures 5.4-5.8 show that the most significant improvements in robustness concern the cut of the curve end, the amplification and the attenuation alterations. The data augmentation technique is the best option for both robustness and nominal behavior improvements, but it needs more time to re-train the model than the incremental learning technique.

Furthermore, the model re-trained with the combination of mixture technique and basic manipulations up to 5% of MAPE (+BM 5%) guarantees the best trade-off for the improvements in nominal and perturbed conditions.

Chapter 6

Bayesian Neural Network - BNN

In this chapter, we present a different approach to determine the confidence of a prediction. Indeed, as described in previous chapters, the reliability in terms of robustness and confidence in predictions is an important property for many machine learning application systems. Bayesian neural networks aim to solve this issue by modeling the uncertainty for each network parameter. These models describe a probabilistic relationship, therefore each prediction is associated with an uncertainty value. Bayesian learning is an important technique in machine learning systems since the real world is uncertain: data are noisy and the target values could be uncertain. This chapter presents an introduction to Bayesian machine learning, highlighting the different uncertainty types which can be modeled. Later in this chapter, a practical example of Bayesian neural networks for this case study will be presented.

6.1 Bayesian machine learning

In literature, there are many different definitions for the Bayesian neural network, but a common one is that a Bayesian neural network is a stochastic artificial neural network trained using Bayesian inference. Therefore, Bayesian neural networks aim to combine the strengths of neural networks and probabilistic modeling.

As explained in Section 2.1.2, in multilayer perceptrons and in general in supervised learning models the training phase is based on the back-propagation technique: the relationship between the input data and the output is approximated by adjusting the weights, in order to minimize a loss function. Therefore, in deep learning, the training phase results in the estimation of network weights which minimizes a cost function. Bayesian neural networks, in contrast, aim to estimate the posterior distribution of the weights, which can be used to quantify the uncertainty of the network.

In particular, Bayesian inference makes predictions by averaging over all probable

explanations under the posterior distribution. This process consists in deducing properties about a population or probability distribution from data using Bayes' theorem (Figure 6.1).

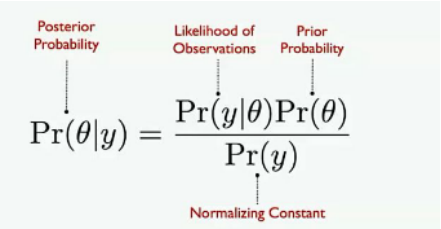


Figure 6.1: Bayes' rule

In the formula in Figure 6.1, the prior distribution $Pr(\theta)$ represents our beliefs about the true value of the parameters. Instead, the likelihood of observations $Pr(y|\theta)$ represents the probability distribution of the observed data, given a parameter's value. While, the posterior distribution $Pr(\theta|y)$, which is the result we want to obtain, is the distribution representing our belief about the parameter values taking both the prior knowledge and the observed data into account. Therefore, the posterior distribution of our parameters can be computed using our prior beliefs updated with our likelihood. We can summarize that BNN imposes a prior on the neural network's parameters and aims to learn a posterior distribution of these parameters [15]. In particular, the Bayesian inference process is composed of three main steps:

- model specification: definition of the prior distribution for each parameter;
- model inference: computation of the posterior distribution for each parameter;
- model checking: evaluation of the trained model and comparison with different models.

The definition of the prior distribution is an important step since the correctness of the posterior depends on it. In many cases, we do not have deep knowledge in the domain and in the right prior distribution to use, therefore the common practice consists of considering a normal distribution [9].

6.1.1 Model Inference

The most important phase for Bayesian inference is the computation of the posterior distribution. There are two main different families of methods to implement this phase: sampling methods and variational inference methods [20].

Sampling methods

Sampling techniques include mostly the Markov Chain Monte Carlo family of algorithms, where the parameter space is determined from the samples of a given distribution. Therefore, instead of trying to deal with complex computations to determine the posterior, we can get samples from this distribution and use these samples to compute statistics such as mean and variance (as shown in Figure 6.2).

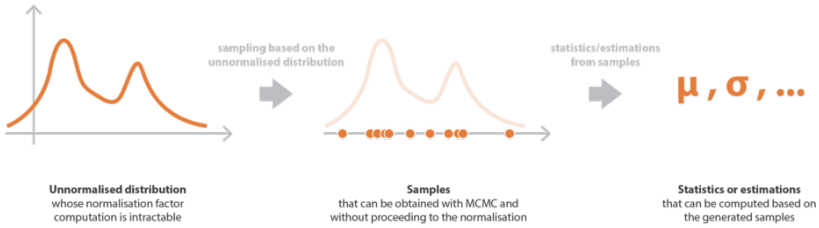


Figure 6.2: MCMC sampling method [20]

These algorithms are based on two methods: Monte Carlo and Markov Chain. Monte Carlo is a technique for sampling a probability distribution randomly to approximate the desired quantity. While Markov Chain is a method for generating a sequence of random variables where the current value is probabilistically dependent on the value of the prior variable.

In order to produce samples, the idea is to set up a Markov Chain whose stationary distribution is the one we want to sample from. From this Markov Chain, we simulate a random sequence of states long enough to reach a steady state. Then, we keep some generated states as samples; in particular, to consider independent the states chosen, they have to be far enough from each other. Therefore, the MCMC approach is based on the ability to build a Markov Chain properly. There are many algorithms for this purpose, the most common are Metropolis-Hasting and Gibbs Sampling algorithms. Using these samples we can then compute the statistics required [20].

Variational Inference

This technique consists in finding the best approximation of the distribution among a parametrized family [20]. Therefore, the inference of the posterior is converted into an optimization problem. Supposing we are given the intractable probability distribution of the posterior p. Variational techniques consist of trying to solve an optimization problem over a class of tractable distributions Q in order to find a $q \in Q$ that is most similar to p. Therefore, to get an approximate solution we will use the distribution q

which approximates p. Since the posterior distribution p is intractable we use its unnormalized version $\hat{p}$. In Figure 6.3 a summary of the variational inference technique.

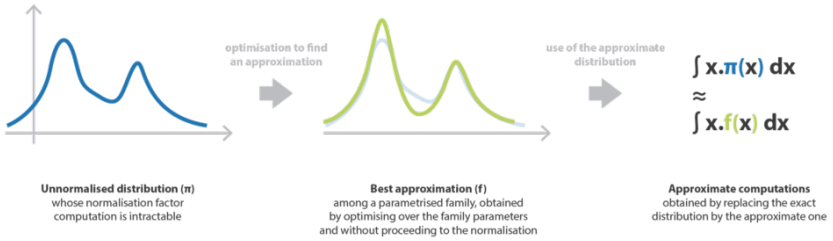


Figure 6.3: Variational Inference technique [20]

Therefore, to cast inference as an optimization problem, we need to choose an approximating family Q and an optimization objective $J(q)$.

The optimization function needs to represent the similarity between q and p. For this purpose, the state of the art provides the Kullback-Leibler (KL) divergence function:

$$KL(q||p) = \int q(z) \log \frac{q(z)}{p(z)} \, dz$$

where z is the vector of distribution parameters to determine. This formula is used to measure differences in the information of two distributions. The KL divergence is based on the following properties:

- $KL(q||p) \geq 0$ for all q, p
- $KL(q||p) = 0$ if $q = p$

However, we cannot compute exactly the KL formulation since we need to evaluate the distribution p, which is not available. Therefore, we will use an optimization objective based on this method using its unnormalized probability distribution $\hat{p}(z) = p(z|\theta)$:

$$J(q) = \int q(z) \log \frac{q(z)}{\hat{p}(z)} \, dz$$

Since $p(z) = \frac{\hat{p}(z)}{Z(\theta)} = p(z|\theta) * p(\theta)$, where $Z(\theta)$ is the normalization constant, we can represent the optimization objective as:

$$J(q) = \int q(z) \log \frac{q(z)}{p(z)} - \log Z(\theta) = KL(q||p) \, dz - \log Z(\theta)$$

Therefore, using the properties defined above, we can determine a lower bound on the log partition function $\log Z(\theta)$ also called variational lower bound or evidence-lower

bound (ELBO):

$$\log Z(\theta) = KL(q||p) - J(q) \geq -J(q) = E_{q(z)}[\log \hat{p}(z) - \log q(z)]$$

Thus, by maximizing the evidence-lower bound, we are minimizing the Kullback-Leibler divergence [7].

After the determination of the optimization objective, the other important step in variational inference concerns the choice of approximating family Q. In machine learning literature there are many different ways to parametrize this class of distributions: exponential families, neural networks, Gaussian processes. The choice of the family is important to control through the model both the bias and the complexity of the method. Indeed, if we assume a restrictive model (simple family), the optimization process is simple, but the bias is high. On the contrary, if we assume a free model (complex family), the bias is much lower, but the optimization is harder and could be intractable. Therefore, we need to determine a balance in the complexity of the family model so that it provides a good quality model, but also a tractable optimization process.

One of the most used classes of distributions is the mean-field variational family. A distribution that belongs to this family has a product density such that each component is independent and it is a distinct distribution. It can be written as follows:

$$q(z) = q_1(z_1)q_2(z_2)\dots q_n(z_n)$$

where each $q_i(z_i)$ is a distribution over a one-dimensional discrete variable [7].

The use of this family of distributions for variational inference is known as mean-field inference.

Mean-field inference is easy to optimize and it is based on solving the following optimization problem:

$$min_{q_1,\dots,q_n} J(q)$$

For example, if each density $q_i(z_i)$ is a Gaussian distribution with both mean and variance parameters, the global density $q(z)$ is defined by a set of parameters coming from all the independent components and the optimization is based on the entire set of parameters.

MCMC vs Variational Inference

The main differences between sampling and variational techniques are:

- the determination of the globally optimal solution: variational approaches will

rarely find it, unlike sampling based-methods;

- computational cost: variational inference is faster than MCMC, indeed it scales and suits better techniques like stochastic gradient optimization, parallelism over multiple processors, and acceleration using GPUs.

Therefore, the variational inference is suited to large datasets where we want to quickly explore many models. On the contrary, MCMC is suited to smaller datasets, in particular in scenarios where we can afford heavier computational costs for more precise samples [7].

In particular, for this case study, we implement variational inference with the support of the Tensorflow Probability framework, this implementation will be described in Section 6.3.

6.2 Model uncertainty

The determination of model uncertainty is really important to make more robust decisions (especially in critical applications such as medical diagnosis) by assigning confidence to predictions. However, many machine learning models for regression tasks do not provide a proper quantification of the uncertainty in the predictions. On the contrary, Bayesian neural networks allow dealing with regression (or classification) problems, determining the uncertainty of the trained model predictions.

Uncertainties can be classified in two different sources: aleatoric and epistemic uncertainty [10].

Aleatoric uncertainty refers to the noise in the generative process of the data which causes variability in the data. It is inherent to the natural randomness in the main process of the system.

Instead, epistemic uncertainty concerns the uncertainty of the model. In particular, it concerns the uncertainty of the type and structure of the model to explain properly the generative process of data. It is due especially to limited data and a lack of knowledge.

6.3 Bayesian Neural Network implementation

For this case study, Bayesian neural networks have been implemented through the Tensorflow Probability library. This library is built on Tensorflow to combine probabilistic models and deep neural networks in an easy way especially for modern hardware such as TPU and GPU. It helps data scientists and ML researchers to increase domain knowledge, understand data and make predictions [25] [19].

Using these tools we implement three different Bayesian neural networks: to model epistemic uncertainty, to model aleatoric uncertainty and to model both epistemic and aleatoric uncertainty. All neural network structures have been implemented to be similar to the multilayer perceptron original network, in order to execute a more accurate comparison. Therefore, we maintain the numbers of layers and neurons per layer described in Section 1.1.1.

For each model, we evaluate the metrics comparing the target value with the mean of the output distributions. In addition to MAPE and "Out of Threshold %" metrics, we compute the percentage of predicted distributions which include the target value.

6.3.1 Model aleatoric uncertainty

Aleatoric uncertainty concerns the variability of the process, therefore it is intrinsic in the data generated by the system. To model the aleatoric uncertainty, in addition to predicting the mean of the output distributions, we also predict their standard deviation; therefore, we need to add 2 more neurons to the output layer. The new model is composed of 258 parameters and we use the loss function "weightedMAE" as in the original MLP model. Thus, for each input sample, we obtain in output a normal distribution. The evaluation of the metrics has been executed using the mean value of the normal distributions as predictions.

