

TDK-dolgozat

Hunor István Lukács

Anomaly detection in serial production and product development

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE



Author:

Hunor István Lukács

Computer Science, Artificial intelligence, MSc

2. year

Supervisors:

Dr. János Botzheim

Associate Professor

Dr. Tamás Fischl

Industrial Supervisor

Budapest, 2023

Contents

1	Introduction	4
2	Anomaly detection in MEMS-based inertial sensor production	6
2.1	MEMS-based inertial sensor	6
2.1.1	MEMS gyroscope	7
2.1.2	MEMS Accelerometer	8
2.1.3	Calibration	9
2.1.4	MEMS fabrication	10
2.2	Use of MEMS-based inertial sensor	11
2.3	About the dataset	11
2.4	Autoencoder based anomaly detection	13
3	Bacterial Evolutionary Algorithm based Autoencoder Architecture Search	18
3.1	Proposed Method	18
3.1.1	Bacterial Evolutionary Algorithm	18
3.1.2	Encoding Method	19
3.1.3	Evaluation of Individuals	22
3.1.4	Bacterial Mutation	23
3.1.5	Gene Transfer	24
3.2	Simulation Results	25
3.2.1	Model training	25
3.2.2	Results	27
4	Explainability based on Fuzzy rule-based system	31
4.1	Fuzzy rule-based system (FRBS)	31
4.1.1	Mamdani inference and center of sums method	33
4.2	Bacterial Memetic Algorithm	34
4.2.1	BMA applied on FRBS	34

CONTENTS

4.2.2	Canberra Distance	36
4.2.3	Levenberg-Marquardt method	36
4.3	Results	40
4.3.1	Linear regression	40
4.3.2	Classification	42
4.4	FRBS classification applied on (MEMS) industrial dataset	44
5	Conclusion and future work	49
	Bibliography	51

Abstract

Nowadays, as a result of the continuous digitization efforts visible in the industry, the data of semifinished and finished products and production processes are becoming available in increasing quantities. A lot of data can be connected through databases, which offer the possibility of recognizing connections that have been hidden to this day. In the production and development area, specifications are always associated with the data, which help to decide whether the product and the process are within the specified limits. At the same time, quality assurance in industry also requires that deviations within limit values and specifications are investigated (outlier detection). It may happen that a small deviation within the specification can later lead to a deviation with a larger effect to the customer. An initial deviation or fluctuation in the multidimensional dataset can therefore be of great importance in terms of the quality and stability of the product or process. Since we are talking about a large amount of data, for the analysis we can use artificial intelligence, which is the express goal of the thesis. In addition to the state-of-the-art solutions (such as the autoencoder architecture applied in an unsupervised manner), the thesis focuses on the explainability of the connections based on fuzzy rule-based system, which greatly facilitates and shortens the time of decision and intervention in the industrial environment. Furthermore, the thesis presents an autoencoder architecture search based on bacterial evolutionary algorithm, applying an abstraction layer based on second order Bézier curves.

Chapter 1

Introduction

With the spread of digitalization in industry, more and more data is becoming available in manufacturing, product development and services. Analyzing and evaluating this data is also a big task, and there are numerous methods for solving it. Today, the use of artificial intelligence to analyze industrial data is becoming important in the field of understanding the correlation in the data.

Numerous AI-based model architectures are available for identifying and inferring relationships in data. In many cases, however, the modelling process is iterative, in which finding the right architecture (providing the right accuracy for a given task) is the result of multiple iterations.

This article analyses data from the production of a MEMS (Micro-electromechanical Systems) based inertial sensor used in the automotive industry. Within that, it deals with the topic of anomaly detection, which consists of searching for small amounts (ppm) of anomalies in the data using an autoencoder neural network architecture. At the same time, the article focuses on the explainability as well, where based on fuzzy rules, the anomalies can be described by defining hyper-cuboids.

The nature of the effects in the data set (and in data sets in general) can also be diverse. They can be the result of random measurement error (not a real error), or the result of systematic behavior either from the product or the environment, or a combination of both. Overall, however, the identification of these deviations, like an early activation, is of paramount importance for the manufacture of the product. If it is considered that an initial deviation that is within the measurement limits, due to long time drift, may lead to an out-of-specification deviation later in the product life cycle, then the detection and identification of these deviations should be part of the quality assurance process.

Although in the present case the autoencoder architecture is given to solve the problem, it is nevertheless not possible to determine in advance the depth of the architecture that is best suited for the task, i.e. how many hidden layers will lead to the best solution. Just as in this case, it is not known which data in the dataset can be called an anomaly.

Thus, this paper/article aims at the bacterial optimization of this chosen autoencoder architecture on the chosen MEMS based inertial production data set, and explaining why a data point is labeled as anomaly or inlier.

Chapter 2

Anomaly detection in MEMS-based inertial sensor production

2.1 MEMS-based inertial sensor

Micro-electromechanical systems (MEMS) [1, 2] are silicon based electro-mechanical systems, approximately 1 to 100 microns (10^{-6}) integrated devices. The MEMS systems combine electrical and mechanical components into small structures in the micrometer scale. The MEMS system consist of the following major components: sensing mechanism, mechanical elements and the application-specific integrated circuit (ASIC). The ASIC drives the mechanical element electrically and evaluate their signals. It is customized for a particular use rather than intended for general purpose use, and typically it is fabricated using metal-oxide-semiconductor technology. These MEMS systems are produced using microfabrication technologies (like lithographic process, polishing, milling, etching), which allows creating complex mechanical and electrical systems. The MEMS-based inertial sensor consists of an acceleration chip a gyroscope and an ASIC. The accelerometers (acceleration chip) measure linear acceleration in one or more axis, while the gyroscopes measure the angular motion. MEMS-based inertial sensors are used in automotive industry, and as consumer sensor, for example in mobile phones. In order to have accurate sensors, a lot of signal must be tested under different conditions. MEMS-based gyroscopes must meet countless requirements like vibration, temperature, humidity, etc., some of which are provided by design, while others must be adjusted through calibration and the key parameters of the product must be provided to the customer. These parameters are e.g. the offset, sensitivity, noise parameters and their temperature dependence. In

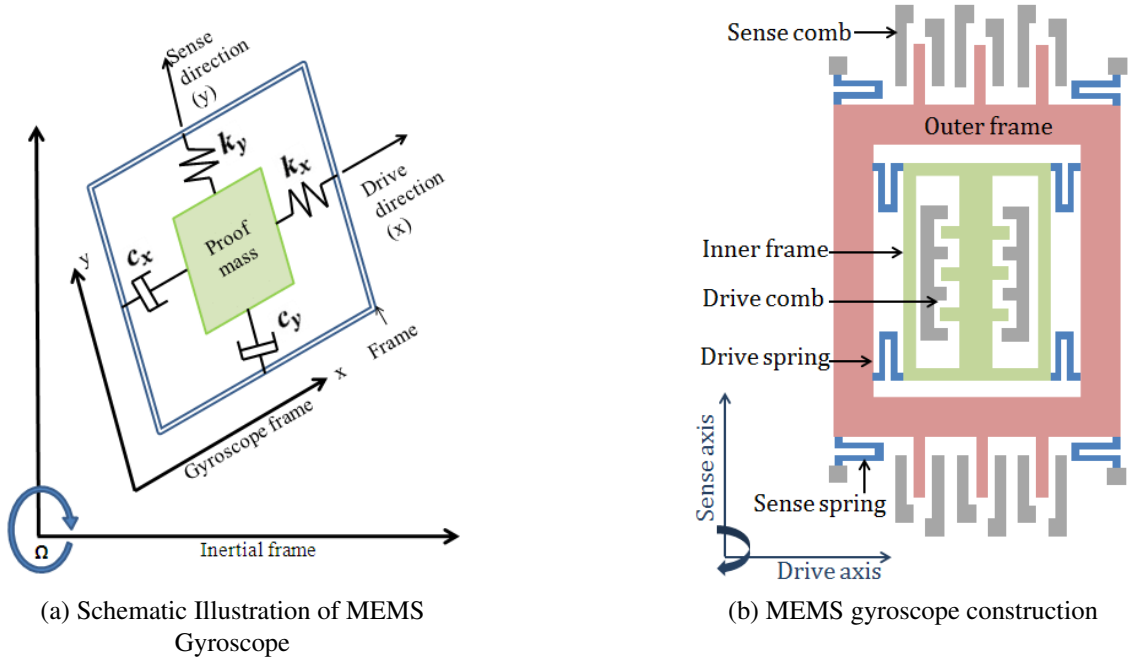


Figure 2.1: MEMS gyroscope [3]

many cases, most parameters are remeasured after subsequent testing, even as a function of temperature, confirming the correctness and stability of the set parameters. Due to the long supply chain process, the numerous production steps and the spread of digitalization, during the manufacturing process thousands of technological and product specific parameters are generated, such as static, time-sequential, temperature, pressure data and so on.

2.1.1 MEMS gyroscope

A MEMS gyroscope sensor [3] is used in measuring the angular rate of an object by means of Coriolis acceleration. In order to sense angular velocity, the majority of these gyroscopes use vibrating mechanical elements. The schematic illustration of such MEMS vibratory gyroscope is presented in Figure 2.1a. The gyroscope structure movement is modelled while it rotates, thus the motion is represented based on a stationary frame (gyroscope frame) and a non-stationary frame (inertial frame). At the core, it has a vibrating (proof) mass, suspended by flexible beams. The device is vibrated at natural frequency, which is known as drive mode. When the gyroscope experiences an angular rotation, a Coriolis force is induced, which causes an energy transfer between the drive mode and sense mode. Thus, the sense mode [4] is the appearing new vibration when the structure begins to rotate and due to the Coriolis force acting on the proof mass, changing the di-

rection of its natural vibration. The movement of the proof mass in the sense direction, caused by the Coriolis force, is proportional to the applied angular rotation. By the use of integrated comb electrodes, based on differential capacitor techniques, this angular rotation can be measured. The Figure 2.1b illustrates schematically an example of such construction of a MEMS gyroscope. Movable comb fingers are connected with the proof mass, the structure is fixed at the anchors, while the rest of the structure is free to move.

The gyroscope can be viewed as two degrees-of freedom (2DoF) system, mass-spring damper system, where the two degrees-of freedom being the two orthogonal directions, the drive direction and the sense direction. The motion equations of the system with respect to the gyroscope frame can be written as follows:

$$m \cdot a_x + c_x \cdot v_x + k_x \cdot x = F_d \quad (2.1)$$

$$m \cdot a_y + c_y \cdot v_y + k_y \cdot y = 0, \quad (2.2)$$

where the m is the mass of the vibrating structure, c is the damping coefficient, k is the beam stiffness, a is the acceleration component, v is the velocity component, and F_d is the electrostatic force by drive combs. The subscripts x and y denote the drive and sense directions correspondingly. Viewed from the inertial frame, the equations can be written as:

$$m \cdot \ddot{x} + c_x \cdot \dot{x} + k_x \cdot x = F_d + 2m\Omega\dot{y} \quad (2.3)$$

$$m \cdot \ddot{y} + c_y \cdot \dot{y} + k_y \cdot y = -2m\Omega\dot{x} \quad (2.4)$$

The induced Coriolis terms are $2m\Omega\dot{y}$ and $2m\Omega\dot{x}$. During an angular rotation The Coriolis term cause dynamic coupling between the drive and sense mode of vibration, thus the suspended proof mass vibrates in the sense direction.

2.1.2 MEMS Accelerometer

The MEMS accelerometer sensors [5] measure the displacement of a mass with an interface circuit. Then the measurement is converted into a digital electrical signal for digital processing. The basic operation of an accelerometer is based on the Newton's second law of motion, which says that the acceleration of a body is directly proportional to, and in the same direction as, the net force acting on the body, and inversely proportional to its mass.

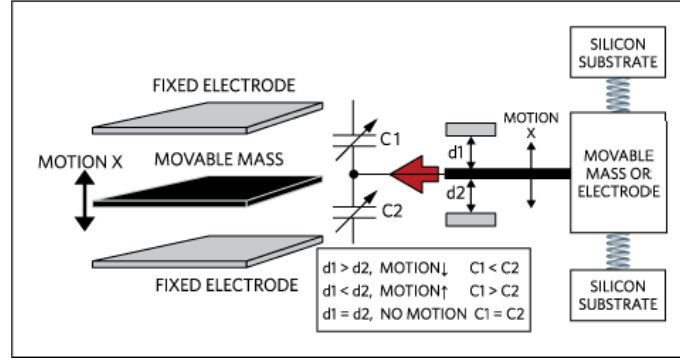


Figure 2.2: Acceleration with a single moving mass [5]

The accelerometer is an electromechanical device, that is machined microfabrication technology, and measures acceleration forces by detecting displacement of the mass relative to fixed electrodes.

In accelerometers, a commonly used sensing approach is capacitor sensing, in which acceleration is based on the change of the capacitor of a moving mass (Figure 2.2). This technique is widespread due to its stability, accuracy and simple structure. The capacitor can be arranged as single-sided or a differential pair. The Figure 2.2 shows an accelerometer with capacitor arranged as a differential pair. It consists of a movable mass placed between two, fixed reference silicon substrates or electrodes. The movement of the mass (Motion x) is relative to the fixed electrodes ($d1$, $d2$), therefore causes change in capacitors ($C1$, $C2$). The displacement of our mass and its direction can be derived by calculating the difference between the two capacitors.

Accelerometers measure linear acceleration along one or more axis. A multiaxis accelerometer can be achieved by mounting an accelerometer differently. (like rotated 90 degree), thus creating a 2-axis accelerometer, which is able to detect horizontal and vertical motions as well. If a rotation is imposed to the accelerometer, the distances $d1$ and $d2$ will not change, thus the accelerometer's output will not respond to a change in angular velocity.

2.1.3 Calibration

With the rise of technologies like navigation and positioning, the demand of inertial sensor accuracy is gradually increasing. The measurement results of MEMS inertial sensors will directly affect the output, thus precision measurement has become an inevitable issue in the development of these inertial sensors. Error may appear in the measurements of the MEMS-based inertial sensor for instance due to short and long-term drifts of the

output signals, misalignment and non-orthogonality of the sensitivity axes, ageing of the silicon structure or some uncertain noise source or random errors and so on. In the calibration process [6] the output of the instrument is compared with known reference information and based on this comparison, coefficients can be determined such that the output agree with the reference information. Non-autonomous calibration needs to be assisted by high-precision instruments, like turntable, centrifuge, shaking table, etc., while autonomous calibration uses external reference excitation provided by gravity field, uniform magnetic field, the rotational angular velocity of the earth and so on.

2.1.4 MEMS fabrication

Numerous technology steps can lead to a proper working MEMS structure. One of the procedures, used in the MEMS fabrication [7] is the silicon-on-glass (SOG) process. The SOG process requires many steps, including anodic bonding, patterning, chemical mechanical polishing, and deep reactive ion etching. SOG based fabricated microstructures are defined by through-etching of a silicon substrate anodically bonded to a glass substrate.

Using chemical mechanical polishing techniques, the desired thickness and surface quality can be achieved for MEMS devices. This polishing process of a silicon wafer consists of lapping and polishing. Firstly, the lapping is used in order to thin the silicon wafer quickly, using large particle size. After the lapping, the polishing process is applied, which consists of prepolishing and final polishing. During the prepolishing, the damaged silicon layer and distortion layer are wiped out, while in the final polishing the aim is to remove the damaged layers resulting from the prepolishing, and also to improve the total roughness and thickness variation of silicon.

Deep reactive ion etching technique is used to define movable structures in inductively coupled plasma etcher. This process is optimized to avoid notching and lag effects on the fabricated silicon structure. The lag effect causes etch uniformity and difficulty in controlling the etch endpoint, thus degrades the mechanical performance, and it may lead to device failure.

2.2 Use of MEMS-based inertial sensor

The MEMS based inertial sensors, by measuring the acceleration and angular velocity of an object (vehicle) along three mutually perpendicular axes, can help in the automotive industry for various tasks. One of the tasks where the control is based on sensor information is active front steering, which describes the steering system of a vehicle where the relationship between the steered road wheels and the driver's steer input may be altered. Furthermore, it is used for hill hold control, which is a driver-assistance system that helps for a stationary vehicle to start uphill by holding the break until the clutch is at the friction point. Roll stability, which monitors the vehicle and identifies dangerous driving conditions that could lead to rolling over. These MEMS sensors are built into Electronic Stability Program in vehicles (ESPi), to improve the vehicle's stability by detecting and reducing loss of traction.

2.3 About the dataset

Our dataset is originating from the MEMS-based inertial sensor production. Data are measurement data of the product from the production process from component up to the final measurement. The data is gathered from a so-called Hadoop database, and the sampled database contains data from 430k sensors. After the data preprocessing and data cleaning, the dataset contains 61 parameters. For the examination of the data, different correlation analysis were applied. The data contained a so-called benchmark problem, which contained a two-dimensional effect that can be found with a suitably sensitive anomaly detector. In addition, the data contained other unknown anomalies, the nature and cause of which are unknown. The MEMS technology and the data compiled in this way have a relatively large standard deviation due to the technology, moreover, the data originated and connected from several factories also contain other interesting and unknown deviations, which proved to be good sample data for the development of the anomaly detector. The aim of the thesis is to create a sufficiently sensitive anomaly detector for known and unknown anomalies, and to investigate how the origin of these anomalies can be explained, transparently represented and explained to MEMS experts.

In this article, normalized data is used in order to prevent sensitive data leakage. The Figure 2.3 depicts the projected data onto 2-dimensional plains, indicating where the data deviations might occur, and also shows the correlation between the selected features. The

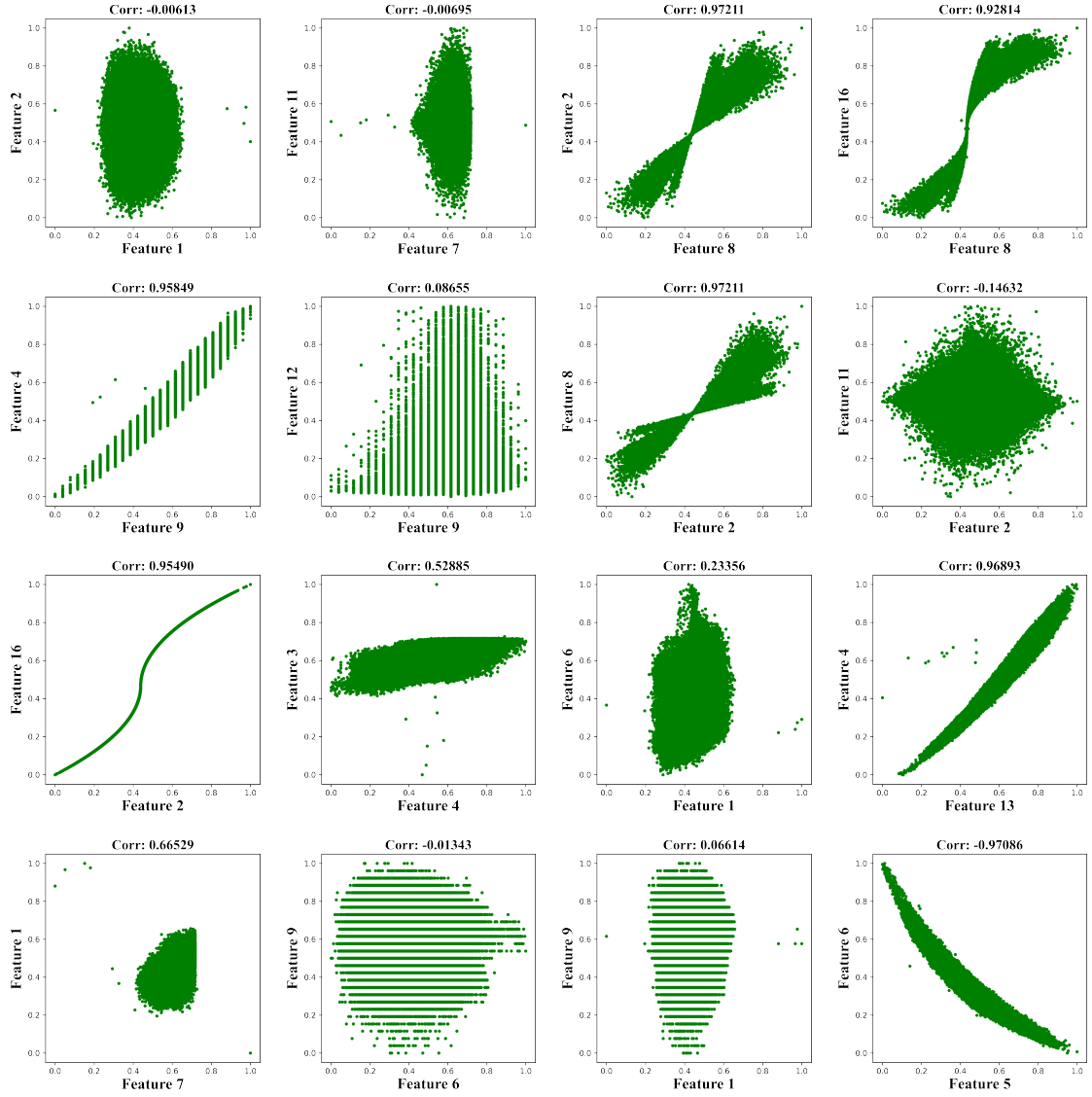


Figure 2.3: Feature correlations and projections



Figure 2.4: Correlation matrix

Figure 2.4 presents the correlation matrix for the selected 16 features, showing the absolute values (magnitude) of the correlations.

2.4 Autoencoder based anomaly detection

Given a dataset and based on expert knowledge a few data points are labeled as anomalies, these are the *ground truth anomalies*. We would like to find an optimal autoencoder architecture, which can split the dataset into inliers and outliers (anomalies) such that all the ground truth anomalies are labeled as anomalies and also to find all the real anomalies.

After the data cleaning and preprocessing procedures, our dataset consists of 430k data point, and 61 features, while the number of ground truth anomalies is 4. The Figure 2.5 illustrates how the 4 ground truth anomalies appear in the correlation of two features. These 4 data points cannot be discovered by the Standard Scaler method, as it only examines the data feature by feature, ignoring correlations between features, it only finds

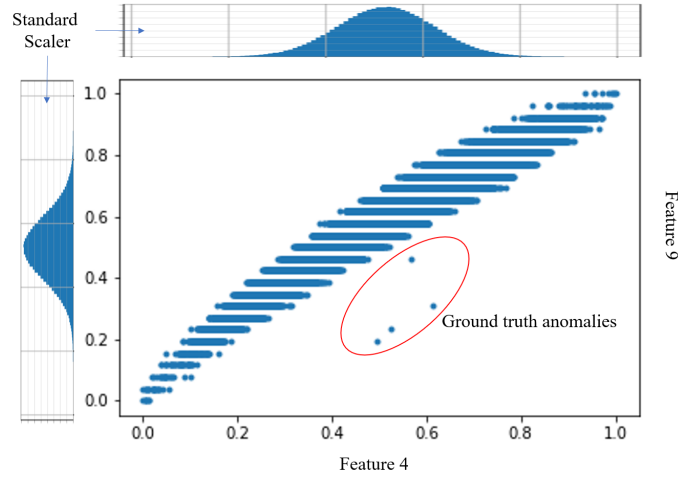


Figure 2.5: Ground truth anomalies

outliers with a standard deviation greater than a given threshold with respect to a single feature. However, this anomaly detection problem [8] can be solved via autoencoders, if the architecture is set appropriately. The Figure 2.5 also shows, that even highly correlated features might contain valuable information regarding the anomalies.

Autoencoder [9] is a type of artificial neural network which is used in unsupervised learning tasks to learn an efficient coding of unlabeled data. The autoencoder model consists of an encoder, a code and a decoder (Figure 2.6). It consists of two main functions, the encoding and the decoding functions. The encoding function transforms the input data into an encoded representation, while the decoding function is used in the recreation of the original input data from the encoded representation. Using these functions, the autoencoder can perform a dimension reduction (typically) in order to learn an efficient representation.

In the case of symmetric autoencoders, the symmetry refers to the architectural symmetry between the encoder and the decoder. The decoder's architecture is the mirrored encoder on the code.

In this section is presented, how an autoencoder with manually set architecture can find these ground truth and real anomalies, while the Chapter 3 presents an automatized solution for the architecture search. In this experiment¹, the autoencoder with manually set architecture has the following layers: 61 nodes in the *input layer*, 57 nodes in the *code*, and 61 nodes in the *output layer*.

The Figure 2.7. shows how would the quantity of anomalies change by choosing dif-

¹System configuration: Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, 3 x NVIDIA Tesla T4 (16 GB) GPU, 256 GB RAM

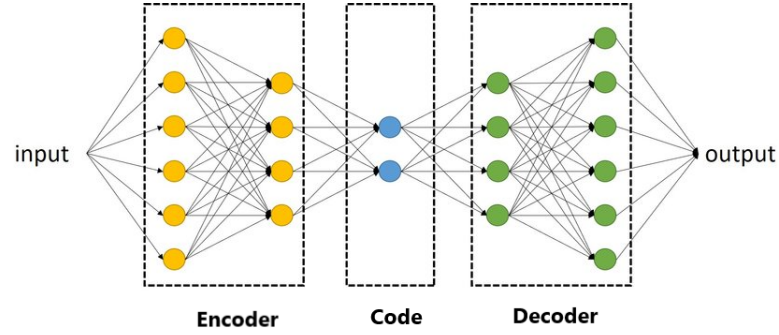


Figure 2.6: Autoencoder architecture [9]

Histogram – anomaly scores – from 4σ to 30σ

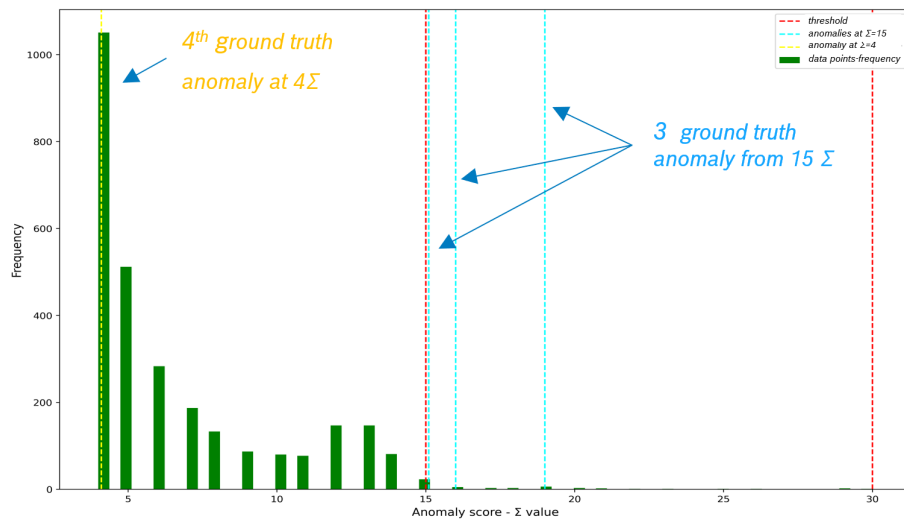


Figure 2.7: Anomalies and the corresponding threshold values

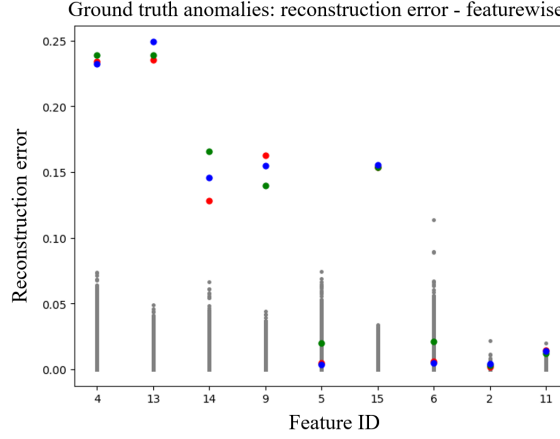


Figure 2.8: Explainability using reconstruction error

ferent Σ values, and also that the 3 of the ground truth anomalies are located outside 15 Σ , while 4th is located at $\Sigma=4$.

The explainability in the case of the autoencoder model is based on the reconstruction error. The Figure 2.8 presents the reconstruction errors of the ground truth anomalies (3 anomalies with $\Sigma > 15$, denoted by red, green and blue) and 10k inliers (randomly selected, denoted by gray) in 9 feature dimensions (for visualization purposes). In the dimensions *Feature*₄, *Feature*₁₃, *Feature*₁₄, *Feature*₉ and *Feature*₁₅ the reconstruction errors are high, while in the *Feature*₅, *Feature*₆, *Feature*₂ and *Feature*₁₁ the reconstruction errors are low for the anomalies. According to Figure 2.5 the ground truth anomalies show deviation in the dimensions of *Feature*₄ and *Feature*₉, furthermore according to the correlation matrix presented in Figure 2.4, the features *Feature*₄, *Feature*₁₃, *Feature*₁₄, *Feature*₉, *Feature*₁₅ are highly correlated, which explains the high reconstruction errors in these feature dimensions.

The Figure 2.9 presents, how on the trained model, a selected threshold value affects the appearing anomalies. Note, that in these figures a 61 dimensional dataset is projected onto a plane, stretched by 2 features. The Figure 2.9a shows the ground truth anomalies (in red frame) and the corresponding sigma values. Since the dataset is projected onto a 2-dimensional space, there are anomalies whose deviation can not be seen in this feature correlation. The Figure 2.9b, Figure 2.9c and Figure 2.9d present those projections, which approves that the data points with high deviation (with red and orange) are indeed anomalies, since they are located far away from the majority. At the same time, the autoencoder can also handle cases where the spread is large (Figure 2.9e, Figure 2.9f).