6.3.2 Model epistemic uncertainty

Epistemic uncertainty concerns the uncertainty on the model type and structure. To model this uncertainty, the standard "Dense" layers have been replaced with "DenseVariational" layers. The "DenseVariational" layer uses variational inference to fit a posterior to the distribution over both the weights and biases to represent the uncertainty in their values. Using a model with this kind of layer, we get a different output every time we pass the same input through the network. This is because "DenseVariational" defines an ensemble of models. Each prediction depends on different model parameters chosen randomly from the posterior distribution of each weight, in this way computing multiple predictions of the same input we can take into account the uncertainty about the model. The variational posterior implemented is a multivariate normal distribution with a trainable diagonal covariance matrix centered on a trainable mean. While the prior distribution is a standard multivariate normal distribution with a trainable mean and fixed unitary scale. Thus, in our model, we are training both the prior and the posterior, so that we do not need to specify the mean of the prior for the network parameters, which can be difficult without prior deep knowledge about the

problem. Moreover, if the priors are set far from their true values, then the posteriors may be affected by this choice. This training method is equivalent to use Empirical Bayes, also known as Type-II Maximum Likelihood. Empirical Bayes methods allow estimating and updating the parameters of a prior probability, before defining a posterior probability distribution. This technique is still based on the general Bayesian statistics theorem. The estimating process consists of two steps: create a probability distribution (hyperparameter) for each parameter in a prior assumption, test the prior probability defined on a sample of data, the hyperparameter is then converted into a value for each parameter. The parameter computed will be used to define the prior probability.

The new model has the same network structure as the multilayer perceptron network: the same number of layers and neurons. The only difference is the type of layers: for this new Bayesian neural network we use dense variational layers, described previously. The number of parameters to learn during the training phase is 708 and we use the loss function "weightedMAE" as in the original MLP model. With this model, we get a different output every time we pass the same input through the network, based on the random model parameters chosen automatically from the posterior by the model. Therefore, the evaluation of the metrics has been implemented by computing the output for each input sample multiple times (100 times per sample). Both MAPE and "Out of Threshold %" values have been determined by comparing for each sample its target value with the mean value of these predictions.

6.3.3 Model aleatoric and epistemic uncertainty

We can also create a model to handle both types of uncertainty at the same time. As for the epistemic uncertainty model, we use dense variational layers to build the network and, as for the aleatoric uncertainty model, we add two extra neurons in the output layer to model the standard deviation of the output distributions. In this way, for each test sample, we predict an ensemble of models based on the posterior distribution and each model reports its variability through a distribution. This model is composed of 774 parameters to train. In this case, we train the model using "weightedMAE" as in the original model, but also the negative log-likelihood loss function. As for the epistemic uncertainty model, the evaluation of the metrics consists of computing the output of each input sample multiple times (100 times per sample). Therefore, for each input sample, we get multiple normal distributions. The MAPE and the "Out of Threshold %" metrics have been computed, as before, comparing the target value of each input sample with the mean values of these distributions.

6.4 Results

The outputs of models that handle aleatoric uncertainty are normal distributions. Therefore, to compute MAPE and "Out of Threshold %" metrics, we consider all the Gaussian distribution through the following formula:

$$\int p(x) * (x - target\_value) * dx$$

Where $p(x)$ is the probability related to each value x in the normal distribution. However, by executing the appropriate simplifications, the previous integral is equal to the difference between the mean value of the distribution, $\hat{x}$, and the target:

$$\int p(x) * (x - target\_value) * dx = \hat{x} - target\_value$$

For this reason, in the following analyses, we computed the error metrics considering only the mean value of the predicted Gaussian distribution. Furthermore, we determine the percentage of predicted distributions which include the target value ("Target In Distribution % (TID)") and the average standard deviation ("StdDev") of the predicted distributions.

In Tables 6.1 and 6.2, you can see the results for current pO2 and pO2 at 37°C predictions for the four BNNs models implemented. For the networks that model epistemic uncertainty, we can compute a range of MAPE and "Out of Threshold %" metric. Indeed, with these models every time we pass the same input through the network we get different predictions; this happens because the network's parameters are chosen randomly from their posterior distribution, therefore the outputs are computed using every time a different model, chosen from the posterior distributions. Both metrics are computed considering the variability of predicted models by executing the prediction for the same input multiple times; therefore, we can determine a range of errors. Differently, the network which models only aleatoric uncertainty has a fixed set of parameters and therefore we get a single value for each metric.

The comparison of these models shows that, modeling only aleatoric uncertainty (aleatoric model), we obtain better performances for both MAPE and "Out of Threshold %" metrics; however, the percentage of target values included in the predicted distribution is lower than the other models. We can notice that modeling epistemic uncertainty and modeling the combination of aleatoric and epistemic uncertainty, using as loss function the "weightedMAE", generate similar results for MAPE and "Out of Threshold %" metrics. While the epistemic model has better performance for TID %, but higher standard deviation. Instead, the model which combines aleatoric and

epistemic uncertainty, trained with the negative log-likelihood cost function, has better performances for MAPE, "Out of Threshold %" and TID metrics than using the "weightedMAE" loss, but higher standard deviation. In particular, this model gets the best results for the TID percentage metric for both current and 37°C pO2 values.

	MAPE [%]	**OT [%]**	**TID [%]**	**StdDev**
aleatoric model	3.86	6.21	85.91	0.0069
epistemic model	5.75 ± 0.85	16.81 ± 5.12	97.52	0.011
epistemic & aleatoric weightedMAE model	5.56 ± 0.86	15.43 ± 5.08	75.06	0.0069
epistemic & aleatoric negloglik model	4.42 ± 0.08	9.43 ± 0.59	98.33	0.013

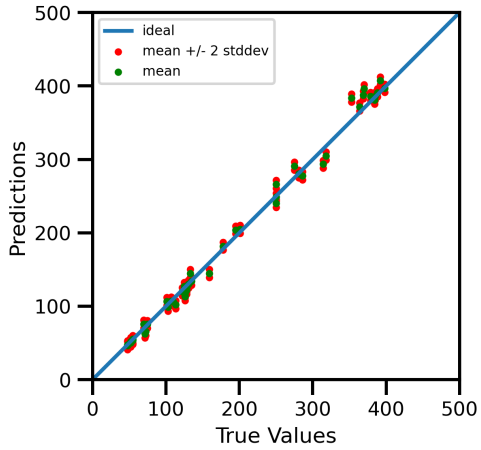
Table 6.1: Summary of results for current pO2 predictions for BNN models

	MAPE [%]	**OT [%]**	**TID [%]**	**StdDev**
aleatoric model	3.56	4.82	91.24	0.0069
epistemic model	5.72 ± 0.73	16.15 ± 4.37	94.57	0.013
epistemic & aleatoric weightedMAE model	5.48 ± 0.83	14.79 ± 4.99	80.23	0.0069
epistemic & aleatoric negloglik model	4.19 ± 0.11	7.99 ± 0.72	99.38	0.013

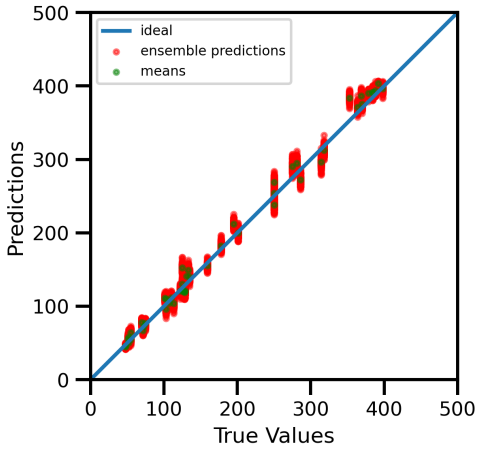
Table 6.2: Summary of results for 37°C pO2 predictions for BNN models

In Figure 6.4 you can see a comparison of predictions for the first 50 samples in the test dataset with the four different Bayesian models presented in Section 6.3. It highlights the differences in the prediction process for BNN models, based on the uncertainty managed. When we model only aleatoric uncertainty, the prediction is a normal distribution, therefore, we can represent the output mean value and the percentiles for each input sample. While, for the epistemic uncertainty model, we get for each input sample a set of different outputs (predictions from different models defined through the posterior distribution of each parameter), wherein we can determine the average of output values. Instead, the combination of aleatoric and epistemic uncertainty generates for each input sample different output predictions which are normal

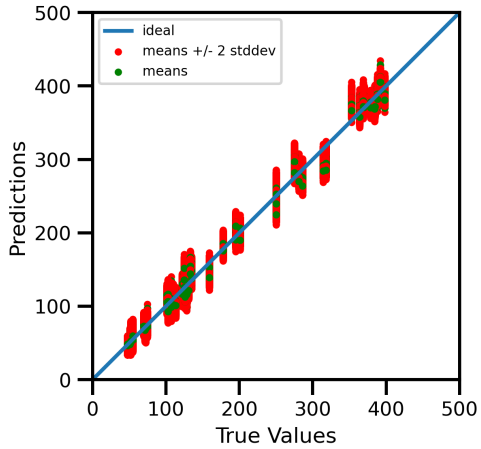
distributions. Therefore, for each input sample, we get different models represented by a mean and a standard deviation.



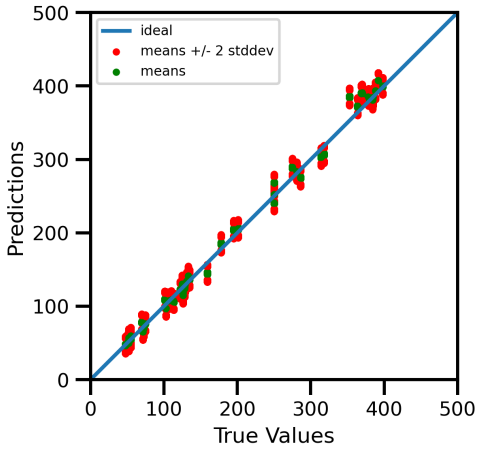
(a) Aleatoric uncertainty model



(b) Epistemic uncertainty model



(c) Aleatoric & epistemic weightedMAE



(d) Aleatoric & epistemic negloglik

Figure 6.4: Examples of current pO2 predictions for some test samples

In general, BNN models have worse results than using the standard MLP model. We expected this worsening because we are implementing a network with similar complexity in terms of layers and neurons, but with more parameters. Indeed, the MLP model is composed of only 224 parameters. Instead, to model epistemic uncertainty, we have to train 708 parameters; while modeling both epistemic and aleatoric uncertainty depends on 774 parameters.

We can conclude that modeling only aleatoric uncertainty generates better results than considering the epistemic one. Furthermore, we get better improvements when we consider the likelihood between the target value and the prediction directly in the loss function. Indeed, the model trained through the negative log-likelihood cost function has better performances than using the "weightedMAE" cost function. In general, all

four models have good results for the “Target In Distribution %” metric, this means that a high percentage of predicted distributions includes the target value.

In the following chapter, we will present the robustness analysis for these models.

Chapter 7

Robustness evaluation of BNN

This chapter presents the robustness evaluation of the main Bayesian neural models realized: a network that models aleatoric uncertainty, networks which model aleatoric and epistemic uncertainty trained with both "weightedMAE" and negative log-likelihood loss functions. The robustness has been evaluated using the four functions described in Section 3.2: classical error function, error with linear weight function, error with exponential weight function (using $\alpha = 0.8$) and Heaviside Step function. We evaluate the robustness of each model on the same tests implemented for the MLP robustness, described in Section 4.3: the cut of the curve end, clock offset, the cut of the peak, amplification, attenuation and additive white noise. Later in this chapter, we will improve the robustness of these models using the data augmentation technique. As first approach, we compute the robustness through the classical MAPE metric, using the mean value of each predicted distribution. However, in order to consider completely the predicted distributions, we will define novel metrics to evaluate the performances of a BNN and we use them to compute the robustness as well.

7.1 Robustness analysis

In Chapter 3 we presented a definition of robustness for regression problems. In particular, we determined four different possible functions to evaluate the robustness of a model: classical error function, error with linear weight function, error with exponential weight function (using $\alpha = 0.8$) and Heaviside Step function. Furthermore, in Chapter 4 we presented the robustness analysis of the original MLP model. To compute this evaluation we defined a test framework and we determined different alterations to test. In order to compare properly the standard MLP model to the Bayesian neural networks, we performed the same analysis on the BNNs which model only aleatoric uncertainty and the combination of aleatoric and epistemic uncertainty. All robust-

ness functions consist of determining how much a metric change when we apply some manipulations on the original dataset. In this case, we have analyzed the MAPE behavior. Therefore, to evaluate the robustness we need to compute the MAPE metric. In Bayesian neural networks, the output is a normal distribution; thus we compute the MAPE metric using as predictions the mean value of the predicted distributions, as explained in Section 6.4. To evaluate the robustness, we compute the same tests executed with the MLP models and we represent each test with a graph showing the variation of MAPE with the increase of a specific alteration of the input data. Each alteration is associated with the same interval used in the robustness analysis for MLP models. Furthermore, to determine the robustness we compute the area included between the MAPE upper bound of 10% and the alteration graph, these results can be compared with the ideal situation, where the MAPE is equal to the nominal value for each grade of alteration, as explained in Section 3.2. For each robustness function, we can obtain a percentage that represents how much the model is robust on the perturbation under analysis.

In this section, we present the robustness analysis only for current pO2 predictions since the results obtained with pO2 at 37°C are similar. In Table 7.1, you can see the robustness results of the model which handles aleatoric and epistemic uncertainty, trained with "weightedMAE" loss function. It presents the results for the four robustness functions when we apply the manipulations described in Section 4.3.

Alteration	**Range**	**Robustness %**			
		Error	**Linear weight**	**Exponential weight**	**Heaviside**
Curve end cut	[641-620]	23.39	40.79	46.81	36.36
Clock offset	[0-30]	36.15	58.29	88.38	50
Peak cut	[4000-1300]	76.25	94.63	100	82.22
Amplification	[1-2]	18.32	32.86	38.74	27.27
Attenuation	[1-0.5]	35.74	57.74	53.25	57.14
Noise	[0-50]	80.42	90.23	100	100

Table 7.1: Summary of current pO2 robustness for aleatoric&epistemic model trained with "weightedMAE" loss

The same analysis has been executed with the network which models aleatoric and epistemic uncertainty, trained with the negative log-likelihood loss function, these results can be seen in Table 7.2.

Alteration	Range	Robustness %			
		Error	Linear weight	Exponential weight	Heaviside
Curve end cut	[641-620]	13.35	24.72	30.08	18.18
Clock offset	[0-30]	47.28	70.66	95.63	69.99
Peak cut	[4000-1300]	73.94	92.90	100	80.74
Amplification	[1-2]	22.06	38.42	44.14	36.36
Attenuation	[1-0.5]	48.59	72.95	66.89	57.14
Noise	[0-50]	76.21	88.18	98.59	100

Table 7.2: Summary of current pO2 robustness for aleatoric&epistemic model trained with negative log-likelihood loss

Instead, in Table 7.3 you can see the robustness results for the model which handles only aleatoric uncertainty, trained with "weightedMAE" loss function.

Alteration	Range	Robustness %			
		Error	Linear weight	Exponential weight	Heaviside
Curve end cut	[641-620]	15.06	27.67	33.38	18.18
Clock offset	[0-30]	33.12	54.48	86.77	43.33
Peak cut	[4000-1300]	75.19	93.46	100	82.96
Amplification	[1-2]	42.12	62.37	65.48	72.72
Attenuation	[1-0.5]	41.59	64.23	59.07	57.14
Noise	[0-50]	77.41	87.75	98.64	100

Table 7.3: Summary of current pO2 robustness for aleatoric uncertainty model trained with "weightedMAE" loss

By comparing Tables 7.1 and 7.2, we notice that modeling both aleatoric and epistemic uncertainty we get similar robustness on the different perturbations, independently by the loss function used in the training phase. Indeed, using the negative log-likelihood cost function, we obtain better robustness for clock offset, but lower results for the curve end cut. While for the other perturbations, the performances are similar using both the negative log-likelihood and the "weightedMAE" loss function.

In general, the networks that model aleatoric and epistemic uncertainty are robust on the peak cut perturbation and additive noise. They are sensitive to the curve end cut and amplification perturbations. While, they have good results, especially for small perturbation degrees, with the clock offset and the attenuation alterations.

Modeling only aleatoric uncertainty generates results similar to modeling both aleatoric and epistemic uncertainty, as you can see from Tables 7.1, 7.2 and 7.3. The main difference regards the amplification robustness, indeed aleatoric model is more robust on this alteration than the others.

We can conclude that in general Bayesian neural models are robust on noise and peak cut perturbations. They get good results for amplification, attenuation and clock offset; while the worst robustness concerns the cut of the curve end.

By comparing these results with Tables 4.1 and 4.2, we can notice that Bayesian neural networks and the standard MLP models have different robustness. In general, Bayesian neural networks are more robust to noise alterations than the standard MLP. However, the multilayer perceptron model has better robustness on the clock offset perturbations. Both models have low robustness on the cut of the curve end. Bayesian neural networks are more sensitive than the MLP model to amplification and attenuation perturbations, but in these cases the difference is less significant. While both models are robust on the peak cut alteration.

7.2 Data augmentation on BNN

In Chapter 5, we presented some techniques to improve the robustness of a regression model. Furthermore, we reported the results of their application on the original multilayer perceptron model. From this analysis we determined that the best technique, which has the best trade-off for the improvements in nominal and perturbed conditions, is data augmentation through the combination of mixture technique and basic manipulations (+BM), considering only alterations which cause up to 5% of MAPE. Therefore, we will apply the same technique with these Bayesian neural networks and, in order to make a correct comparison, we will use the same augmented dataset, generated for the multilayer perceptron model.

In particular, we have re-trained the aleatoric model (AL) and the models which combine aleatoric and epistemic uncertainty using both loss functions, “weighted-MAE” (AL&EP weightedMAE) and negative log-likelihood (AL&EP negloglik). The application of the data augmentation technique must not degrade the results on the nominal dataset. Therefore, for each BNN model, we compare the metrics obtained with the original and the re-trained models. In Tables 7.4 and 7.5 you can see the re-

sults for each original BNN model and its re-trained version for both current pO2 and pO2 at 37°C predictions.

	MAPE [%]	**OT [%]**	**TID [%]**	**StdDev**
AL	3.86	6.21	85.91	0.0069
AL augmented	3.79	5.68	86.69	0.0069
AL&EP weightedMAE	5.56 ± 0.86	15.43 ± 5.08	75.06	0.0069
AL&EP weightedMAE augmented	5.42 ± 0.69	14.29 ± 4.29	75.58	0.0069
AL&EP negloglik	4.42 ± 0.08	9.43 ± 0.59	98.33	0.013
AL&EP negloglik augmented	3.96 ± 0.027	5.98 ± 0.19	98.62	0.013

Table 7.4: Summary of results for current pO2 prediction for Bayesian neural network models re-trained with data augmentation technique

	MAPE [%]	**OT [%]**	**TID [%]**	**StdDev**
AL	3.56	4.82	91.24	0.0069
AL augmented	3.65	4.78	90.53	0.0069
AL&EP weightedMAE	5.48 ± 0.83	14.79 ± 4.99	80.23	0.0069
AL&EP weightedMAE augmented	5.38 ± 0.78	14.33 ± 4.86	80.96	0.0069
AL&EP negloglik	4.19 ± 0.11	7.99 ± 0.72	99.38	0.013
AL&EP negloglik augmented	3.71 ± 0.03	4.76 ± 0.25	99.38	0.013

Table 7.5: Summary of results for 37°C pO2 prediction for Bayesian neural network models re-trained with data augmentation technique

From Tables 7.4 and 7.5, you can notice that the data augmentation technique, in general, does not degrade the models' behavior in nominal conditions, indeed only

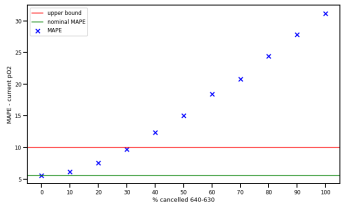
for the aleatoric model there are small worsening. The data augmentation technique allows improving the performances in nominal conditions as well, especially for the aleatoric and epistemic model trained with the negative log-likelihood loss function.

Figure 7.1 shows the comparison of robustness between the original model, which handles aleatoric and epistemic uncertainty, trained with the "weightedMAE" loss function, and the model re-trained with data augmentation technique. You can notice that in general there are improvements for each alteration; however, the most significant results regard the cut of the curve end and the clock offset, while the robustness to peak cut, amplification and attenuation have only small differences. The only worsening concerns the noise perturbation.

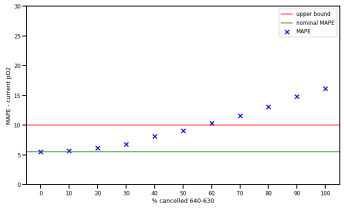
Instead, in Figure 7.2, you can see the comparison between the network which models aleatoric and epistemic uncertainty trained with the negative log-likelihood loss function and the same model re-trained with the data augmentation method. In this case, you can notice significant improvements for clock offset, attenuation and amplification, while for peak cut and cut of the curve end there are lower changes. Even in this case, the noise robustness becomes worse.

Differently, in Figure 7.3, you can see the comparison between the aleatoric model and the same model re-trained with the data augmentation technique. In this case, there are improvements in all alterations, for some of them are more significant, such as amplification and attenuation. Moreover, the model re-trained maintains its robustness on noise alteration as well.

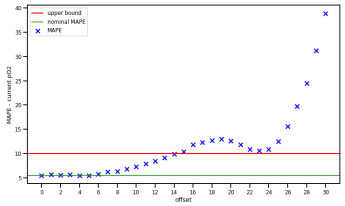
By comparing the three model robustness, we can conclude that the model which handles aleatoric and epistemic uncertainty trained with "weightedMAE" has fewer improvements both in nominal and in perturbed conditions than the same model trained with the negative log-likelihood loss function. In particular, this last model has the best improvements in nominal conditions, it is robust on amplification, attenuation and clock offset alterations; while it remains more sensitive on peak cut and cut of curve end. However, its noise robustness gets worse. On the contrary, the network which models only aleatoric uncertainty gets improvements for all the perturbations, without degrading the noise robustness. It is less robust to the cut of the end curve and clock offset than the other models, but it has better performances on amplification and attenuation.



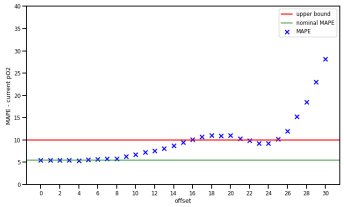
(a) Cut of the curve end original model



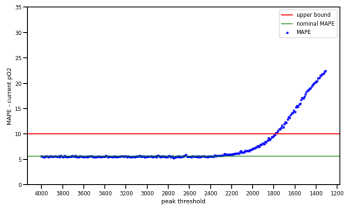
(b) Cut of the curve end augmented model



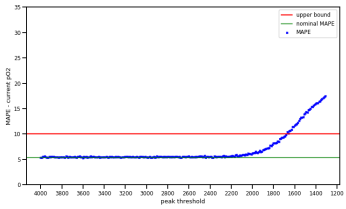
(c) Clock offset original model



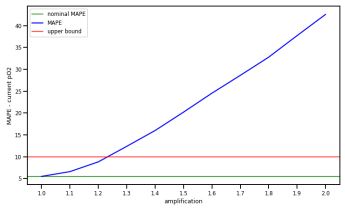
(d) Clock offset augmented model



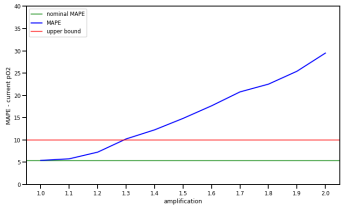
(e) Peak cut original model



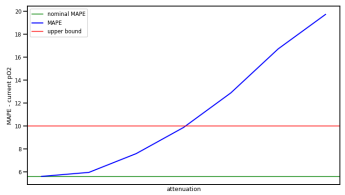
(f) Peak cut augmented model



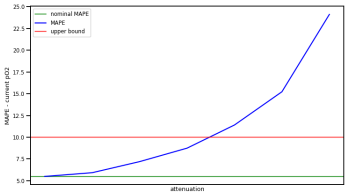
(g) Amplification original model



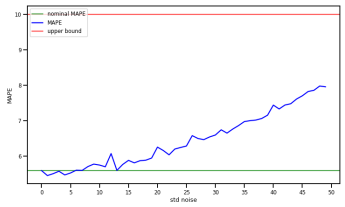
(h) Amplification augmented model



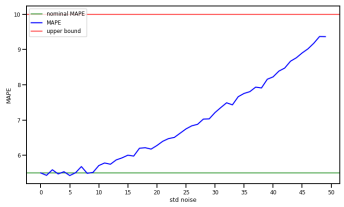
(i) Attenuation original model



(j) Attenuation augmented model

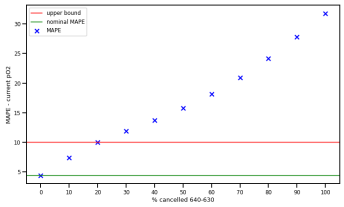


(k) Noise original model

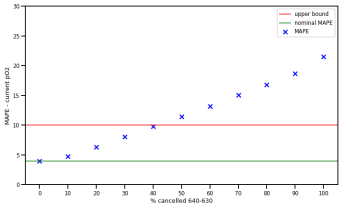


(l) Noise augmented model

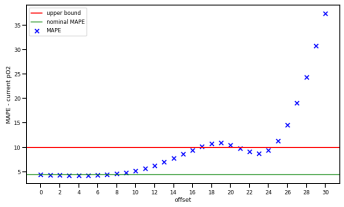
Figure 7.1: MAPE graphs for each perturbation applied to the aleatoric&epistemic model trained with “weightedMAE” loss