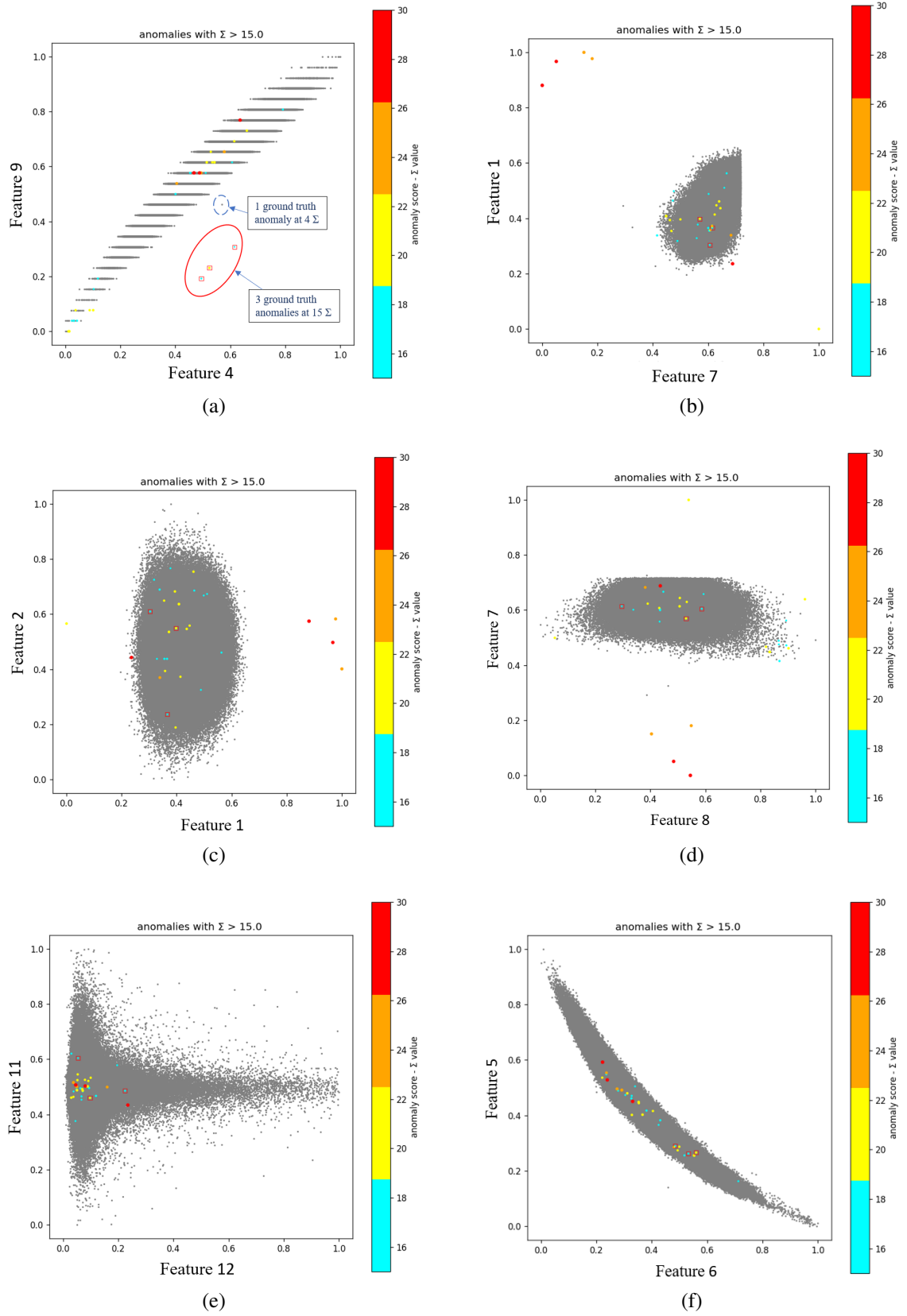


Figure 2.9: Anomaly projections with different Σ values

Chapter 3

Bacterial Evolutionary Algorithm based Autoencoder Architecture Search

This chapter aims to find a solution for such architecture optimization problem based bacterial evolutionary algorithm, thus automatizing the architecture adjustment.

3.1 Proposed Method

3.1.1 Bacterial Evolutionary Algorithm

The bacterial evolutionary algorithm (BEA) [10] is a general optimization method, where each bacterium represents a solution to the original problem. In our case, each bacterium represents a symmetric autoencoder architecture with fully-connected structure. The optimization process consists of three main steps, first is the random generation of the initial population, then the two main operators, the bacterial mutation (Section 3.1.4) and the gene transfer (Section 3.1.5) are performed in each generation as illustrated in Algorithm 1. The maximum number of generations is the stopping criteria, and it is indicated by the N_{gen} hyperparameter. At the end of the algorithm, the solution will be the bacterium with the lowest error value according to the evaluation criteria (Section 3.1.3).

Algorithm 1 Bacterial Evolutionary Algorithm [11]

- 1: initialize population randomly
 - 2: **do**
 - 3: apply bacterial mutation operator (Section 3.1.4)
 - 4: apply gene transfer operator (Section 3.1.5)
 - 5: **while** stopping criteria not reached
-

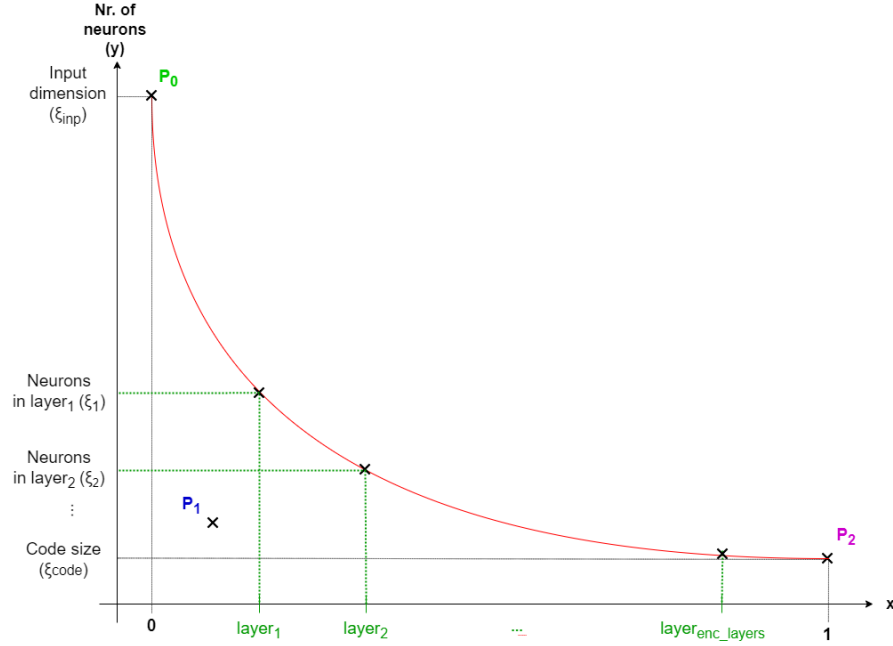


Figure 3.1: Individual representation in the abstract space using Bézier curve

3.1.2 Encoding Method

Each bacterium encodes a symmetric autoencoder in an abstract space, which is based on the quadratic Bézier curves. Bézier curves are used to represent the relationship between the number of neurons in the model's layers. Any desired architecture can be represented by a well-chosen Bézier curve with an appropriate degree of curvature, following the presented method and tailoring the constraints accordingly. In this paper, the quadratic Bézier curve is used.

Due to symmetry (Section 2.6), it's enough to encode the encoder's architecture, whilst the decoder's architecture is simply just the mirrored encoder on the code. The architecture representation in the abstract space based on Bézier curves is presented in Figure 3.1.

Bézier curve

The quadratic Bézier curve is the path traced by the $B(t)$ function, given the control points P_0 , P_1 , and P_2 :

$$B(t) = P_0 \cdot (1-t)^2 + P_1 \cdot 2 \cdot t \cdot (1-t) + P_2 \cdot t^2, 0 \leq t \leq 1 \quad (3.1)$$

Rational Bézier curves adds adjustable weights w_0 , w_1 , and w_2 to the control points P_0 , P_1 , and P_2 respectively:

$$B(t) = \frac{P_0 \cdot w_0 \cdot (1-t)^2 + P_1 \cdot w_1 \cdot 2 \cdot t \cdot (1-t) + P_2 \cdot w_2 \cdot t^2}{w_0 \cdot (1-t)^2 + w_1 \cdot 2 \cdot t \cdot (1-t) + w_2 \cdot t^2}, 0 \leq t \leq 1 \quad (3.2)$$

The modification of the curve, by changing the parameters of the $B(t)$ functions described by the Equation 3.2, will lead to different architectures.

Constraints and DoF

The architecture encoding has the following constraints in the abstract representation:

- 1) P_0 is fixed at $[0, \text{input dimension}]$
- 2) The x coordinate of the P_2 point is fixed at 1
- 3) The x coordinate of $P_1 \in [0, 1]$
- 4) The y coordinate of $P_1 \in [\text{code_size}, \text{input_dimension}]$
- 5) $w_0 = w_2 = 1$, while $w_1 = 5$

The constraint regarding the y coordinate of the P_1 control point is important, since we are encoding a symmetric autoencoder architecture. If this value were greater than the input dimension, the autoencoder could learn the identity. In other, non-autoencoder architectures, this constraint won't be necessary.

The constraint regarding the weight values is important because we want to make sure that moving the P_1 control point will have effect on the curve. If the w_1 value were low, then all the curves generated by three control points would be just a line between P_0 and P_2 .

The degrees of freedom (DoF) is 4 and these are as follows:

- 1) Code size (The y coordinate of the P_2 point)
- 2) N_{enc_layers} : Number of hidden layers in the encoder (same as in the decoder)
- 3) The x coordinate of the P_1 control point
- 4) The y coordinate of the P_1 control point

Since the DoF is four, each bacterial individual represents a symmetric autoencoder architecture with fully-connected structure using only four values.

From abstract representation to architecture.

We take N_{enc_layers} equidistant points in the $[0, 1]$ interval on the horizontal axis, as illustrated in Fig. 3.1 by $layer_1, layer_2, \dots, layer_{enc_layers}$. By projecting these points onto the curve, we get the corresponding N_{enc_layers} intersection points. The rounded y coordinate values of the intersection points show the number of neurons the corresponding layers will contain, these will be denoted by ξ_i .

In order to get the y coordinates of the intersection points, first we have to calculate the t timestamp corresponding to the x coordinate. For this we have to solve the quadratic equation w.r.t t . Let's denote with P_{i_x} and P_{i_y} the x and y coordinates of the P_i control point, respectively. Then we get the a , b , and c coefficients as follows:

$$\begin{aligned} a &= P_{0_x} \cdot w_0 - 2 \cdot P_{1_x} \cdot w_1 + P_{2_x} \cdot w_2 - x \cdot w_0 + x \cdot 2 \cdot w_1 - x \cdot w_2 \\ b &= 2 \cdot P_{0_x} \cdot w_0 + 2 \cdot P_{1_x} \cdot w_1 + x \cdot 2 \cdot w_0 - x \cdot 2 \cdot w_1 \\ c &= P_{0_x} \cdot w_0 - x \cdot w_0. \end{aligned} \quad (3.3)$$

From these coefficients t value can be computed as: $0 = a \cdot t^2 + b \cdot t + c$, where t has to be in the $[0, 1]$ interval.

Then, by substituting the derived t value in Equation 3.2 we derive the intersection point's y coordinates on the Bézier curve corresponding to the t timestamp. The rounded y coordinates will be the ξ_i values. Then the associated autoencoder architecture has the following form:

$$Encoder : [\xi_{inp}, \xi_1] \rightarrow [\xi_1, \xi_2] \rightarrow \dots \rightarrow [\xi_{enc_layers}, \xi_{code}] \quad (3.4)$$

$$Decoder : [\xi_{code}, \xi_{enc_layers}] \rightarrow \dots \rightarrow [\xi_2, \xi_1] \rightarrow [\xi_1, \xi_{inp}] \quad (3.5)$$

where ξ_i is the number of neurons/nodes in the i th layer, ξ_{inp} is the number of nodes in the input layer, ξ_{code} is the number of nodes in the code, while ξ_{enc_layers} is the number of nodes in the N_{enc_layers} -th layer. $[\xi_{in}, \xi_{out}]$ represents a fully connected layer with ξ_{in} number of input nodes and ξ_{out} number of output nodes.

Examples: Individuals, abstract representations, and the associated architectures

The Figure 3.2 presents 2 different bacteria, while Table 3.1 shows the corresponding architectures. The input dimension is 61.

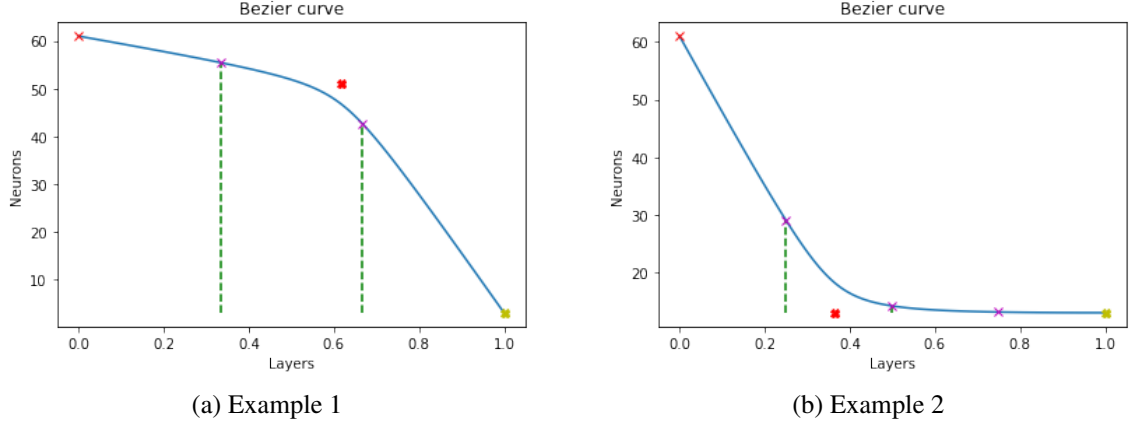


Figure 3.2: Abstract representation, Bézier curves

Id	Bacterium Genes	Autoencoder architecture (fully connected layers)
1	[3; 2; 0.617; 51.207]	[61,55] → [55,43] → [43,3] → [3,43] → [43,55] → [55,61]
2	[13; 3; 0.363; 13]	[61,29] → [29,14] → [14,13] → [13,13] → [13,13] → [13,14] → [14,29] → [29,61]

Table 3.1: Examples

In Table 3.1 each bacterium has the following form:

$[code_size; N_{enc_layers}; P_{1_x}; P_{1_y}]$, referring to the 4 DoF, while in the autoencoder architecture, $[IN, OUT]$ represents a fully connected layer with IN number of input nodes and OUT number of output nodes.

3.1.3 Evaluation of Individuals

By the evaluation of the individual, we assign an error value to it, so it rewards accuracy and penalizes complexity.

The final error value of an individual has the following form:

$$error = loss + \lambda \cdot complexity \quad (3.6)$$

where λ is a hyperparameter of the algorithm, which represents the tradeoff between the accuracy and complexity.

In our case, determining the loss value for an individual consists of two steps. First, we train an autoencoder with the architecture encoded by the individual. At the beginning of the evolutionary algorithm we split our dataset into train set (70%), validation set (15%), and test set (15%). In order to prevent overfitting, early stopping is applied combined with

a learning rate scheduler. In the second step, according to a threshold value on the standard deviation (Σ) of the reconstruction errors, the trained autoencoder splits the whole dataset into two parts, the inliers and the anomalies. The data points whose reconstruction error is less than the threshold value are the inliers, whilst the rest data points are the anomalies. A different threshold value is determined for each architecture, such that all the ground truth anomalies belong to the anomalies, and at the same time, we would like to find all real anomalies (Section 2.4).