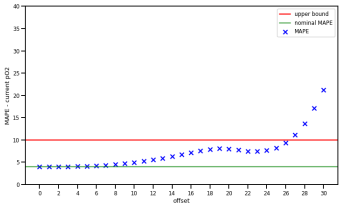
(a) Cut of the curve end original model



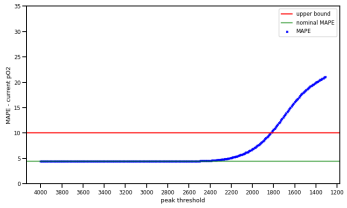
(b) Cut of the curve end augmented model



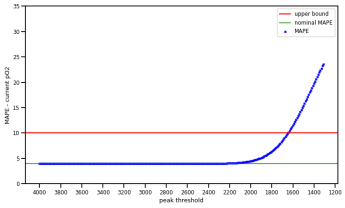
(c) Clock offset original model



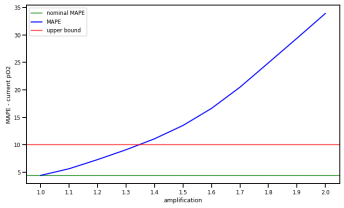
(d) Clock offset augmented model



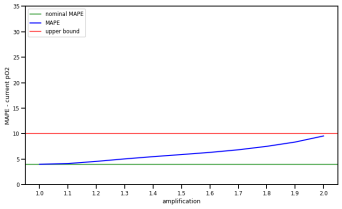
(e) Peak cut original model



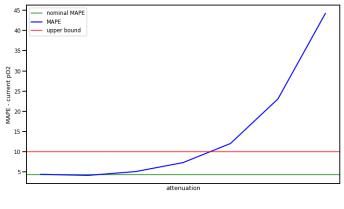
(f) Peak cut augmented model



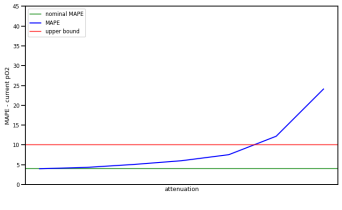
(g) Amplification original model



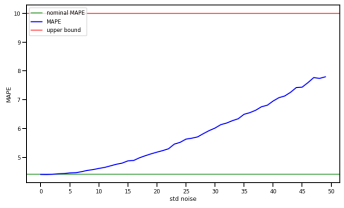
(h) Amplification augmented model



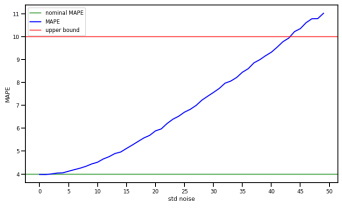
(i) Attenuation original model



(j) Attenuation augmented model

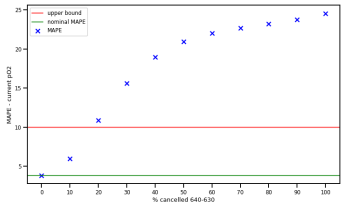


(k) Noise original model

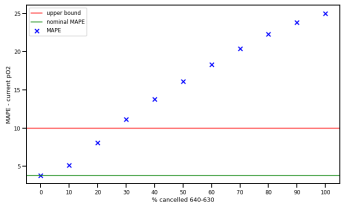


(l) Noise augmented model

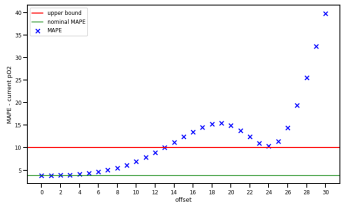
Figure 7.2: MAPE graphs for each perturbation applied to the aleatoric&epistemic model trained with negative log-likelihood loss



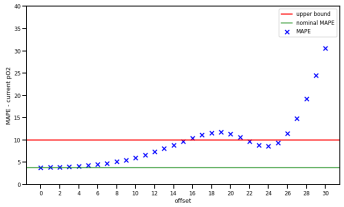
(a) Cut of the curve end original model



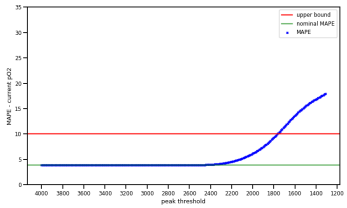
(b) Cut of the curve end augmented model



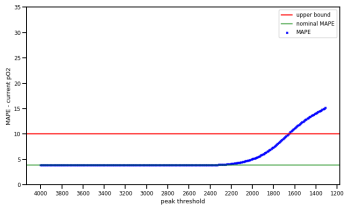
(c) Clock offset original model



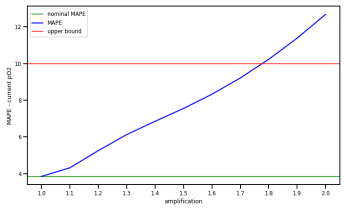
(d) Clock offset augmented model



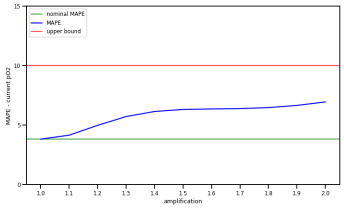
(e) Peak cut original model



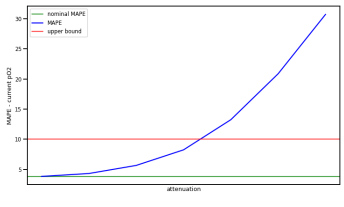
(f) Peak cut augmented model



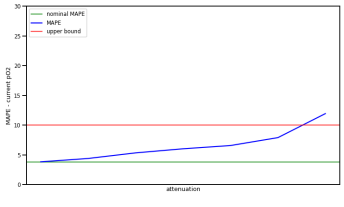
(g) Amplification original model



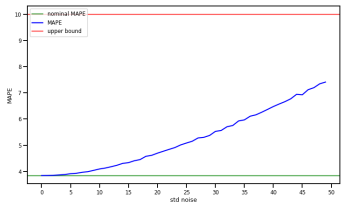
(h) Amplification augmented model



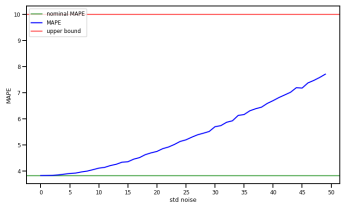
(i) Attenuation original model



(j) Attenuation augmented model



(k) Noise original model



(l) Noise augmented model

Figure 7.3: MAPE graphs for each perturbation applied to the aleatoric model trained with “weightedMAE” loss

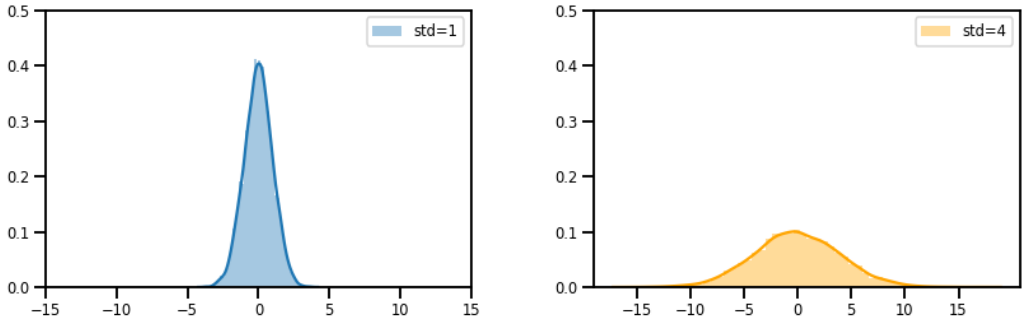
We can conclude that the data augmentation technique improves the performances for BNN models as well. In particular, using the negative log-likelihood loss function we can model both aleatoric and epistemic uncertainty with significant improvements both in nominal and perturbed conditions. Differently, modeling only aleatoric uncertainty we do not improve the nominal behavior which is affected only by small worsening, but it has the best performances in perturbed data.

By comparing the robustness of re-trained BNN models with the robustness of re-trained MLP models, you can notice that in general MLP models have better performances on these alterations both using the nominal and the re-trained models. While the main advantage of BNN is its robustness on noise.

In this section, we presented the robustness analysis for the three BNN models implemented. This analysis has been executed through the same process realized for MLP models, in order to make a proper comparison. Therefore, for the determination of the MAPE metric, we considered for each predicted distribution only its mean value. However, we could analyze the BNN model robustness taking into account the standard deviation of its distribution as well. In the following section, we will present different approaches to evaluating the performances of BNN models by considering the distribution predicted and we will use these novels metrics to determine the robustness.

7.3 Other metrics for BNN

In previous sections, the performances of a BNN model have been computed using the mean value of each predicted distribution to calculate the error metrics. This method evaluates the model through the same metrics used for the MLP models: MAPE and "Out of Threshold %". Moreover, we can determine the model robustness on perturbations using the same techniques implemented for the multilayer perceptron. This approach is the typical method to evaluate the performances of a Bayesian neural network and it is really useful to make a proper comparison with the MLP neural network; however, it does not consider properly the distribution. The main problem of this method is that two distributions with the same mean values, but different variances are considered in the same way, since we consider their mean values as predictions. In Figure 7.4 an example of this problem.



(a) stddev=1 (b) stddev=4

Figure 7.4: Example of normal distributions with mean 0 and different standard deviations

To overcome this issue, we evaluate the performances of a BNN model by determining the difference between the maximum likelihood in the distribution predicted and the likelihood related to the target value: lower differences generate a better performing model. In particular, to compare properly different models we compute the percentage of the difference between the maximum likelihood predicted and the likelihood of the target value. In Figure 7.5, there is an example of an output distribution used to compute this metric. In particular, we determine the maximum likelihood of the predicted distribution $p_m(x)$ (red line in the figure) and the likelihood related to the target value $p_t(x)$ (blue line in the figure), then we compute the percentage of the difference between the maximum likelihood and the true value likelihood:

$$\textit{Likelihood\_Difference} = \frac{p_m(x) - p_t(x)}{p_m(x)} * 100$$

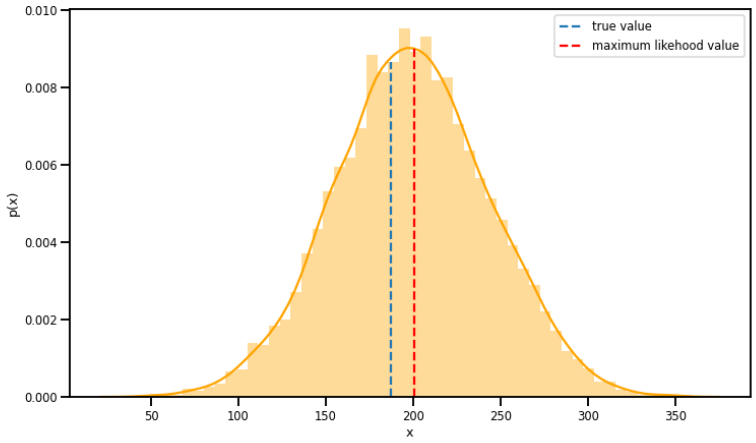


Figure 7.5: Example to compute the "% Likelihood Difference" metric

However, this metric is not enough to evaluate the model properly. Indeed, if the predicted distributions are uniform with high variances, the model will result in high performances, but in reality its predictions are not correct. In Figure 7.6 you can see an example of this issue: in this output distribution the value related to the maximum likelihood is 50, however the target value is 100. The probability related to both values is similar since the distribution predicted has high variance, therefore this model will be considered by the "% Likelihood Difference" metric as well-performing, even though its predictions are wrong.

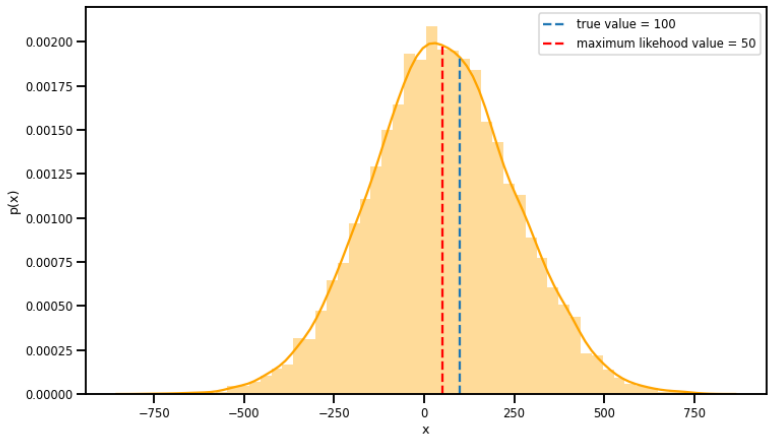
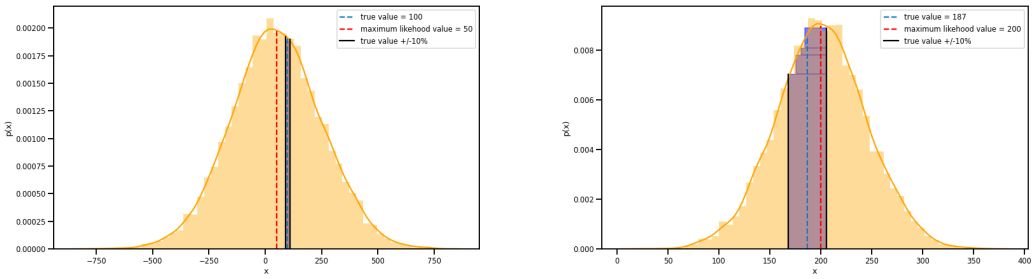


Figure 7.6: Example of the "% Likelihood Difference" metric issue

Therefore, we need to consider another metric that determines how much distribution is included in a specific range close to the target value. In this case study, since the accepted error between the target value and the prediction is up to $\pm 10\%$ of the true value, we evaluate the model by computing how much distribution is included between $\pm 10\%$ of the target value: more distribution is included in the range specified, better the model will be. Therefore, for each output distribution we determine the percentage of distribution included in this range:

$$\int_{target*0.9}^{target*1.1} p(x) * dx$$

In Figure 7.7, you can see two examples of this metric. In particular, the prediction in Figure 7.7b will have better results for the "Distribution in Range" metric, indeed the distribution included in the $\pm$ 10% range (purple area) is higher than that in Figure 7.7a.



(a) (b)

Figure 7.7: Examples of normal "Distribution in Range" metric

The combination of these two techniques allows evaluating the model by considering its distribution. In particular, we will favor distributions with low variance and mean value close to the target.

This analysis has been implemented on the same models analyzed in Section 7.1: the model which handles only the aleatoric uncertainty and models for both aleatoric and epistemic uncertainty trained with both "weightedMAE" and negative log-likelihood loss functions. In Table 7.6 you can see the results of these two metrics on the original BNN models and on models re-trained with data augmentation, combining mixture technique and basic manipulations.

	Likelihood Difference [%] [current pO2, 37° pO2]	**Distribution in Range [%]** [current pO2, 37° pO2]
AL	[58.91, 52.66]	[90.48, 91.12]
AL augmented	[59.44, 53.95]	[91.15, 91.02]
AL&EP weightedMAE	[70.56, 65.87]	[82.55, 82.92]
AL&EP weightedMAE augmented	[69.75, 65.40]	[83.70, 83.39]
AL&EP negloglik	[43, 36]	[82.76, 82.01]
AL&EP negloglik augmented	[39.96, 33.32]	[84.44, 83.55]

Table 7.6: Summary of results for current pO2 prediction for Bayesian neural network models with new metrics

From Table 7.6, we can notice that modeling both aleatoric and epistemic uncertainty, using the "weightedMAE" loss function (AL&EP weightedMAE) generates a

worse-performing model, especially for "% Likelihood Difference". Instead, training through the negative log-likelihood loss function (AL&EP negloglik) generates the best performances for "% Likelihood Difference" and good results for "Distribution in Range" as well. Both models improve with data augmentation technique. On the contrary, the aleatoric model (AL) has better performances than the AL&EP weightedMAE model, especially for the "Distribution in Range" metric, but the data augmentation technique does not improve its metrics.

The robustness analysis has been implemented through an approach similar to the framework presented in Chapter 3. In particular, to evaluate the correctness of the prediction we will use the two novel metrics: "% Likelihood Difference" and "Distribution in Range". We compute the robustness of each model on the different alterations defined in Section 4.3. For each test, we determine two graphs showing the variation of each metric with the increase of a specific alteration/perturbation of the input data. Each alteration is associated with the same interval chosen for MLP models. For the "% Likelihood Difference", we consider as upper bound a difference of 90% between the predicted maximum likelihood and the target value likelihood. In this case, the robustness will be determined by computing the area included between the upper bound and the alteration graph, delimited in the range chosen for each alteration:

$$\int_n^m max(90\% - \%\_Likelihood\_Difference(x), 0) * dx$$

This allows us to obtain a value that can be compared with the ideal situation, where the "% Likelihood Difference" is equal to the nominal value for each grade of alteration. In Figure 7.8 an example of the significant area to determine the robustness to each alteration for the "% Likelihood Difference" metric.

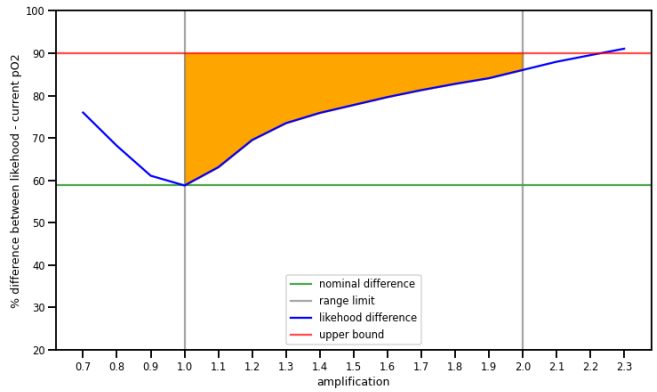


Figure 7.8: Example of the significant area of the "% Likelihood Difference" graph to evaluate the robustness

Differently, for the "Distribution in Range" metric we need to set a lower bound. Indeed, the increase of a perturbation on data causes the degradation of the percentage of distribution included in the prefixed range close to the target value. Therefore, the robustness will be determined by computing the area included between the alteration graph and the lower bound. Even in this case, we obtain a value that can be compared with the ideal situation, where the metric is equal to the nominal value for each grade of alteration. The lower bound can be chosen based on the different applications, in this analysis we used the limit of 50%. The classical error function has been computed through the following formula (where m and n represent the range limits for each specific alteration):

$$\int_{n}^{m} max(\textit{Distribution\_In\_Range}(x) - 50\%, 0) * dx$$

In Figure 7.9 you can see an example of the significant area for the robustness evaluation for the "Distribution in Range" metric.

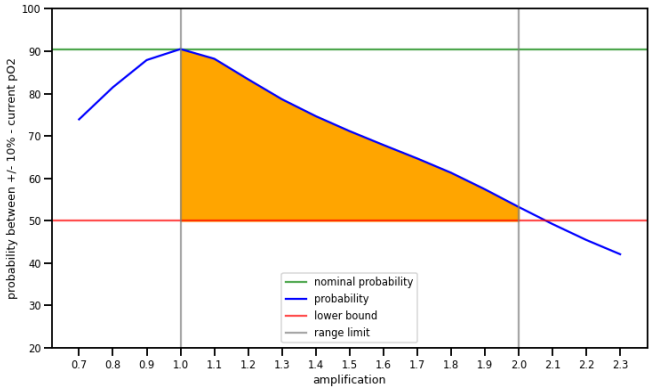


Figure 7.9: Example of the significant area of the "Distribution in Range" graph to evaluate the robustness

For each metric, the robustness has been evaluated, comparing the area in perturbed situations with the ideal one, through the linear weight function defined in Section 3.2. The same analysis could be executed using one of the other functions as well. In Tables 7.7 and 7.8 you can see the robustness results based on the two novel metrics for the three BNN models in analysis.

Alteration	Range	Robustness %		
		Aleatoric	Al&Ep weightedMAE	Al&Ep negloglik
Curve end cut	[641-620]	34.68	52.09	48.46
Clock offset	[0-30]	62.18	72.30	83.46
Peak cut	[4000-1300]	95.64	95.04	95.91
Amplification	[1-2]	60.64	37.85	54.54
Attenuation	[1-0.5]	69.29	62.62	80.83
Noise	[0-50]	84.64	90.92	88.41

Table 7.7: Summary of current pO2 robustness through the "% Likelihood Difference" metric using the linear weight function

Alteration	Range	Robustness %		
		Aleatoric	Al&Ep weightedMAE	Al&Ep negloglik
Curve end cut	[641-620]	38.98	55.69	43.32
Clock offset	[0-30]	60.12	68.04	75.91
Peak cut	[4000-1300]	94.60	94.37	93.91
Amplification	[1-2]	70.85	39.24	47.86
Attenuation	[1-0.5]	75.04	63.35	77.67
Noise	[0-50]	90.91	92.12	90.01

Table 7.8: Summary of current pO2 robustness through the "Distribution in Range" metric using the linear weight function

By observing Tables 7.7 and 7.8, you can notice that the aleatoric model is more robust on amplification and attenuation perturbations than the other models, but it has lower robustness on the other alterations. In general, the model for aleatoric and epistemic uncertainty, trained with the negative log-likelihood loss function, is more robust than the same model trained with the "weightedMAE" cost function.

The robustness results on different perturbations are similar to those obtained in Section 7.1, where we used the MAPE metric computed through the mean value of each distribution. Indeed, these models are robust on noise and peak cut perturbations,

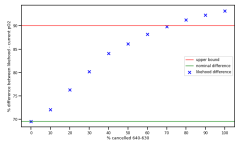
they get good results for amplification, attenuation and clock offset alterations, while they are really sensitive to curve end cut.

Moreover, we compare the robustness between each BNN model and the same model re-trained through the data augmentation technique (using the combination of the mixture technique and basic manipulations). In particular, Figure 7.10 shows the comparison between the robustness of the original aleatoric&epistemic model, trained with "weightedMAE" loss function, and the re-trained model robustness. You can see that computing the robustness through these two novel metrics we do not obtain significant improvements in any perturbation. On the contrary, as you can see from Figure 7.1, using the MAPE metric to evaluate the robustness of this model we obtain more significant improvements. Instead, in Figure 7.11, you can see the comparison of robustness between aleatoric&epistemic models trained with the negative log-likelihood loss function. In this case, data augmentation gets significant improvements for both novel metrics, especially for amplification, attenuation and clock offset perturbations. The comparison with Figure 7.3 highlights that we obtain similar robustness results using both the classical MAPE metric, computed through the mean value of each distribution and the novel metrics. Similarly, in Figure 7.12 you can see the comparison of graph alteration for each perturbation on the aleatoric models. Data augmentation allows us to obtain improvements mainly in the amplification, attenuation and clock offset alterations; the noise robustness does not degrade, while we do not obtain significant improvements for the cut of the curve end and the peak cut. Comparing Figures 7.12 and 7.3, we can notice that also in this case we get similar robustness results using both the classical MAPE metric and the novel metrics.

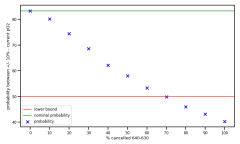
To conclude, all models have good results for the "Distribution in Range" metric, therefore the distributions predicted have low variances, as you can see from Table 7.6. Furthermore, using the negative log-likelihood cost function we get good performance for the "% Likelihood Difference" metric as well. From the robustness evaluation, we notice that through these two novel metrics, "% Likelihood Difference" and "Distribution in Range", we obtain similar robustness results especially for the aleatoric model and the network for aleatoric and epistemic uncertainty trained with the negative log-likelihood loss function. In particular, training the BNN models with the negative log-likelihood loss function generates the best performances both in nominal and in perturbed conditions, especially for the "% Likelihood Difference" metric.

This analysis highlights the importance to train a BNN using the negative log-likelihood, a cost function that updates the parameters minimizing the negative likelihood between the predictions and the target values. In addition, we presented two novel metrics to evaluate these networks taking into account the predicted distributions. The main advantage is that we can evaluate the robustness through the same

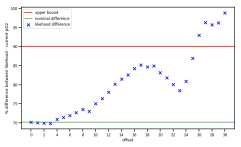
framework implemented for MLP models, considering in a better way the predicted distribution.