Then the loss term is the ratio between the number of data points labeled as anomalies and the number of data points in the whole dataset.

$$loss_{anomaly} = \frac{N_{anomalies}}{N_{all_data_points}} \quad (3.7)$$

The complexity term is the ratio between the trainable parameters of the model (N_{params_model}) and the maximum possible trainable parameters that a model can have (N_{params_max}). In our case, the trainable parameters are the weights and biases in the artificial neural network. The maximum possible trainable parameters an individual can have is determined by the N_{enc_layers} hyperparameter of the algorithm. This hyperparameter shows what the maximum number of layers is in the encoder's architecture between the input layer and the code (which will be mirrored in the decoder). In order to get the number of maximum possible trainable parameters, we assume that each layer contains the maximum amount of neurons, which in our case is the input layer's neuron number.

$$complexity = \frac{N_{params_model}}{N_{params_max}} = \frac{N_{params_model}}{2 \cdot (N_{enc_layers} + 1) \cdot (inp_{dim}^2 + inp_{dim})} \quad (3.8)$$

3.1.4 Bacterial Mutation

The bacterial mutation operator (Figure 3.3, Algorithm 2) is applied for each individual in the population [13]. It creates $N_{clones} + 1$ number of clones for each individual. One group of genes is randomly selected and changed in N_{clones} clones, with one remaining unchanged. The best clone from $N_{clones} + 1$ clones is selected to transfer the mutated genes into the other N_{clones} clones. The same process executes again with the yet-unselected group of genes, and repeats until all the genes in the bacterial chromosome are selected

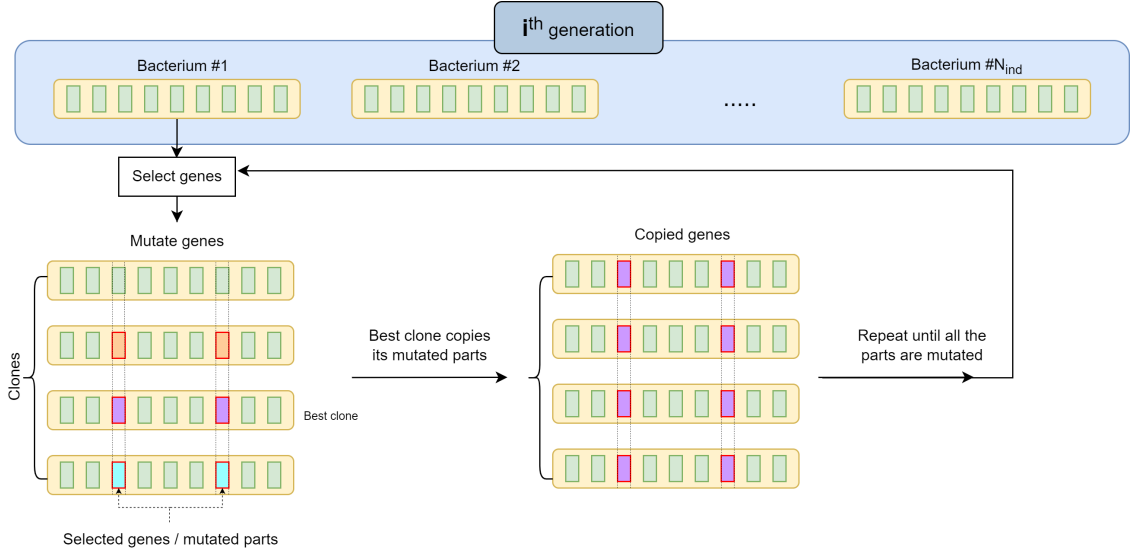


Figure 3.3: Bacterial Mutation [12]

exactly once. The best clone replaces the original individual in the population, discarding the other clones.

Algorithm 2 Bacterial Mutation [11]

Require: *Parameters:* $N_{ind}, N_{clone}, l_{bm}, M_{unit}$

- 1: Create $(N_{clone} + 1)$ clones of *Bacterium*
 - 2: Set $N_{segment} = 4 \cdot N_{rule} \cdot (N_{input} + 1) / l_{bm}$
 - 3: **for** $i = 1$ to $N_{segment}$ **do**
 - 4: Select l_{bm} yet unselected random gene
 - 5: **for** $j = 1$ to N_{clone} **do**
 - 6: Mutate selected gene in *Clone_j*
 - 7: **end for**
 - 8: **for** $j = 1$ to $N_{clone} + 1$ **do**
 - 9: Evaluate *Clone_j* using Eq. 3.6
 - 10: **end for**
 - 11: Select *BestClone*
 - 12: **end for**
 - 13: Set *Bacterium* = *BestClone*
-

3.1.5 Gene Transfer

The gene transfer operator (Figure 3.4, Algorithm 3) works on the population level. The bacterial population is divided into two halves according to error definition, where the individuals with a low error value will belong to the Superior Half, while the individuals with higher error value, to the Inferior Half. A randomly chosen individual from the Superior Half of the population (*Donor*) transfers genes to another randomly chosen bac-

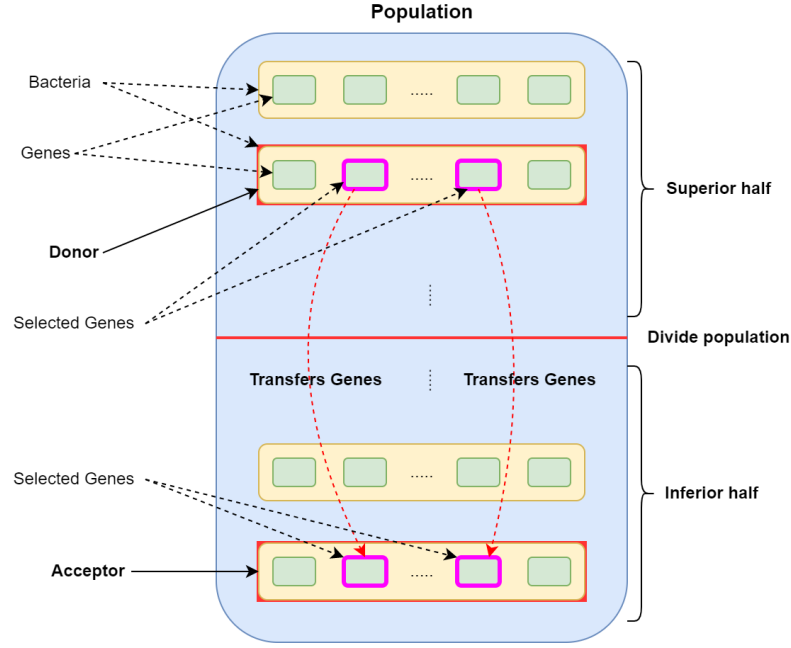


Figure 3.4: Gene Transfer

terium from the Inferior Half (*Acceptor*). The number of genes transferred is l_{gt} , which is a parameter of the algorithm. After transferring the genes, the population is sorted again, and the procedure is repeated N_{inf} times.

Algorithm 3 Gene Transfer [11]

Require: *Population*: N_{ind} Bacterium

Require: *Parameters* : $N_{ind}, N_{inf}, l_{gt}, I_{unit}$

- 1: **for** $i = 1$ to N_{inf} **do**
 - 2: Ascending order of the population acc. to Eq. 3.6
 - 3: $SrcBact = Random(0, \dots, N_{ind}/2 - 1)$
 - 4: $DestBact = Random(N_{ind}/2, \dots, N_{ind})$
 - 5: Select random l_{gt} genes
 - 6: Transfer the selected genes from $SrcBact$ to $DestBact$
 - 7: **end for**
-

3.2 Simulation Results

3.2.1 Model training

In order to evaluate a bacterial individual, first we have to train the encoded autoencoder model. The model training procedure between bacteria are independent of each other. The goal of this training process is to train the autoencoder such that the reconstruction error on the validation set to be minimal. Different architectures yield different

minimum validation errors. In the training procedure in our experiments we used the techniques below.

Mini-batch training

For training the model, we used mini-batch training [14]. The mini-batch is a group of samples, fixed size subset of the original dataset, and in each iteration we train the model on a different subset until all the samples of the original dataset are used. The hyperparameter, which shows the size of a batch is the N_{batch_size} .

Early stopping

In order to avoid overfitting, we used early stopping [15], which is an optimization technique and a form of regularization. The early stopping is used for stopping the training process before the model starts to overfit. In order to decide if the model is overfitting in the early stopping technique, the validation loss is tracked. As soon as the validation loss is increasing or at least not decreasing for $N_{patience}$ iterations, the learning procedure stops. The $N_{patience}$ is a hyperparameter of the algorithm. The other stopping criteria is the $N_{max_iteration}$ the maximum iteration number.

Adam optimizer

For the training of the autoencoder, we used the Adam optimizer [16], which is an extended version of stochastic gradient descent. The Adam optimizer uses adaptive learning rates based on the second moments of the gradients. It calculates the exponential moving average of the gradients and the square gradients, thus the Adam algorithm can be interpreted as a combination of the gradient descent with momentum, and the RMS method.

Learning rate schedule

The Adam optimizer uses adaptive learning rate, thus all the parameters will be updated with an individual learning rate. All the individual learning rates use the initial learning rate, which is a parameter of the algorithm, as an upper limit. By decreasing this learning rate during training, we can reduce the loss at the latest steps.

The learning rate is decreasing during training using the following schedule. First, we set the $learning_rates = [lr_1, lr_2, \dots, lr_n]$ list, which is a hyperparameter and contains the learning rates $(lr_1, \dots, lr_n)$ in decreasing order. At the beginning of training, the learning

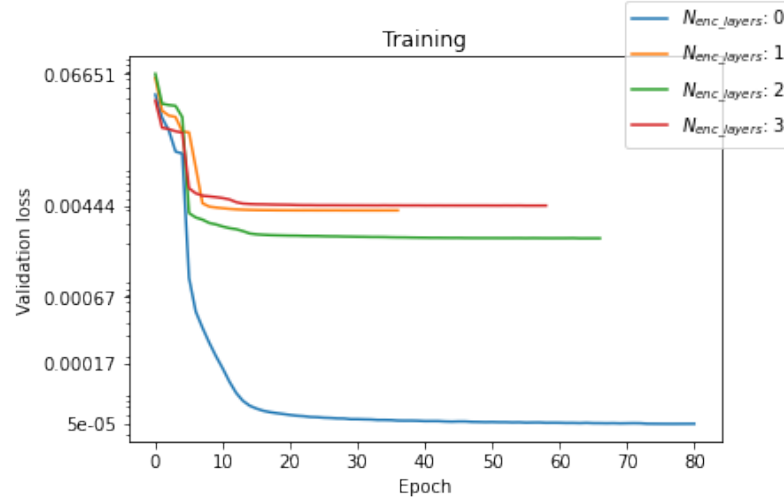


Figure 3.5: Training models with different architectures

rate parameter of Adam is lr_1 . Decreasing the learning rate happens as follows. The optimizer uses the learning rate lr_i and when the early stopper is triggered, we set the learning rate to the next (decreased) value lr_{i+1} and reset the early stopper's counter. If the early stopper is triggered when using the lr_n , then we stop the training.

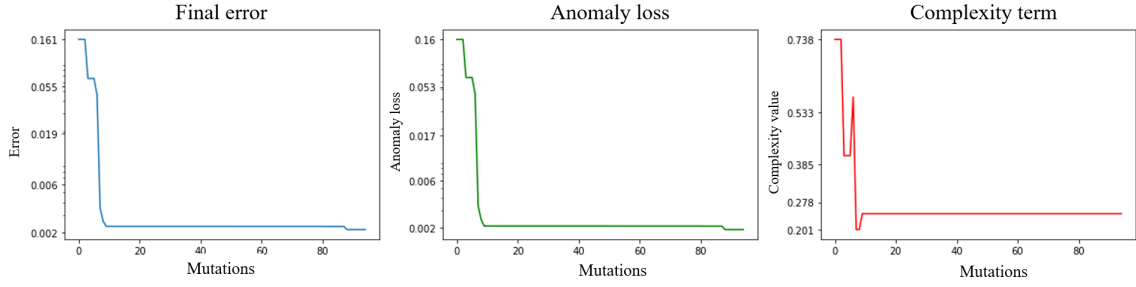
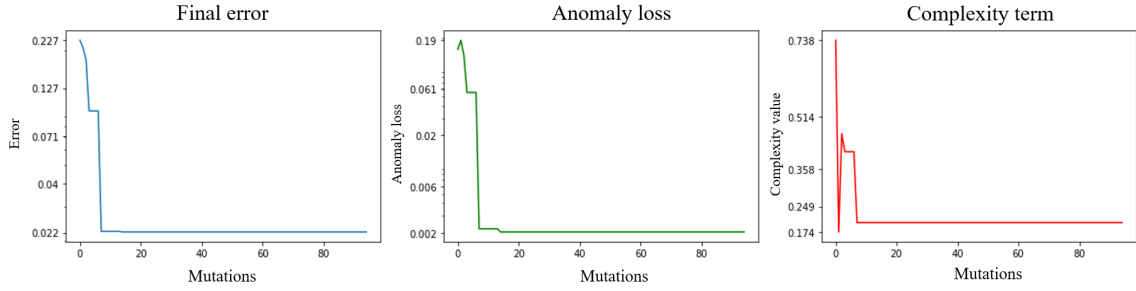
Experiments on model training

In our experiments the following hyperparameters were used: $N_{batch_size} = 256$, $N_{max_iteration} = 100$, $N_{patience} = 3$, $learning_rates = [10^{-3}, 10^{-4}]$. In Fig. 3.5 we can follow on four randomly chosen models with different architectures, how the validation loss decreases over the epochs. The N_{enc_layers} parameter shows how many layers the model has in the encoder (code excluded), and it refers to the N_{enc_layers} parameter (Fig. 3.1). Note, that if the N_{enc_layers} is 0, then the corresponding architecture consists of only an input layer, code, and output layer.

3.2.2 Results

In the first experiment, the λ value is set to 10^{-4} , while in the second experiment it's value is 10^{-1} . All the other hyperparameters are the same in both experiments¹. The used hyperparameters for the bacterial evolutionary algorithm (3.1.1) are the followings: $N_{gen} = 10$, $N_{ind} = 3$, $N_{clone} = 3$, $N_{inf} = 2$, while the maximum number of layers N_{enc_layers} is set to 3. Thus, the number of all gene mutations through the training is 95.