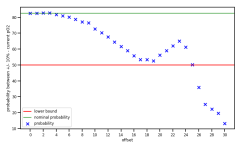
(a) Cut of the curve end original model



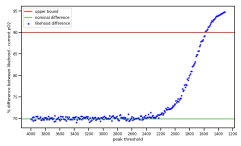
(b) Cut of the curve end augmented model



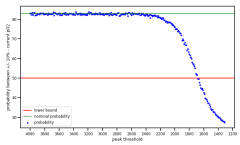
(c) Clock offset original model



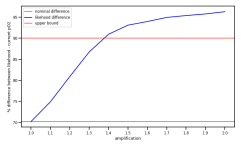
(d) Clock offset augmented model



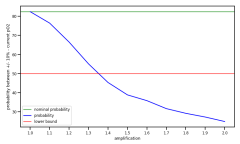
(e) Peak cut original model



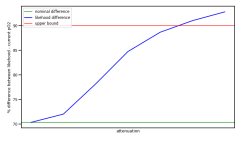
(f) Peak cut augmented model



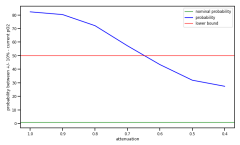
(g) Amplification original model



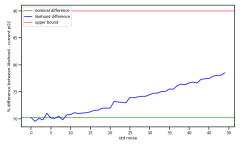
(h) Amplification augmented model



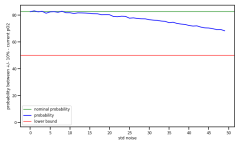
(i) Attenuation original model



(j) Attenuation augmented model

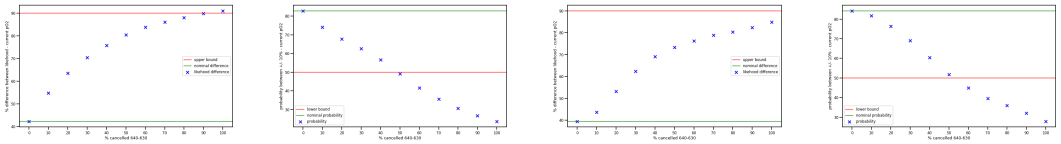


(k) Noise original model



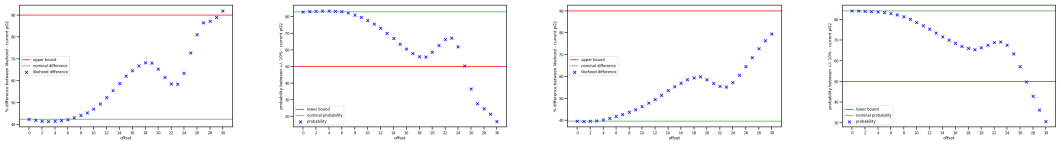
(l) Noise augmented model

Figure 7.10: Alterations graphs for each perturbation applied to the aleatoric&epistemic model trained with “weightedMAE” loss using the novel robustness metrics



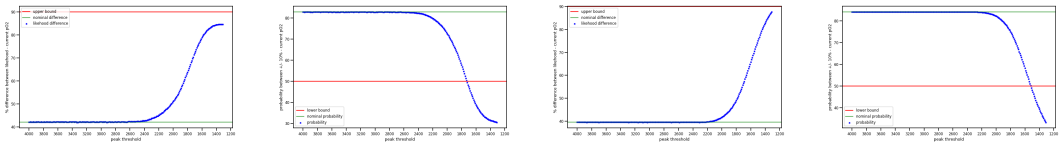
(a) Cut of the curve end original model

(b) Cut of the curve end augmented model



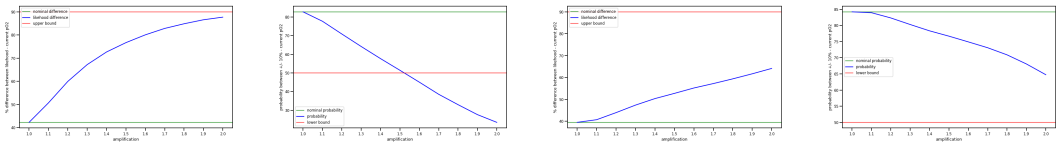
(c) Clock offset original model

(d) Clock offset augmented model



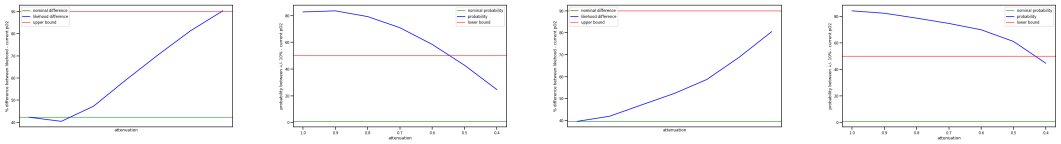
(e) Peak cut original model

(f) Peak cut augmented model



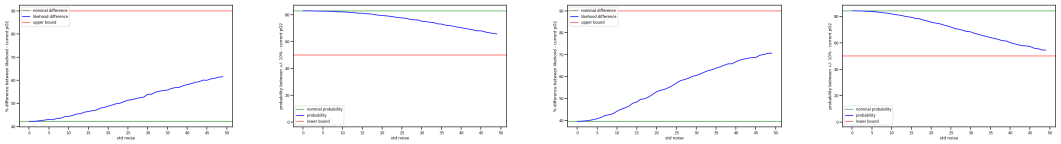
(g) Amplification original model

(h) Amplification augmented model



(i) Attenuation original model

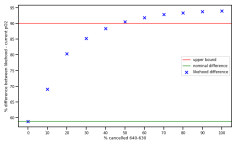
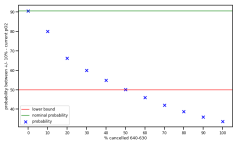
(j) Attenuation augmented model



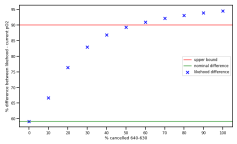
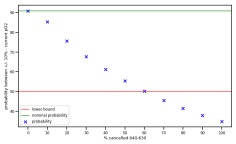
(k) Noise original model

(l) Noise augmented model

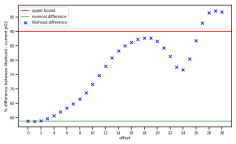
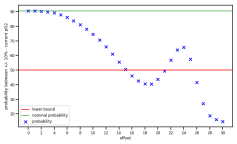
Figure 7.11: Alterations graphs for each perturbation applied to the aleatoric&epistemic model trained with negative log-likelihood loss using the novel robustness metrics

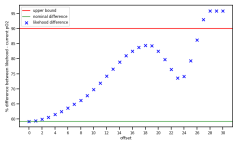
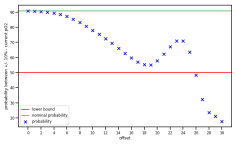
(a) Cut of the curve end original model

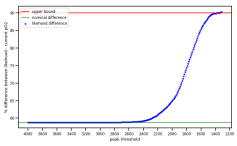
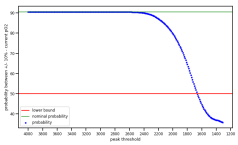
(b) Cut of the curve end augmented model

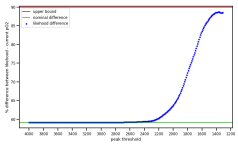
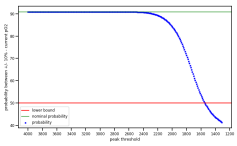
(c) Clock offset original model

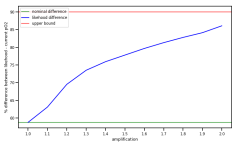
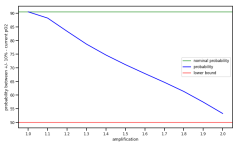
(d) Clock offset augmented model

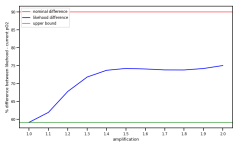
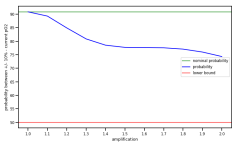
(e) Peak cut original model

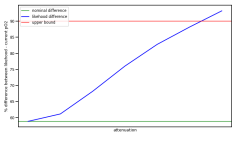
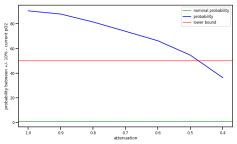
(f) Peak cut augmented model

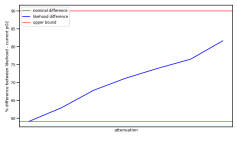
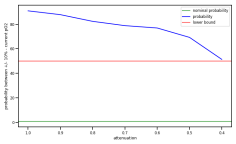
(g) Amplification original model

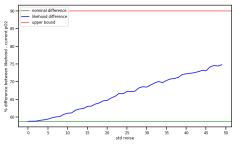
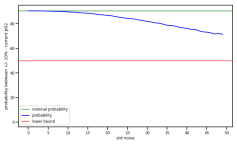
(h) Amplification augmented model

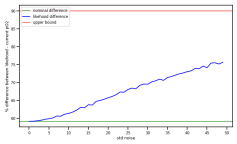
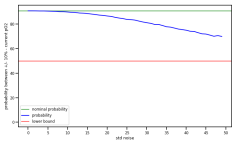
(i) Attenuation original model

(j) Attenuation augmented model

(k) Noise original model

(l) Noise augmented model

Figure 7.12: Alterations graphs for each perturbation applied to the aleatoric model using the novel robustness metrics

Chapter 8

Adversarial robustness

This chapter presents an introduction to one of the most important research problems in deep learning models: vulnerability to adversarial examples. In Chapters 3 and 4 we analyzed the robustness of the MLP model on natural inputs, new data generated by altering the original data with perturbations that could happen in reality. Instead, this chapter aims to discuss the robustness of the model on adversarial inputs: input data created to cause the model to make a mistake. The adversarial robustness has been performed with a common adversarial attack: Fast Gradient Sign Method. Later in this chapter, we will present a comparison of the adversarial robustness between the original model and the models re-trained to improve the basic robustness. Furthermore, the same analysis will be performed using Bayesian neural networks as well.

8.1 State of the art

Machine learning is being applied in many safety-critical environments. However, machine learning models are in general vulnerable to well-designed input samples: adversarial examples. Adversarial examples are data generated to cause the failure of a machine learning model. The differences between a right input data and an adversarial one are imperceptible to humans, but can easily fool the neural network. This vulnerability becomes one of the major risks of applying machine learning models in safety-critical environments [26].

Adversarial attacks can be classified, based on their knowledge of the system, into:

- White-box attacks: the adversary knows everything related to the model (model architecture, hyper-parameters, number of layers, activation functions, loss, weights);
- Black-box attacks: the adversary has no access to the model information.

Attacks can also be divided into targeted attacks, which misguide the model to a specific output (they are usually applied to multi-class classification problems), and non-targeted attacks, which do not assign a specific target to the neural network output. In addition, we can distinguish attacks, based on their frequency, into one-time attacks, which take only one step to generate the adversarial example and iterative attacks which take multiple steps to update the adversarial.

Adversarial attack and defense methods have been implemented mainly for classification problems, while few works investigate the regression case. However, there are some methods that can be applied to regression problems: Fast Gradient Sign Method and Projected Gradient Descent [16].

Fast Gradient Sign Method (FGSM) is a method proposed by Goodfellow et al. [8] to generate adversarial examples. It is a white-box method, indeed it has all information on the model, its weights and the training data. Furthermore, it is a non-targeted attack method, based on a single step. This method determines the direction of the attack by computing, for each input sample, the sign of the gradient on the loss function applied to it. The adversarial examples are determined with the following formula:

$$\tilde{x} = x + \epsilon * sign(\nabla_x J(\theta, x, y))$$

where x is the features vector, y is the target value, J the cost function and θ the model parameters. Instead, ϵ is the magnitude of the perturbation, which controls the attack strength.

The Projected Gradient Descent method is a variant of FGSM. It is a multi-step white-box non-targeted method. The attack is defined recursively through this iteration:

$$x^0 = x$$

$$x^{q+1} = clip_{x,\rho}(x^q) + \epsilon * sign(\nabla_x J(\theta, x, y))$$

$$\tilde{x} = x^Q$$

Where $clip_{x,\rho}(x^a)$ is the element-wise clipping of x^a to [x$-\rho$, x+ρ].

Since Projected Gradient Descent depends strictly on FGSM we perform the adversarial robustness evaluation using only FGSM.

As you can see from previous methods, the generation of adversarial examples depends on the application of small perturbations. In particular, in classification problems we consider adversarial examples all perturbed input data, close to the original samples so that the difference is imperceptible to a human, which causes an incorrect classification, an example in Figure 8.1.

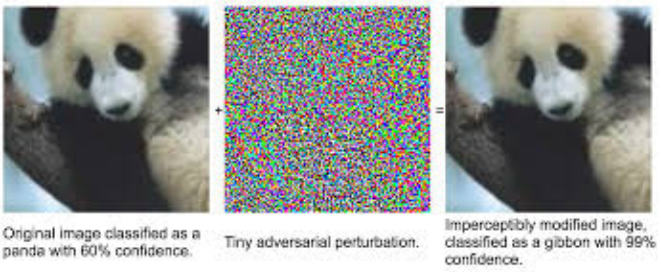


Figure 8.1: Example of adversarial example for classification neural networks

Similarly, in regression problems, adversarial examples are still close to the original samples, as you can see in Figure 8.2.