¹System configuration: Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, 3 x NVIDIA Tesla T4 (16 GB) GPU, 256 GB RAM


Figure 3.6: Error through evolution, $\lambda = 10^{-4}$

Figure 3.7: Error through evolution, $\lambda = 10^{-1}$

Evolution

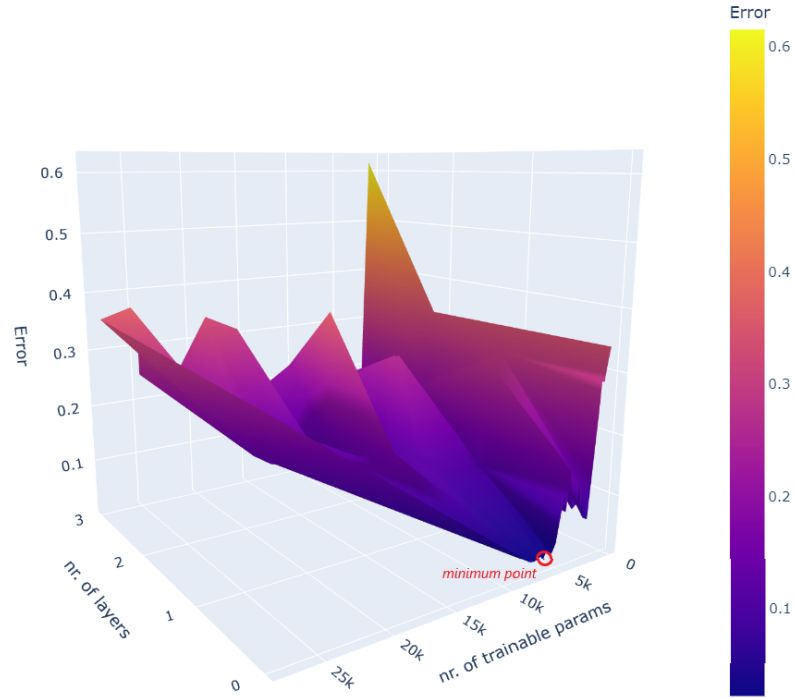
Figures 3.6 and 3.7 depict how the error (Eq. 3.6) is decreasing through all the gene mutations, which includes all the mutations from the bacterial mutation (Section 3.1.4) and gene transfer operators (Section 3.1.5) through the iterations. On the blue curve the evolution of the error value is presented, which has two components, the $loss_{anomaly}$ and the $complexity$, as Eq. 3.6 shows. The green curve shows the corresponding anomaly loss, and the red is the complexity term.

Error meshgrid

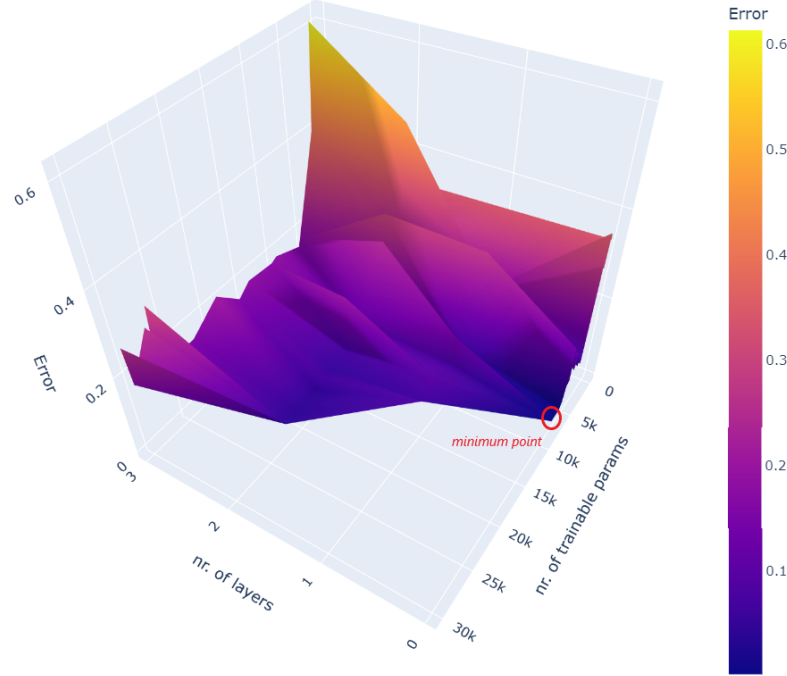
Figures 3.8a and 3.8b present the 3-dimensional meshgrid where we can follow how the error value is affected by the number of layers and the number of trainable parameters in the model. It is observable that the highest peak in error value occurs when the number of layers is the highest and the number of trainable parameters is low. These architectures have the highest anomaly loss. In both cases, the minimum error points are found where the number of layers is minimal and the number of trainable parameters is high.

Solutions

Figure 3.9 presents the two solutions (Figs. 3.9a and 3.9b) for the 2 different λ values, while Table 3.2 presents the corresponding architectures.



(a) Error meshgrid, $\lambda=0.1$



(b) Error meshgrid, $\lambda=0.0001$

Figure 3.8: Error meshgrids

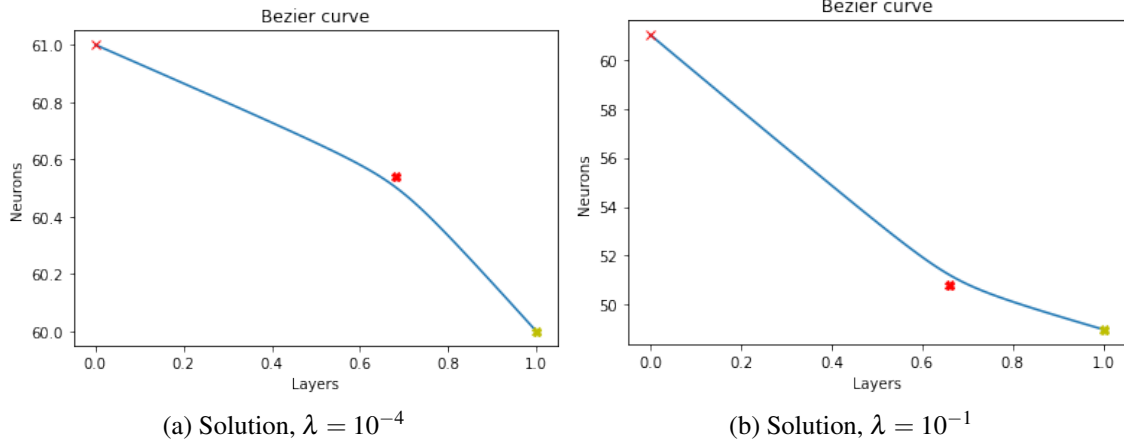


Figure 3.9: Solutions, abstract representations, Bézier curves

Id	Bacterium Genes	Autoencoder architecture (fully connected layers)
a	[60; 0; 0.681; 60.541]	[61, 60] $\rightarrow$ [60, 61]
b	[49; 0; 0.66; 50.834]	[61, 49] $\rightarrow$ [49, 61]

Table 3.2: Solutions

Chapter 4

Explainability based on Fuzzy rule-based system

4.1 Fuzzy rule-based system (FRBS)

The fuzzy logic systems [18] consist of four fundamental mechanisms: fuzzification inference, a knowledge base (also known as a rule base), decision-making (also referred to as inference engine), and defuzzification (Figure 4.1). The rule base is as follows: $R = \{R_i\}$, where R_i are the fuzzy rules [19]. Each rule has the following form:

$$R_i : \text{if } (x_1 \text{ is } A_{i1}) \text{ and } (x_2 \text{ is } A_{i2}) \text{ and } \dots \text{ and } (x_n \text{ is } A_{in}) \text{ then } (y \text{ is } B_i) \quad (4.1)$$

The A_{ij} antecedent membership function belongs to the i th rule and j th input vari-

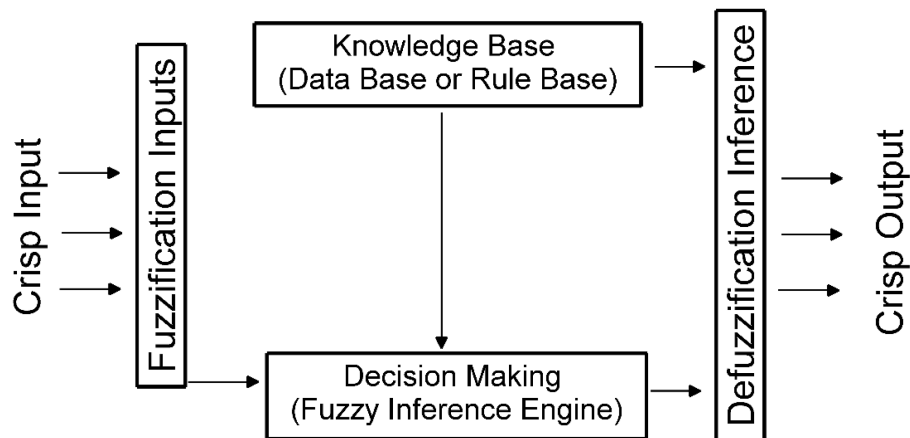


Figure 4.1: Fuzzy logic inference system [17]

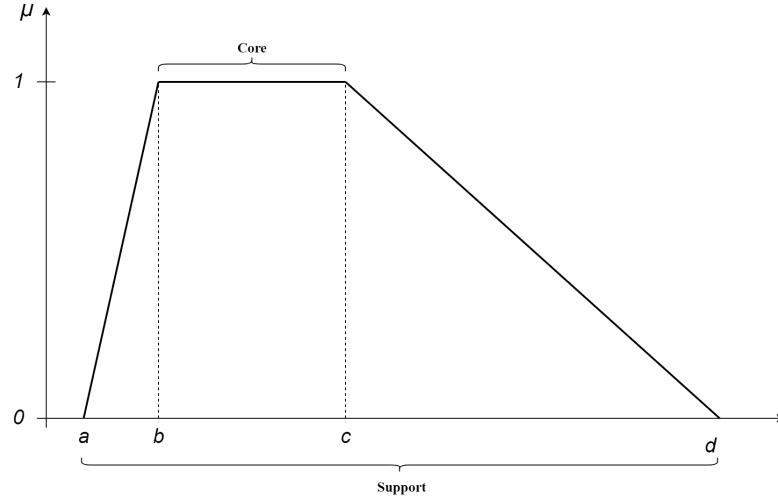


Figure 4.2: Trapezoidal fuzzy set

able and has 4 parameters (breakpoints): a_{ij} , b_{ij} , c_{ij} , d_{ij} (Figure 4.2). The B_i consequent membership function belongs to the i th rule, and it has 4 parameters as well: a_i , b_i , c_i , d_i . Where the crisp input is $X \in \mathbb{R}^n$. The *core* of a fuzzy set (A) is the crisp set that have a membership of one.

$$\text{core}(A) = \{x \in X \mid \mu_A(x) = 1\} \quad (4.2)$$

The *support* of a fuzzy set is the crisp set that have membership greater than zero.

$$\text{supp}(A) = \{x \in X \mid \mu_A(x) > 0\} \quad (4.3)$$

The relative importance of the j th variable in the i th rule is as follows:

$$\mu_{ij}(x_j) = \frac{x_j - a_{ij}}{b_{ij} - a_{ij}} N_{i,j,1}(x_j) + N_{i,j,2}(x_j) + \frac{d_{ij} - x_j}{d_{ij} - c_{ij}} N_{i,j,3}(x_j) \quad (4.4)$$

where $a_{ij} \leq b_{ij} \leq c_{ij} \leq d_{ij}$ must hold, and

$$\begin{aligned} N_{i,j,1}(x_j) &= \begin{cases} 1, & \text{if } x_j \in [a_{ij}, b_{ij}] \\ 0, & \text{if } x_j \notin [a_{ij}, b_{ij}] \end{cases} \\ N_{i,j,2}(x_j) &= \begin{cases} 1, & \text{if } x_j \in (b_{ij}, c_{ij}) \\ 0, & \text{if } x_j \notin (b_{ij}, c_{ij}) \end{cases} \\ N_{i,j,3}(x_j) &= \begin{cases} 1, & \text{if } x_j \in [c_{ij}, d_{ij}] \\ 0, & \text{if } x_j \notin [c_{ij}, d_{ij}] \end{cases} \end{aligned} \quad (4.5)$$

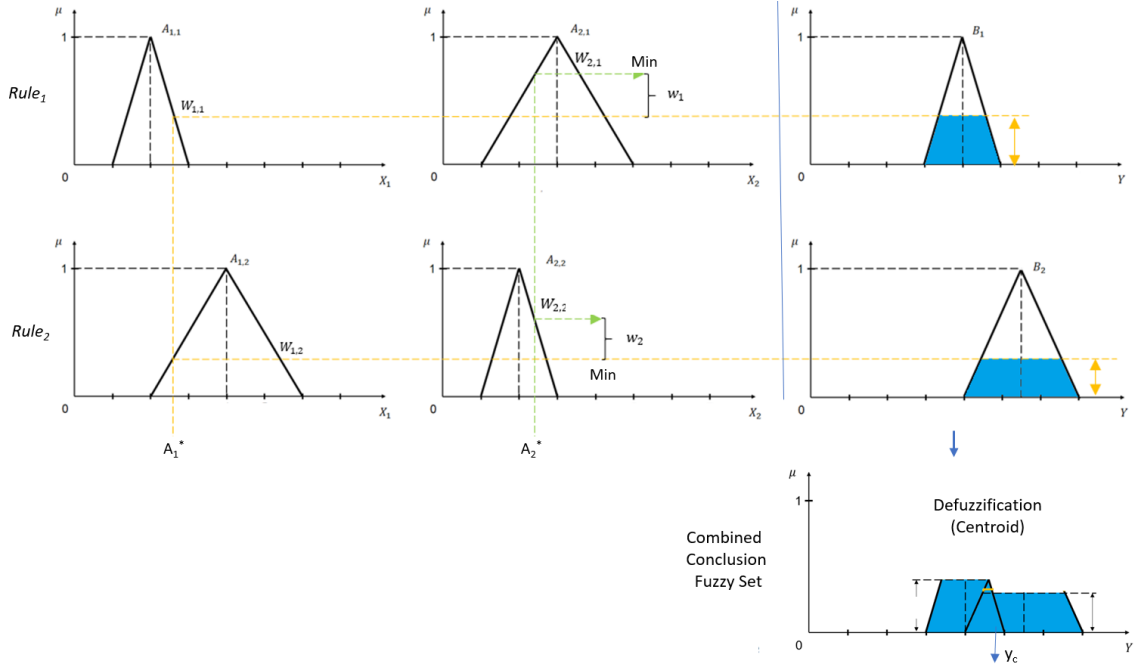


Figure 4.3: Mamdani inference and center of gravity method

The activation degree of the i th rule (the t -norm is the minimum):

$$w_i = \min_{j=1}^n \mu_{ij}(x_j) \quad (4.6)$$

here the n is the number of input dimensions, w_i is the importance of the i th rule if the input vector is $\mathbf{x}$, and $\mu_{i,j}(x_j)$ is the importance of the j th element of the input vector in the i th rule.

4.1.1 Mamdani inference and center of sums method

More fuzzy inference systems exist, but the most popular and most frequently applied is the Mamdani inference system [18]. The Figure 4.3 shows, how a two-rule Mamdani fuzzy inference system derives the output y , having two crisp input values A_1^* and A_2^* . The system uses the minimum for t -norm (Equation 4.6), and maximum for t -conorm.

In order to obtain a crisp output a defuzzification method is applied, which in our case is, the most commonly used, center of sums (COS) method. This method quantifies the gravity center of the combined consequent fuzzy sets (the area of the overlapping region is considered multiple times), and projects onto the y -axis:

$$y_{cos} = \frac{\sum_{i=1}^R y_i \cdot \int \mu_{yi}(y_i) dy}{\sum_{i=1}^R \int \mu_{yi}(y_i) dy} \quad (4.7)$$

where y (or y_{cos}) is the crisp output, R is the number of rules and μ_{yi} is the fuzzy membership value at y_i .