Target: ***[146., 125.]***

Original input: [32.2 , 1782.1 , 1398.6499, 946.75 , 564.25 , 219.]
Original normalized input: [0.664151 , 0.51569664, 0.51651675, 0.44204903, 0.37579012, 0.29418087]

Original output: ***[144.72557, 124.80188]***

Perturbation: [1, 1, 1, -1, 1, -1] * 0.001

Perturbed normalized input : [0.665151 , 0.51669663, 0.51751673, 0.44104904, 0.3767901 , 0.29318088]
Perturbed input : [32.2265, 1784.301 , 1400.601 , 945.1775, 565.4602, 218.2559]

New output: ***[139.61649, 121.01311]***

Figure 8.2: Example of adversarial example for regression problems using FGSM with $\epsilon = 0.001$

However, in this case study, the prediction is a single value, therefore instead of checking the wrong classification, the adversarial robustness concerns the observation of numerical stability. The numerical instability of an algorithm represents the degree to which a model's output may change with changes in the input [16]. This method depends on the basic idea that small perturbations on the input data should cause small changes in the output. This concept does not apply to all regression problems, since there could be tasks where this kind of change is really related to different output values. However, in this case study, we noticed from the domain analysis, presented in Section 4.2, that the same target output can cause different responses to the bright pulse, based on the spotlight and probe. Therefore, for this problem the basic idea that small perturbations on the input should not drastically change the output prediction is verified. Thus, the evaluation of adversarial robustness has been implemented considering how much the MAPE metrics change when we apply a FGSM attack. The adversarial robustness has been evaluated for both the multilayer perceptron model and the BNN models; in the following sections, you can see the results obtained.

8.2 FGSM attack on MLP

The first implementation of the FGSM algorithm has been performed to evaluate the adversarial robustness of standard multilayer perceptron models. In particular, we compute this robustness both on the original model and on the new models trained through the data augmentation technique. FGSM algorithm has been implemented with the following code, as explained in [24].

```
loss_object = weightedMAE
# Choose the perturbation magnitude
eps = 0.0001

def create_adversarial_pattern(input_data, input_label):
  with tf.GradientTape() as tape:
  tape.watch(input_data)
  prediction = model(input_data)
  loss = loss_object(input_label, prediction)

  # Get the gradients of the loss
  gradient = tape.gradient(loss, input_data)
  # Get the sign of the gradients to create the perturbation
  signed_grad = tf.sign(gradient)

  return signed_grad


perturbations = create_adversarial_pattern(x_test, y_test)
adv_x = x_test + eps*perturbations
# Maintain the new samples in the original input range
adv_x = tf.clip_by_value(adv_x, 0, 1)
```

The basic idea of this method consists of the application of small perturbations on input data. Therefore, we apply FGSM using different ϵ, magnitude of perturbation: $10^{-4}, 5 * 10^{-4}, 10^{-3}, 2 * 10^{-3}$. In this way, we can compare how the increase of the perturbation magnitude changes the error metrics. Adversarial examples have been computed using the test dataset, composed of 4330 samples.

In Table 8.1, you can see the MAPE error and the "Out of Threshold %" metric on the adversarial examples dataset obtained from the original MLP model, using different perturbation degrees. Furthermore, we calculate their percentage of increase with respect to the metrics computed on the original test dataset without any perturbation.

This allows us to compare more easily the adversarial robustness of different models, in fact considering only the nominal increase of error metrics does not highlight properly if a model is more or less robust since it depends also on the error metrics on the original test dataset. For example, a model could have lower errors on adversarial examples only because the error on the original test dataset is lower, but the difference between original error and error on adversarial examples could be higher.

From Table 8.1, we can notice that with really small perturbations ($\epsilon = 10^{-4}$) the error metrics have only small increases, however when we overtake this limit the error increase rapidly. The main worsening regards "Out of Threshold %" metrics, indeed on the original dataset we have only around 5% of out of threshold predictions, but applying perturbations with $\epsilon = 2 * 10^{-3}$ almost 40% of samples are out of threshold.

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]
0 (Original)	[3.70, 3.35]		[5.17, 3.60]	
10^{-4}	[3.94, 3.63]	[6.48, 8.35]	[5.70, 4.04]	[10.25, 12.22]
$5 * 10^{-4}$	[4.93, 4.79]	[33.24, 42.98]	[8.61, 7.04]	[66.53, 95.55]
10^{-3}	[6.20, 6.26]	[67.56, 86.86]	[14.36, 13.62]	[177, 278]
$2 * 10^{-3}$	[8.81, 9.23]	[138, 175]	[31.80, 38.70]	[515, 975]

Table 8.1: Robustness to adversarial examples generated through the original model

In addition, we perform the same analysis using some models re-trained with the different augmented datasets presented in Chapter 5: augmentation with mixture technique and basic manipulations, using altered data which cause up to 5% of MAPE (+BM), augmentation with mixture technique, basic manipulation and additive white noise (+AWN) and augmentation with only additive white noise (only noise). In Tables 8.2, 8.3 and 8.4, you can see all the results with these three models, respectively +BM, +AWN and only noise, using the same perturbation magnitudes as before.

By comparing Tables 8.1 and 8.2, you can see that the model re-trained with the augmented dataset, generated through mixture technique and basic manipulations, is less robust on adversarial examples than the original one. In particular, the nominal errors for small perturbations are lower since this model has higher performances on the original test dataset than the original model; however comparing the percentage of increase you can see that errors rise more rapidly, especially for the "Out of Threshold %" metric. In addition, with higher perturbations, such as $\epsilon = 2 * 10^{-3}$, all metrics are worse than the original model. By comparing Tables 8.1 and 8.3 we get similar conclusions. In this case, however, the increase of both error metrics is

lower than using the +BM model (Table 8.2). Therefore, data augmentation with the combination of mixture technique, basic manipulations and noise injection generates a more robust model against adversarial examples than using the model re-trained with the dataset augmented only through mixture technique and basic manipulations. The +AWN model has anyway lower adversarial robustness than the original model. Since the model +AWN performs better on adversarial attacks than the +BM, we decided to implement the same analysis using the model re-trained with the augmented dataset based on noise injection, indeed the noise injection is the only difference between +AWN and +BM data augmentation process. From Table 8.4, we can see that data augmentation with noise injection allows training a model that is more robust to adversarial examples than the original one. In particular, there are improvements both in the percentage of increase and in the nominal values of all metrics.

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]
0 (+ BM)	[3.12, 3.06]		[2.21, 3.03]	
$\mathbf{10^{-4}}$	[3.39, 3.31]	[8.65, 8.16]	[2.65, 3.44]	[19.90, 13.53]
$\mathbf{5 * 10^{-4}}$	[4.50, 4.35]	[44.23, 42.15]	[5.15, 5.81]	[133, 91.74]
$\mathbf{10^{-3}}$	[5.97, 5.74]	[91.34, 87.58]	[10.16, 11.29]	[359, 272]
$\mathbf{2 * 10^{-3}}$	[9.34, 8.74]	[199, 185]	[33.37, 34.82]	[1409, 1049]

Table 8.2: Adversarial robustness on samples generated through the +BM model

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]
0 (+ AWN)	[3.11, 3.06]		[2, 2.79]	
$\mathbf{10^{-4}}$	[3.35, 3.29]	[7.71, 7.51]	[2.31, 3.16]	[15.5, 13.26]
$\mathbf{5 * 10^{-4}}$	[4.31, 4.24]	[38.58, 38.56]	[4.24, 5.21]	[112, 86]
$\mathbf{10^{-3}}$	[5.61, 5.50]	[80.38, 79.73]	[8.70, 9.88]	[335, 254]
$\mathbf{2 * 10^{-3}}$	[8.56, 8.23]	[175, 168]	[25.35, 28.31]	[1167, 914]

Table 8.3: Adversarial robustness on samples generated through the +AWN model

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]
0 (only noise)	[3.12, 3.10]		[2.35, 2.60]	
$\mathbf{10^{-4}}$	[3.29, 3.26]	[5.44, 5.16]	[2.60, 2.97]	[10.63, 14.23]
$\mathbf{5 * 10^{-4}}$	[3.98, 3.92]	[27.56, 26.45]	[3.69, 4.59]	[57, 76]
$\mathbf{10^{-3}}$	[4.83, 4.76]	[54.80, 53.54]	[5.81, 6.46]	[147, 148]
$\mathbf{2 * 10^{-3}}$	[6.62, 6.51]	[112, 110]	[13.67, 14.84]	[481, 470]

Table 8.4: Adversarial robustness on samples generated through the "only noise" model

To conclude, the original model is robust on adversarial attacks, generated with the FGSM technique, only for small perturbation degrees; while high perturbations cause a high increase especially for the "Out of Threshold %" metrics. Furthermore, in general, models re-trained through data augmentation techniques, which improve the basic robustness, as presented in Section 5.2, do not obtain improvements in the adversarial robustness, as shown with +BM and +AWN models. Only executing data augmentation with the noise injection technique, we can get a better performing model on adversarial attacks. However, the "only noise" has been trained to improve the robustness only on additive white noise injection, therefore it is not robust on basic manipulations.

We can say that in multilayer perceptron models trained for this case study, we cannot improve the basic and the adversarial robustness at the same time. Based on the specific robustness requirements, we have to choose between a model that has better basic robustness, in this case we will choose the +BM model, or a model robust on adversarial attacks and noise injection, where "only noise" model is the best choice.

In the next section, we will present the same analysis on Bayesian neural networks.

8.3 FGSM attack on BNN

FGSM algorithm has been applied to evaluate adversarial robustness on Bayesian neural networks as well. In particular, the adversarial robustness evaluation has been computed for the aleatoric model and for the models which combine aleatoric and epistemic uncertainty. We implement the FGSM algorithm using the same function presented in the previous section, the only difference concerns that the outputs of these models are distributions, therefore we consider their mean values to compute the different metrics, as explained in Section 6.4. The robustness has been evaluated by applying the same

perturbation degrees used for the MLP analysis: $10^{-4}, 5 * 10^{-4}, 10^{-3}, 2 * 10^{-3}$. Furthermore, as before, the adversarial examples have been generated by applying these perturbations on the original test dataset, which is composed of 4330 samples.

In Table 8.5 there are the results about adversarial robustness of the network which models aleatoric and epistemic uncertainty, trained with "weightedMAE" loss function.

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]	**TID [%]** [current pO2, 37° pO2]
0 (Al&Ep weightedMAE)	[5.56 ± 0.86, 5.48 ± 0.83]		[15.43 ± 5.08, 14.79 ± 4.99]		[75.06, 80.23]
10^{-4}	[5.65 ± 0.83, 5.56 ± 0.81]	[1.61, 1.46]	[15.91 ± 5.03, 15.22 ± 5.03]	[3.11, 2.90]	[74.28, 79.65]
$5 * 10^{-4}$	[5.95 ± 0.71, 5.81 ± 0.74]	[7.01, 6.02]	[17.33 ± 4.46, 16.47 ± 4.70]	[12.31, 11.35]	[71.21, 77.40]
10^{-3}	[6.49 ± 0.68, 6.23 ± 0.69]	[16.7, 13.68]	[20.18 ± 4.61, 18.71 ± 4.58]	[30.78, 26.50]	[66.21, 73.99]
$2 * 10^{-3}$	[7.60 ± 0.58, 7.22 ± 0.55]	[36.69, 31.75]	[26.47 ± 3.79, 24.57 ± 3.90]	[71.54, 66.12]	[56.48, 66.07]

Table 8.5: Robustness to adversarial examples generated through Bayesian neural network which models aleatoric and epistemic uncertainty trained with "weightedMAE" loss function

Instead, in Table 8.6 you can see the robustness for the model which handles aleatoric and epistemic uncertainty trained with the negative log-likelihood loss function.

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]	**TID [%]** [current pO2, 37° pO2]
0 (Al&Ep negloglik)	[4.42 ± 0.08, 4.19 ± 0.11]		[9.43 ± 0.59, 7.99 ± 0.72]		[98.33, 99.38]
10^{-4}	[4.58 ± 0.08, 4.38 ± 0.11]	[3.61, 3.34]	[9.95 ± 0.59, 8.42 ± 0.72]	[5.51, 5.38]	[98.10, 99.26]
$5 * 10^{-4}$	[5.22 ± 0.08, 4.93 ± 0.11]	[18.09, 17.66]	[12.03 ± 0.58, 10.34 ± 0.8]	[27.57, 29.41]	[96.88, 98.67]
10^{-3}	[6.04 ± 0.88, 5.71 ± 0.11]	[36.65, 36.27]	[15.05 ± 0.63, 13.25 ± 0.97]	[59.59, 65.83]	[94.62, 97.70]
$2 * 10^{-3}$	[7.69 ± 0.088, 7.31 ± 0.11]	[73.98, 74.46]	[23.92 ± 0.79, 21.32 ± 1.03]	[153, 166]	[87.50, 93.13]

Table 8.6: Robustness to adversarial examples generated through Bayesian neural network which models aleatoric and epistemic uncertainty trained with "negative log-likelihood" loss function

The same analysis has been performed with the network which models only aleatoric uncertainty, in Table 8.7 you can see its robustness results.