The prediction of the Fuzzy rule-based system, which uses trapezoidal membership functions, Mamdani inference and center of gravity defuzzification method, can be written in the following explicit form (as presented in Article [20]):

$$y(\underline{x}) = \frac{1}{3} \frac{\sum_{i=1}^R 3w_i(d_i^2 - a_i^2)(1-w_i) + 3w_i^2(c_i d_i - a_i b_i) + w_i^3(c_i - d_i + a_i - b_i)(c_i - d_i - a_i + b_i)}{\sum_{i=1}^R 2w_i(d_i - a_i) + w_i^2(c_i + a_i - d_i - b_i)} \quad (4.8)$$

4.2 Bacterial Memetic Algorithm

The Bacterial Memetic Algorithm (BMA) (Fig. 4.4) [10, 11] is a population-based stochastic optimization algorithm, that combines the Bacterial Evolutionary Algorithm (Section 3.1.1) and the Levenberg-Marquardt local search technique (Section 4.2.3).

As discussed in the 3.1.1. section, BEA uses two evolutionary operators, the bacterial mutation and the gene transfer. The bacterial mutation aims to optimize the chromosome, while the gene transfer allows a part of the encoded data to be copied among the bacteria. In this case the used error metric for evaluation of the bacteria and clones, is the Canberra distance (Section 4.2.2).

As a local search technique, the Levenberg-Marquardt method [21] is used, and it is applied with a given probability for each individual due to its computational demand.

Algorithm 4 Bacterial Memetic Algorithm [11]

- 1: initialize population randomly
 - 2: **do**
 - 3: apply bacterial mutation operator (Section 3.1.4)
 - 4: apply Levenberg-Marquardt method (Section 4.2.3)
 - 5: apply gene transfer operator (Section 3.1.5)
 - 6: **while** stopping criteria not reached
-

4.2.1 BMA applied on FRBS

One fuzzy rule-based system corresponds to one bacterial individual in the population. The encoded data are the four breakpoints of the trapezoidal membership functions

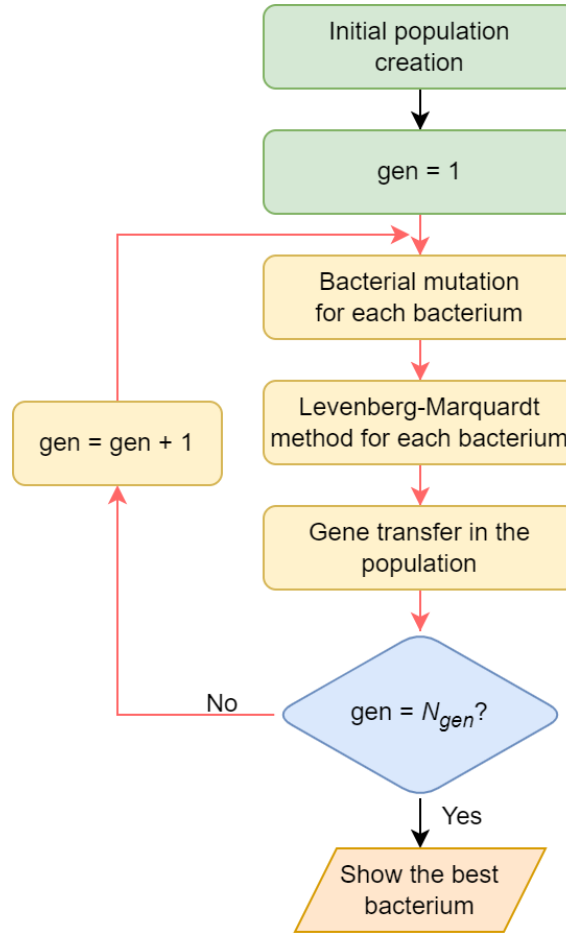
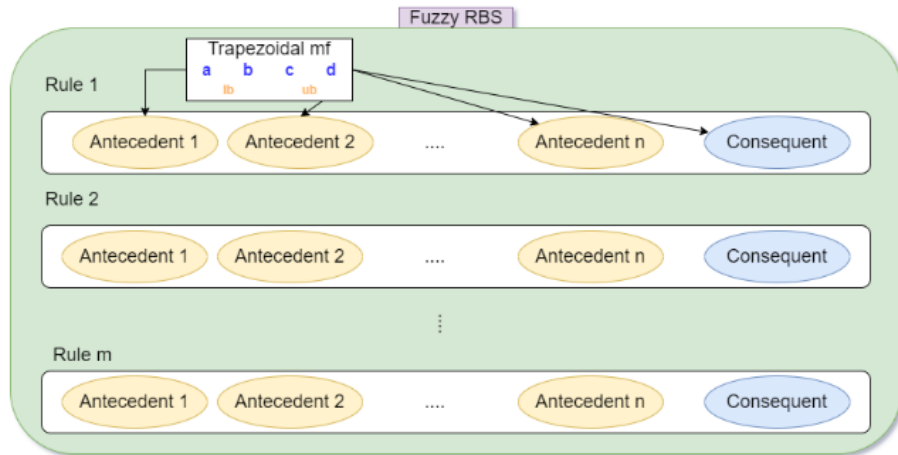


Figure 4.4: Bacterial Memetic Algorithm [20]



(a) Fuzzy rule-based system (FRBS), based on trapezoidal membership functions



(b) Bacterial encoding of FRBS

Figure 4.5: FRBS and the encoded bacterium

for all antecedents and consequents (Figure 4.5). An individual is evaluated according to the Canberra distance (Section 4.2.2). A padding (e.g. 20%) is also applied when searching for the optimal breakpoints of the fuzzy system, meaning that it is allowed to have breakpoints outside the range of interpretation for each feature dimension.

4.2.2 Canberra Distance

The Canberra Distance is the applied error metric, which intrinsically gives greater weight to data points which fall far from their expected value and less weight to data points which fall close to their expected value. For the prediction, we can use the previously presented formula (Formula 4.8). This metric also has the advantage of normalizing the error between 0 and 1. The Canberra distance for the p th pattern:

$$e_p = \frac{|y_p - t_p|}{|y_p| + |t_p|} \quad (4.9)$$

4.2.3 Levenberg-Marquardt method

Algorithm 5 Levenberg-Marquardt Algorithm [11]

Require: *Patterns:* $N_{patterns}$ patterns

Require: *Parameters:* $LM_{prob}, LM_{iter}, \gamma_{init}$

```

1: if  $Random(probability) < LM_{prob}$  then
2:   while  $k < LM_{iter}$  and stopping criteria not reached (Eq. 4.23) do
3:     Calculate  $s_k$  update vector (Eq. 4.19)
4:     Calculate  $r_k$  trust region (Eq. 4.20)
5:     Calculate  $\gamma_{k+1}$  bravery factor (Eq. 4.21)
6:     Evaluate the update vector's ( $s_k$ ) effect (Eq. 4.22)
7:      $k = k + 1$ 
8:   end while
9: end if

```

The Levenberg-Marquardt method (Algorithm 5) is applied for each bacterium by a given probability, indicated by the LM_{prob} parameter [11].

In this step, unlike in the bacterial mutation or gene transfer, the sum of squares error (SSE) is used as the criterion. For the prediction, we use the formula presented before (4.8).

$$SSE = \frac{\|\vec{y} - \vec{t}\|^2}{2} = \frac{\|\vec{e}\|^2}{2} \quad (4.10)$$

The Jacobian matrix of an FRBS

This algorithm is a gradient-based optimization technique, therefore it is vital of calculating the model's output derivatives with respect to the model's parameters (Article [20]). The number of rows of the Jacobian matrix is equal to the number of patterns, while the number of columns is the number of all membership functions' parameters (all break points in the membership functions). The Jacobian matrix [20]:

$$\mathbf{J}_k = \left[\frac{\partial y(\vec{x}^{(p)}, \vec{b}_k)}{\partial \vec{b}_k^T} \right] \quad (4.11)$$

where $\vec{b}_k$ stands for the parameters of the fuzzy model, the encoded FRBS, and k is the step index. By substituting the parameters of the model, the Jacobian matrix can be written as follows:

$$\mathbf{J} = \left[\frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{11}} \frac{\partial y(\mathbf{x}^{(p)})}{\partial b_{11}} \dots \frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{12}} \dots \frac{\partial y(\mathbf{x}^{(p)})}{\partial d_1} \dots \frac{\partial y(\mathbf{x}^{(p)})}{\partial d_R} \right] \quad (4.12)$$

where,

$$\begin{aligned} \frac{\partial y(\mathbf{x}^{(p)})}{\partial a_{ij}} &= \frac{\partial y}{\partial w_i} \frac{\partial w_i}{\partial \mu_{ij}} \frac{\partial \mu_{ij}}{\partial a_{ij}} & \frac{\partial y(\mathbf{x}^{(p)})}{\partial c_{ij}} &= \frac{\partial y}{\partial w_i} \frac{\partial w_i}{\partial \mu_{ij}} \frac{\partial \mu_{ij}}{\partial c_{ij}} \\ \frac{\partial y(\mathbf{x}^{(p)})}{\partial b_{ij}} &= \frac{\partial y}{\partial w_i} \frac{\partial w_i}{\partial \mu_{ij}} \frac{\partial \mu_{ij}}{\partial b_{ij}} & \frac{\partial y(\mathbf{x}^{(p)})}{\partial d_{ij}} &= \frac{\partial y}{\partial w_i} \frac{\partial w_i}{\partial \mu_{ij}} \frac{\partial \mu_{ij}}{\partial d_{ij}} \end{aligned} \quad (4.13)$$

From equation 4.6 we can see, that w_i depends on the membership functions, and each membership function depends on 4 breakpoints. The derivatives of w_i are:

$$\frac{\partial w_i}{\partial \mu_{ij}} = \begin{cases} 1, & \text{if } \mu_{ij} = \min_{k=1}^n \mu_{ik} \\ 0, & \text{otherwise} \end{cases} \quad (4.14)$$

and the derivatives of the membership functions:

$$\begin{aligned} \frac{\partial \mu_{ij}}{\partial a_{ij}} &= \frac{x_j^{(p)} - b_{ij}}{(b_{ij} - a_{ij})^2} N_{i,j,1}(x_j^{(p)}) & \frac{\partial \mu_{ij}}{\partial c_{ij}} &= \frac{d_{ij} - x_j^{(p)}}{(d_{ij} - c_{ij})^2} N_{i,j,3}(x_j^{(p)}) \\ \frac{\partial \mu_{ij}}{\partial b_{ij}} &= \frac{a_{ij} - x_j^{(p)}}{(b_{ij} - a_{ij})^2} N_{i,j,1}(x_j^{(p)}) & \frac{\partial \mu_{ij}}{\partial d_{ij}} &= \frac{x_j^{(p)} - c_{ij}}{(d_{ij} - c_{ij})^2} N_{i,j,3}(x_j^{(p)}) \end{aligned} \quad (4.15)$$

Then, we also have to compute $\frac{\partial y}{\partial w_i}$ and the derivatives of the output membership functions parameters. From Equation 4.8, it follows that

$$\begin{aligned} \frac{\partial y}{\partial w_i} &= \frac{1}{3} \frac{\text{den} \frac{\partial F_i}{\partial w_i} - \text{num} \frac{\partial G_i}{\partial w_i}}{(\text{den})^2} & \frac{\partial y}{\partial c_i} &= \frac{1}{3} \frac{\text{den} \frac{\partial F_i}{\partial c_i} - \text{num} \frac{\partial G_i}{\partial c_i}}{(\text{den})^2} \\ \frac{\partial y}{\partial a_i} &= \frac{1}{3} \frac{\text{den} \frac{\partial F_i}{\partial a_i} - \text{num} \frac{\partial G_i}{\partial a_i}}{(\text{den})^2} & \frac{\partial y}{\partial d_i} &= \frac{1}{3} \frac{\text{den} \frac{\partial F_i}{\partial d_i} - \text{num} \frac{\partial G_i}{\partial d_i}}{(\text{den})^2} \\ \frac{\partial y}{\partial b_i} &= \frac{1}{3} \frac{\text{den} \frac{\partial F_i}{\partial b_i} - \text{num} \frac{\partial G_i}{\partial b_i}}{(\text{den})^2} \end{aligned} \quad (4.16)$$

where *den* and *num* are the denominator and the numerator of Equation 4.8, respectively. F_i and G_i are the i th member of the sum in the numerator and the denominator, respectively. In Equation 4.16 the derivatives will be:

$$\begin{aligned} \frac{\partial F_i}{\partial w_i} &= 3(d_i^2 - a_i^2)(1 - 2w_i) + 6w_i(c_i d_i - a_i b_i) + 3w_i^2 [(c_i - d_i)^2 - (a_i - b_i)^2] \\ \frac{\partial G_i}{\partial w_i} &= 2(d_i - a_i) + 2w_i(c_i + a_i - d_i - b_i) \end{aligned} \quad (4.17)$$

$$\begin{aligned} \frac{\partial F_i}{\partial a_i} &= -6w_i a_i + 6w_i^2 a_i - 3w_i^2 b_i - 2w_i^3 (a_i - b_i) & \frac{\partial G_i}{\partial a_i} &= -2w_i + w_i^2 \\ \frac{\partial F_i}{\partial b_i} &= -3w_i^2 a_i + 2w_i^3 (a_i - b_i) & \frac{\partial G_i}{\partial b_i} &= -w_i^2 \\ \frac{\partial F_i}{\partial c_i} &= 3w_i^2 d_i - 2w_i^3 (d_i - c_i) & \frac{\partial G_i}{\partial c_i} &= w_i^2 \\ \frac{\partial F_i}{\partial d_i} &= 6w_i d_i - 6w_i^2 d_i + 3w_i^2 c_i + 2w_i^3 (d_i - c_i) & \frac{\partial G_i}{\partial d_i} &= 2w_i - w_i^2 \end{aligned} \quad (4.18)$$

Update-vector

At the k -th iteration, the update-vector for a bacterium:

$$\vec{s}_k = -(\mathbf{J}_k^T \mathbf{J}_k + \gamma_k \mathbf{I})^{-1} \mathbf{J}_k^T \vec{e}_k \quad (4.19)$$

where $\mathbf{I}$ is the identity matrix and γ_k is initially set to any positive value ($\gamma_{\text{init}} = \gamma_1 > 0$). The $\vec{e}_k$ is the sum of squares error (SSE) between the prediction and the desired output.

Trust region

The trust region is computed as follows:

$$r_k = \frac{||\vec{b}_k||^2 - ||\vec{b}_k + \vec{s}_k||^2}{||\vec{b}_k||^2 - ||\mathbf{J}_k \vec{s}_k + \vec{e}_k||^2} \quad (4.20)$$

where $\vec{b}_k + \vec{s}_k$ is the parameter vector of the updated fuzzy model.

Bravery factor

The γ parameter (the bravery factor) is dynamically adjusted depending on the trust region value:

$$\gamma_{k+1} = \begin{cases} 4\gamma_k, & \text{if } r_k < 0.25 \\ \gamma_k/2, & \text{if } r_k > 0.75 \\ \gamma_k, & \text{otherwise} \end{cases} \quad (4.21)$$

Evaluation

Then, the new bacterium $\vec{b}_{k+1}$ will be the following:

$$\vec{b}_{k+1} = \begin{cases} \vec{b}_k + \vec{s}_k, & \text{if } ||\vec{b}_k + \vec{s}_k||^2 < ||\vec{b}_k||^2 \\ \vec{b}_k, & \text{otherwise} \end{cases} \quad (4.22)$$

where the $\vec{b}_k$ is the original bacterium, before the LM operation.

Stopping criterion

The algorithm stops when either the stopping criterion is met, or when the maximum iteration number is reached.

$$\begin{aligned} \Omega[k-1] - \Omega[k] &< \theta[k] \\ ||\mathbf{par}[k-1] - \mathbf{par}[k]|| &< \sqrt{\tau_f}(1 + ||\mathbf{par}[k]||) \\ ||g[k]|| &\leq \sqrt[3]{\tau_f}(1 + |\Omega[k]|) \end{aligned} \quad (4.23)$$

where Ω is the SSE (Eq. 4.10), $\theta[k] = \tau_f(1 + \Omega[k])$, $\tau_f = 10^{-2}$, $g[k] = \partial\Omega/\partial\mathbf{par}^T[k]$ ($g[k]$ is the gradient vector), $\mathbf{par}$ is the vector of parameters to be optimized, and k is the iteration variable.

4.3 Results

4.3.1 Linear regression

Before applying a fuzzy model on the MEMS dataset, it is essential to see how these models behave in solving different machine learning tasks. The scope of this experiment is to investigate how the fuzzy rule-based model can be used, optimized with the presented bacterial memetic algorithm, in linear regression problems. In this example, the chosen function is rather simple ($f(x_1, x_2) = 2 \cdot \sin(x_1) + x_2$) which is used to generate the train dataset disturbed with noise. The used hyperparameters in this experiment¹ are: $N_{gen} = 100$, $N_{ind} = 16$, $N_{clone} = 3$, $N_{inf} = 10$, $LM_{prob} = 0.2$, $LM_{iter} = 10$, and $\gamma_{init} = 0.5$. The Figure 4.6 illustrates how the fuzzy rule-base model can learn this arbitrarily chosen function. In blue, we can see the test dataset and with different colors the predictions of the trained model. The test dataset is also generated using the function $f(x_1, x_2) = 2 \cdot \sin(x_1) + x_2$ and disturbed by added noise.

In order to prevent the overfitting, the optimal number of rules can be determined (Algorithm 6) as presented in Article [12]. In the *CreateFRBbyBMA* method, in each run the fuzzy rule base extraction method is applied, with the given rule number. In the *EvalFRB* method, the provided fuzzy rule base system is evaluated using all the training patterns, according to the Canberra distance 4.2.2. The optimal rule number is calculated by checking the drops of the average errors between the fuzzy models with adjacent number of rules. The optimal rule number is achieved in the fuzzy model, where the drop between the current average error value and the next average error value (average error value of the fuzzy model with one more rule) is the highest. The model right before the highest average-error-drop is the best in detecting outliers because it's not-so-accurate, compared to the model with one more rule, which tend to overfit the training dataset. Based on the highest average-error-drop, the model with fewer rules is the optimal candidate since it has a good enough approximation but at the same time it does not overfit the dataset.

¹System configuration: Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, 3 x NVIDIA Tesla T4 (16 GB) GPU, 256 GB RAM

Algorithm 6 Optimal Number of Rules [12]

Require: $minRule$, $maxRule$, $nRuns$ parameters

```

1: for  $rule = minRule$  to  $maxRule$  do
2:   for  $run = 1$  to  $nRuns$  do
3:     FuzzyRuleBase[rule][run] = CreateFRBbyBMA(rule)
4:     ErrorMatrix[rule][run] = EvalFRB(FuzzyRuleBase[rule][run])
5:   end for
6:   AverageErrors[rule] =  $\frac{1}{nRuns} \sum_{i=1}^{nRuns} ErrorMatrix[rule][i]$ 
7: end for
8: ErrorDrop = AverageErrors[minRule] - AverageErrors[minRule+1]
9: maxErrorDrop = ErrorDrop
10: optRule = minRule
11: for  $rule = minRule + 1$  to  $maxRule - 1$  do
12:   ErrorDrop = AverageErrors[rule] - AverageErrors[rule+1]
13:   if ErrorDrop > maxErrorDrop then
14:     maxErrorDrop = ErrorDrop
15:     optRule = rule
16:   end if
17: end for

```

N_{rules}	$N_{rules} = 1$	$N_{rules} = 2$	$N_{rules} = 3$	$N_{rules} = 4$	$N_{rules} = 5$
<i>AvgError</i>	4.2318	3.0208	2.7862	1.4707	1.3195

Table 4.1: Errors and the corresponding number of rules

N_{rules}	$1 \Leftrightarrow 2$	$2 \Leftrightarrow 3$	$3 \Leftrightarrow 4$	$4 \Leftrightarrow 5$
<i>ErrorDrop</i>	1.2109	0.2346	1.3115	0.1511

Table 4.2: Error drops

In this experiment the used hyperparameters are the following: $minRule = 1$, $maxRule = 5$, $nRuns = 2$. The Table 4.1 presents how the average error is changing by using different number of rules in our fuzzy model, while Table 4.2 shows the error drop between models with consecutive number of rules. The higher error drop is between the models with 3 and 4 rules, thus the optimal rule number according to the Algorithm 6 is 3.

The rules of the trained model are shown in the Figure 4.7. Each rule has as many Antecedents as many features the dataset has (it is 2 in our case) and one Consequent. With dashed lines are represented 3 observations (each with different color) from the test dataset, and we can follow, how the decision is made by the model. By labeling intervals, a linguistic interpretation could be associated to these rules. Using the trained model with $N_{rules} = 3$ (Figure 4.7), the associated linguistic interpretations can be, for example:

- if X is *high* and Y is *high* then Z is *high* (acc. to Rule[2])
- if X is *mid* and Y is *high* then Z is *high* (acc. to Rule[2])
- if X is *mid* and Y is *low* then Z is *low* (acc. to Rule[1])
- if X is *mid* and Y is *mid* then Z is *mid* (acc. to Rule[0])
- if X is *mid* and Y is *low* then Z is *mid* (acc. to Rule[0])

4.3.2 Classification

In this experiment we apply the fuzzy rule-based system on a classification problem. As the 4.8. figure shows, we have a linearly non-separable train dataset, represented by the larger spheres, where each data point belongs to either in the yellow or in the blue class. The data points in the yellow class have value 1 on the z axis, while the blue data points have 0. With smaller spheres are represented the predictions on the test dataset.

The Figure 4.9 presents the trained fuzzy rules for the yellow class. The consequent's value (on the y axis) shows the certainty of the model, thus if it's value is 1, than model is certain that the data point (observation) belongs to the yellow class. If the consequent's value is 0, than the observation belongs to the blue class. A consequent value between 0 and 1 can be interpreted as the extent of uncertainty, and the color transition of the predicted values in the Figure 4.8 refers also to this uncertainty.

As matter of explainability, the fuzzy rules help us describe intervals in each feature dimension, defining hyper cuboids, corresponding to each class. According to the Figure 4.9

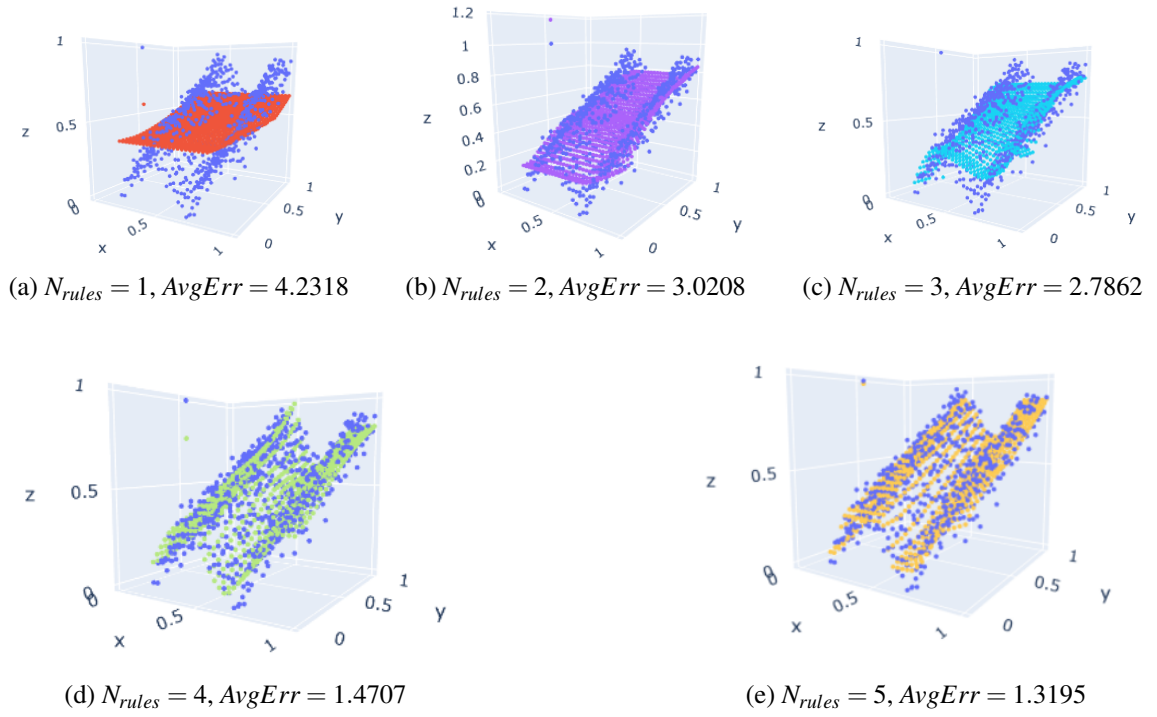


Figure 4.6: Linear regression using different number of rules

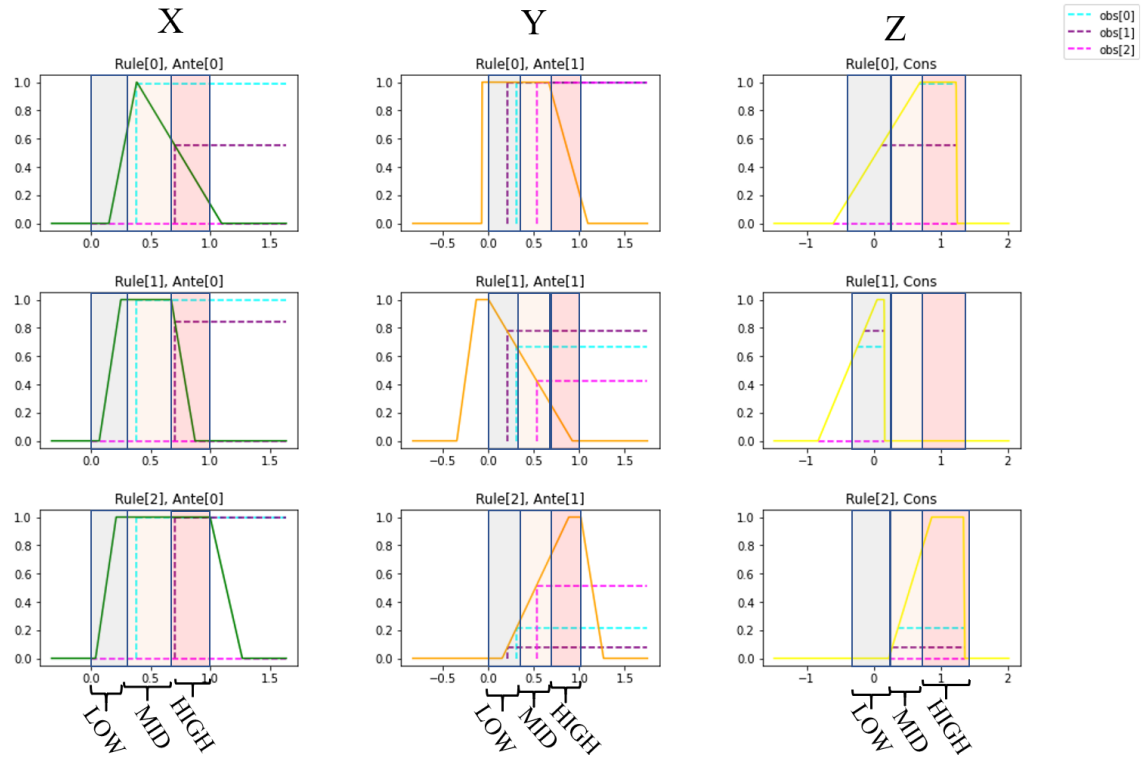


Figure 4.7: Fuzzy rules for linear regression, model with 3 rules

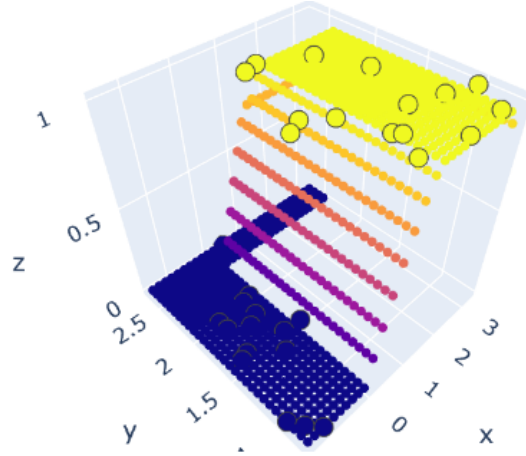


Figure 4.8: Fuzzy rules for classification

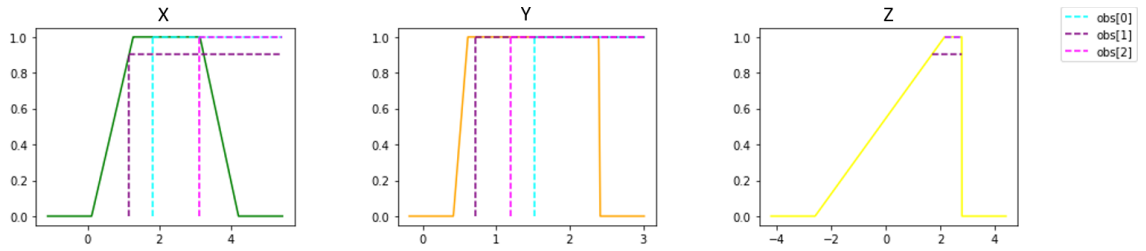


Figure 4.9: Rules for classification

and using the rule Structure 4.1, the following rule can be derived:

$$\text{Rule : if } (x \in [0 \quad 1.5 \quad 3 \quad 4.1]) \text{ and } (y \in [0.45 \quad 0.65 \quad 2.4 \quad 2.41]) \text{ then } (z \text{ is Yellow}) \quad (4.24)$$

4.4 FRBS classification applied on (MEMS) industrial dataset

In this experiment², the used dataset contains measurements of MEMS inertial sensors, and it consists of 1k data points and 6 features. The features selected are the following: *Feature*₄, *Feature*₁₃, *Feature*₁₄, *Feature*₉, *Feature*₅ and *Feature*₁₅. The Features were selected such that the ground truth anomalies can be discovered (Figure 2.5), furthermore the narrowed dataset also contains other real anomalies in menegable quantity that are hidden in the multidimensional correlations. As in the Figure 2.4 the correlation matrix shows, the *Feature*₄, *Feature*₁₃, *Feature*₁₄, *Feature*₉, and *Feature*₁₅ are highly correlated, while the *Feature*₅ does not correlate with any of the other Features. First, the

²System configuration: Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, 3 x NVIDIA Tesla T4 (16 GB GPU, 256 GB RAM

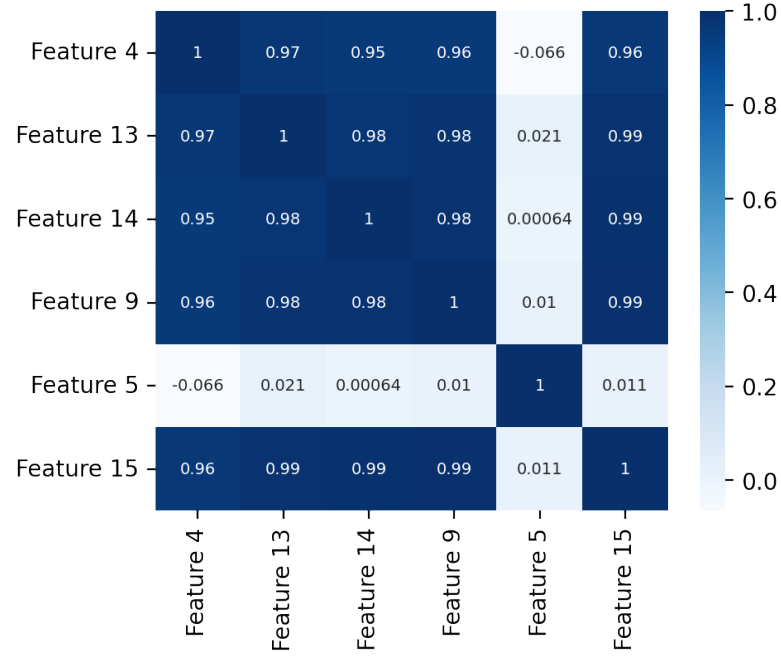


Figure 4.10: Feature correlation

dataset is labeled by an autoencoder following the procedure presented in Section 2.4, splitting the data into inliers and outliers. The threshold Σ value is chosen such that the three ground truth anomalies lie in the outlier zone. The labeled dataset contains 994 inliers and 6 outliers according to the autoencoder. The task in this case is to use a fuzzy model on this classification task and see how the anomalies can be described according to the fuzzy rules.