ϵ	**MAPE [%]** [current pO2, 37° pO2]	**MAPE variation [%]** [current pO2, 37° pO2]	**OT [%]** [current pO2, 37° pO2]	**OT variation [%]** [current pO2, 37° pO2]	**TID [%]** [current pO2, 37° pO2]
0 (Aleatoric)	[3.86, 3.56]		[6.21, 4.82]		[85.91, 91.24]
10^{-4}	[4.01, 3.69]	[3.88, 3.65]	[6.48, 5.12]	[4.34, 6.22]	[84.89, 90.57]
$5 * 10^{-4}$	[4.64, 4.24]	[20.20, 19.10]	[8.36, 6.58]	[34.62, 36.51]	[79.81, 87.02]
10^{-3}	[5.42, 4.96]	[40.41, 39.32]	[10.78, 8.68]	[73.59, 80.08]	[72.63, 81.73]
$2 * 10^{-3}$	[7.00, 6.44]	[81.34, 80.89]	[18.52, 14.78]	[198, 206]	[55.95, 68.98]

Table 8.7: Robustness to adversarial examples generated through Bayesian neural network which models aleatoric uncertainty trained with "weightedMAE" loss function

Observing Tables 8.5, 8.6 and 8.7, you can notice that Bayesian neural networks are robust to adversarial attacks especially with small perturbation magnitudes. The comparison of Table 8.5 and 8.6 shows that using the "weightedMAE" loss function generates a more robust model to adversarial attacks than using the negative log-likelihood one, for the MAPE and "Out of Threshold %" metrics. However, the model trained with negative log-likelihood is better performing and it is more robust to adversarial attacks for the TID metric. Furthermore, the results obtained modeling aleatoric and epistemic uncertainty through the negative log-likelihood are similar to those related to the aleatoric model.

By comparing Tables 8.5, 8.6 and 8.7 with 8.1, 8.2, 8.3 and 8.4, you can see that in general Bayesian neural networks are more robust on adversarial attacks than multilayer perceptron models. In particular, for BNN models we get a lower percentage of increase in the different metrics and for high perturbation, such as $2 * 10^{-3}$, we obtain better nominal results as well. The best improvements regard the "Out of Threshold %" metric.

Chapter 9

Conclusion

This work aimed to improve the safety of a developing medical system to predict the blood pO2 values analyzing the robustness of the multilayer perceptron used in the system. In particular, we explored many different techniques to evaluate and improve the robustness both on natural and on adversarial inputs. In addition, we implemented a different type of neural network: Bayesian neural networks, which provide directly the uncertainty of the prediction, and we compared their robustness with the original models. In this last chapter, we will summarize the main results obtained with previous analyses and we will briefly discuss possible future works on this topic.

9.1 Results summary

Through this project, we determined a useful testing framework to evaluate the robustness of a network for regression problems (Section 4.1). Furthermore, we defined some robustness functions which we can be applied with different neural networks for regression problems and we explained the best function to use based on the case study, as presented in Section 3.2. The implementation of this robustness analysis provided information about the behavior of the multilayer perceptron model in perturbed conditions and highlights some critical situations. In particular, the main issues in our case study are the cut of the curve end, which can be caused by anomalies in data acquisition, amplification and attenuation of the curve, which depend on the spots and probes applied in the system. Especially, attenuation and amplification of the signal are really common in this case study since they depend strictly on the other components. An important result regards the robustness to noise, indeed we obtain good performances for small noises: white noises with standard deviations up to 10 cause small changes in error metrics. Instead, injecting higher noises provokes significant error degradation.

In order to improve the robustness of regression networks, as the model in analy-

sis, we presented some novel techniques. In particular, we defined new interpretations of common data augmentation methods, usually applied for computer vision problems: mixture technique, basic manipulations and noise injection. Furthermore, we determined a new version of the incremental learning technique which can be applied to regression models. The application of these techniques generates new models with better performances both in nominal and in perturbed conditions. From the comparison of these models, we conclude that using data augmentation methods you can get better results, especially in nominal conditions. Even though all techniques improve the performances, the model with the best trade-off between robustness on perturbed data and performances in nominal behavior is the model re-trained using the combination of mixture technique and basic manipulations (by applying only small perturbations). All these conclusions can be seen in Section 5.3.

In machine learning for medical systems, but also in many other domains, an interesting aspect is the uncertainty of the estimation provided. For this reason, we implemented a different type of neural network: the Bayesian neural network. As explained in Chapter 6, these neural networks model the uncertainty of the data (aleatoric uncertainty), but also the uncertainty about the model (epistemic uncertainty). Instead of determining the model weights which minimize a loss function, as the classical neural, the BNNs aim to determine the best posterior distribution for each parameter. In general, the output of these networks is not a single value prediction, but it is a normal distribution. For this case study, we implemented four Bayesian neural networks, using the variational inference technique: to model the aleatoric uncertainty, to model the epistemic uncertainty, to model both the aleatoric and the epistemic uncertainty with "weightedMAE" and negative log-likelihood loss functions. Firstly we evaluated these models through the same metrics used for the standard MLP models, considering as predictions the mean value of each predicted distribution: MAPE and "Out of Threshold %". In addition, in order to take into account the distribution predicted, we compute an additive metric to determine the percentage of predicted distributions which include the target value (TID). We determined that modeling only the aleatoric uncertainty we get the best performances for MAPE and "Out of Threshold %" metrics. Instead, the model which combines aleatoric and epistemic uncertainty, trained with the negative log-likelihood loss function, gets the best results for the TID metric, it has lower performances on MAPE and "Out of Threshold %" than the aleatoric model, but better than the other models. In general, in nominal conditions, BNN models are worse performing than the multilayer perceptron model, because of many parameters to train, however they provide the uncertainty related to the prediction which can be really useful in many applications. Through the MAPE metric, we analyzed the robustness of the most significant BNNs: the model which considers the uncertainty on data

(aleatoric model) and the two models which take into account both the aleatoric and the epistemic uncertainty, trained with the two different loss functions. Evaluating the robustness on the same perturbations applied to MLP models, we determined that these BNN models have similar robustness on these alterations. The main difference regards the aleatoric model which is more robust to amplification and attenuation perturbations. The comparison of the robustness between MLP and BNN models highlights that BNNs are more robust on noise perturbations, with significant noises as well, while they are more sensitive to clock offset alterations. In general, both models have similar robustness to the other perturbations. To improve BNN behavior, we applied the best data augmentation technique determined in the previous analysis. This method is useful for BNNs as well and you can get the best improvements, both in nominal and in perturbed conditions, by modeling both the aleatoric and epistemic uncertainty through the negative log-likelihood loss function.

These techniques have been applied considering only the mean value of the distribution predicted. However, this method is affected by the problem that two distributions with the same means, but difference variances are considered in the same way. To overcome this issue, we need to use different metrics that consider properly the predicted distribution. In this work, we defined two novel metrics for this purpose (Section 7.3): "% Likelihood Difference" and "Distribution in Range". The first metric evaluates how the likelihood of the target is close to the maximum likelihood value in the predicted distribution, while the second one computes the percentage of probability included in a range close to the target (in this case study $\pm 10\%$). The combination of these metrics favors distributions with the mean value close to the target and small variances. Through these new metrics, we determined that the best performing model is the aleatoric&epistemic model trained with the negative log-likelihood loss function. This model has better results, especially for the "% Likelihood Difference", both in nominal and in perturbed conditions. For this model, the robustness results are similar to those obtained through the classical MAPE. Furthermore, the data augmentation technique provides more significant improvements both in nominal behavior and in the robustness only for this model. We can conclude that the best way to train a BNN is to maximize the likelihood between the prediction and the target value directly in the training phase. Therefore, the best approach consists of using the negative log-likelihood cost function or another loss with similar behavior.

Furthermore, we analyzed the robustness to adversarial attacks both on MLP and BNN models. To perform this analysis we implemented a classical method to generate adversarial examples: FGSM. We noticed that the original model is robust on adversarial attacks only for small perturbation degrees. In general, the models re-trained through data augmentation techniques are more sensitive to adversarial attacks than the

original model. However, the model generated with the noise injection data augmentation technique improves the adversarial robustness. Thus, in multilayer perceptron models trained for this case study, we cannot improve basic and adversarial robustness at the same time.

The application of adversarial attacks on BNN highlights that the model which combines aleatoric and epistemic uncertainty trained with "weightedMAE" cost function is more robust than the same model trained with negative log-likelihood loss. Moreover, BNN models are more robust to adversarial attacks than MLP models.

To conclude, multilayer perceptron models have better performances both in nominal and in perturbed conditions than Bayesian neural networks. However they are more sensitive to noise perturbations and adversarial attacks. Instead, Bayesian neural networks provide an estimation of the uncertainty in the prediction, they are robust to noise and adversarial attacks, but with some loss in the nominal performances.

9.2 Future works

In this work, we analyzed some techniques to improve the robustness for regression problems. In particular, we applied these methods to two different kinds of neural networks: multilayer perceptrons and Bayesian neural networks. The comparison of their performances in nominal and perturbed conditions allows us to improve the knowledge about the advantages and weaknesses of these networks, in order to choose the most robust and appropriate solution for the case study. However, machine learning is a deep and dynamic field, therefore many challenges are still open.

One interesting topic concerns the evaluation of the robustness of BNNs by analyzing how the output distributions, related to close target values, are different from each other. Indeed, a robust BNN should predict similar distributions for close target values.

Another interesting development concerns training the BNN models using custom loss functions which consider the likelihood between the prediction and the target value, but also some other specific metrics related to the distribution, such as the novel "Distribution in Range" metric defined in Section 7.3.

Bibliography

[1] Ieee standard glossary of software engineering terminology, 1990.

[2] Heaviside step function, 2017.

[3] Apoorva Agrawal. Loss functions and optimization algorithms. *Medium*, 2017.

[4] Murtaza Eren Akbiyik. Data augmentation in training cnns: Injecting noise to images, 2020.

[5] Jason Brownlee. Crash course on multi-layer perceptron neural networks. *Machine Learning Mastery*, 2016.

[6] Florian Dubost, Gerda Bortsova, Hieab Adams, M. Arfan Ikram, Wiro Niessen, Meike Vernooij, and Marleen de Bruijne. Hydranet: Data augmentation for regression neural networks. *arxiv.org*, 2018.

[7] Stefano Ermon and Volodymyr Kuleshov. Variational inference and mcmc, 2020.

[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples.

[9] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks – a tutorial for deep learning users.

[10] Michel Kana. Uncertainty in deep learning. how to measure? *towards data science*, 2020.

[11] Renu Khandelwal. Overview of different optimizers for neural networks. *Medium*, 2019.

[12] Sungil Kim and Heeyoung Kim. A new metric of absolute percentage error for intermittent demand forecasts. *International Journal of Forecasting*, 32:669–679, 07 2016.

[13] S Lakshmanan. How, when and why should you normalize/standardize/rescale your data. *Towards AI*, 2017.

[14] Assaad Moawad. Neural networks and back-propagation explained in a simple way. *Medium*, 2018.

[15] Vikram Mullachery, Aniruddh Khera, and Amir Husain. Bayesian neural networks. *arxiv.org*, 2018.

[16] Andre T. Nguyen and Edward Raff. Adversarial attacks, regression, and numerical stability regularization.

[17] Maria Elena Nor, Hisyam Lee, Robiah Adnan, and Tun Hussein Onn. Neural network forecasting: Error magnitude and directional change error evaluation. 2015.

[18] Christian Pascual. Tutorial: Understanding regression error metrics in python. *Dataquest*, 2018.

[19] Sountsov Pavel, Suter Chris, Burnim Jacob, V. Dillon Joshua, and the TensorFlow Probability team. Regression with probabilistic layers in tensorflow probability, 2019.

[20] Joseph Rocca. Bayesian inference problem, mcmc and variational inference. *towards data science*, 2019.

[21] Ali Shahrokni and Robert Feldt. A systematic review of software robustness. *Inf. Softw. Technol.*, 55(1):1–17, January 2013.

[22] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *arxiv.org*, 2017.

[23] Connor Shorten and T. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[24] Tensorflow.org. Adversarial example using fgsm.

[25] Tensorflow.org. Tensorflow probability, 2020.

[26] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning.

[27] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *arxiv.org*, 2019.

[28] Alice Zheng. *Evaluating Machine Learning Models*. O'Reilly Media, Inc, City, 2015.