The Figure 4.11 presents the training curve of the fuzzy model, how the error decreases by generations. The Figure 4.11 reveals that if the number of rules is set to 1, than the derived model is not complex enough, thus it tends to underfit, while using 2 rules the model's complexity is enough to separate the dataset according to the labels provided by the autoencoder.

The Figure 4.12a and Figure 4.12b depict the trained fuzzy rule-based system. The system constitutes of 2 rules, each has 6 antecedents and 1 consequent, while with dashed lines are represented the observations. Different anomalies are represented by different colors, with the ground truth anomalies in red, while the other three real anomalies are purple, magenta and cyan. The observation values for each anomaly are presented in Table 4.4.

The Figure 4.12a illustrates how the decisions are made in the case of the three ground thruth anomalies. The dashed red lines show the ground thruth anomalies as observations

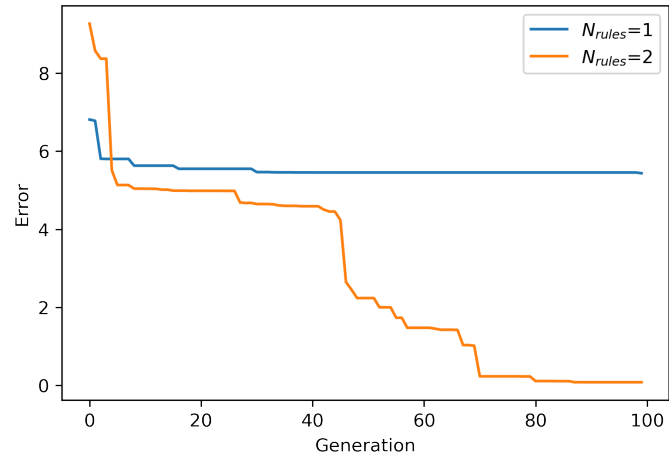
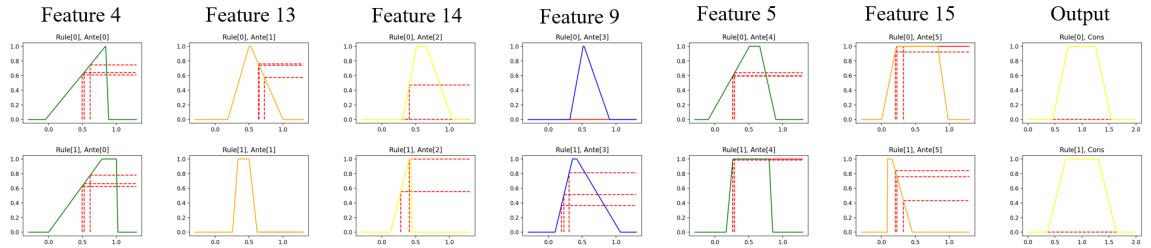
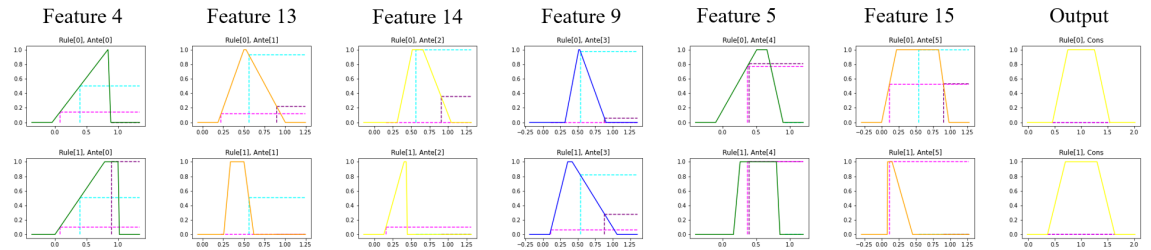


Figure 4.11: Training curves



(a) Ground truth anomalies



(b) Real anomalies

Figure 4.12: Found anomalies

Rule Id	Feature4				Feature13				Feature14									
1	[−0.045	0.841	0.844	0.882]	[0.186	0.5	0.526	1.002]	[0.309	0.514	0.659	1.042]						
2	[0.0005	0.79	1.002	1.024]	[0.2562	0.337	0.501	0.62]	[0.129	0.404	0.432	0.447]						
Feature9				Feature5				Feature15				Output						
[0.317				0.514	0.528	0.905]	[−0.098				0.51	0.661	0.899]	[0.001	0.219	0.831	0.983]	Inlier
[0.098				0.356	0.42	1.064]	[0.165				0.264	0.8	0.853]	[0.084	0.089	0.154	0.454]	Inlier

Table 4.3: Fuzzy rules

Anomaly	Feature4	Feature13	Feature14	Feature9	Feature5	Feature15
1 st ground truth anomaly	0.614	0.729	0.406	0.307	0.289	0.325
2 nd ground truth anomaly	0.523	0.65	0.281	0.23	0.266	0.227
3 rd ground truth anomaly	0.494	0.64	0.281	0.192	0.262	0.202
cyan, real anomaly	0.401	0.56	0.562	0.538	0.927	0.544
purple, real anomaly	0.899	0.899	0.906	0.884	0.393	0.902
magenta, real anomaly	0.082	0.224	0.156	0.115	0.37	0.116

Table 4.4: Values of anomalies

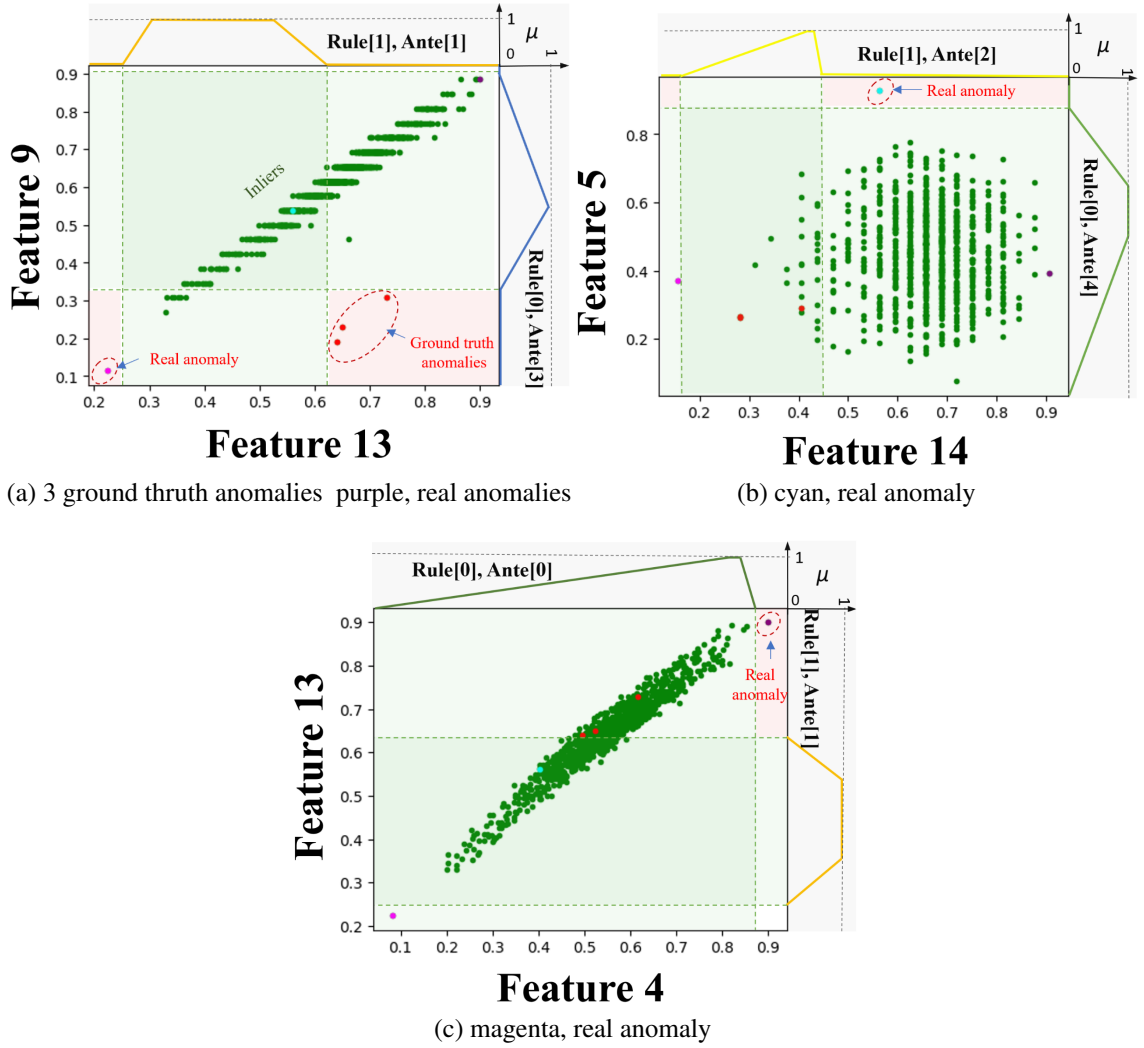


Figure 4.13: Fuzzy explainability

as discussed in Section 2.4 (Figure 4.12a).

In case of inliers the FRBS model has output value of 1, while in case of anomalies the output will be 0. In the Figure 4.12 the fuzzy membership functions in the Consequents are centered around 1, which means that if there is even a small activation in each Antecedent, the output of the model will be 1. The output will be 0, if at least one Antecedent with 0 activation degree exists in each Rule.

The Figure 4.12a shows that in the 0th Rule at the 3rd Antecedent (Rule[0], Ante[3]) and in the 1st Rule at the 1st Antecedent (Rule[1], Ante[1]) the relative importances are 0 for all three ground truth anomalies as observations. The 3rd Antecedent belongs to the *Feature*₉, while the 1st Antecedent belongs to the *Feature*₁₃. The Figure 4.13a shows the data projected onto the two dimensional plane defined by the *Feature*₁₃ and *Feature*₉. At the upper and at the right axes are presented the corresponding membership functions, and their rule in the decision making. If a data point is covered at least by one of the membership functions that it is an inlier, in case it is not covered by non of the membership functions than it is an anomaly. Note that in Section 2.4 the ground truth anomalies were depicted in the context of *Feature*₄ and *Feature*₉, but according to the correlation matrix in the Figure 4.10 the *Feature*₄ and *Feature*₁₃ are highly correlated, and as the fuzzy model (Figure 4.12a) suggested and the Figure 4.13a proved the ground truth anomalies can be described in this new context as well. Other than these three ground truth anomalies, the purple anomaly is also described in this context.

The Figure 4.12b presents the three found real anomalies as observations distinguished by colors cyan, magenta and purple. The anomaly with cyan has 0 activation degree in the Rule[0]-Ante[4] corresponding to the *Feature*₅ and also in Rule[1]-Ante[2] (*Feature*₁₄), Rule[1]-Ante[4], Rule[1]-Ante[5]. The Figure 4.13b presents the cyan anomaly in the context of *Feature*₅ and *Feature*₁₄, using the membership functions Rule[0]-Ante[4] and Rule[1]-Ante[2]. Similarly, in case of the magenta anomaly the Antecedents with 0 activation are the following: Rule[0]-Ante[0], Rule[1]-Ante[1] and Rule[1]-Ante[5] (Figure 4.12b), thus in the Figure 4.13c the dataset is projected onto the dimensions of *Feature*₄ and *Feature*₁₃.

Chapter 5

Conclusion and future work

The results of the architecture, optimized by the evolutionary algorithm, match well with the results of the manually adjusted architecture. The most optimal Autoencoder architecture yielded the optimum exactly at the expected location, demonstrating that bacterial optimization can be effective in finding the right architecture for model development.

Although the computational demand of bacterial optimization may be higher, it can help to find the optimal architecture depth for simple and even complex data sets, shortening the time that one has to invest in an uncertain number of test runs to find the right architecture. The iteration of the memetic algorithm is time-consuming, but it can be parallelized, reducing the training time further.

The article has been showed, how a fuzzy rule-based model that can be used for linear regression and classification problems. Due to the model's white-box behavior, we could see how the decision-making can be explained. The optimization process, the bacterial memetic algorithm, is also a strength of the presented method, since it combines the bacterial evolutionary algorithm with the Levenberg-Marquardt local search technique, which speeds up the learning procedure of a fuzzy system. In the case of the classification task, utilized in anomaly detection,

As a result, we saw how a fuzzy rule-based model combined with an autoencoder, can be used in anomaly detection tasks, how the fuzzy system makes the decisions, and also that how can one use this model in order to explain these decisions. In terms of explainability, the article presented how, in the case of the autoencoder, the reconstruction error can be used in order to interpret the relevant features (for anomalies), while in the case of the FRB system the trained model's rules themselves show the relevant feature dimensions, due to the white-box behavior. Furthermore, the fuzzy rules can be interpreted

as linguistic "*if-then*" constructions, and can be associated with linguistic interpretation.

A future work will be developing a fuzzy rule-based autoencoder, which can replace the currently used neural network-based autoencoder model, thus providing explainability regarding the data encoding and decoding. Furthermore, applying the presented Bézier curve abstraction (using curves of degrees higher than the presented quadratic) to other architecture optimization problem (other than autoencoders) is another promising topic.

Bibliography

- [1] Kazusuke Maenaka. “MEMS inertial sensors and their applications”. In: *2008 5th International Conference on Networked Sensing Systems*. 2008, pp. 71–73. DOI: 10.1109/INSS.2008.4610859.
- [2] Itilekha Podder, Tamas Fischl, and Udo Bub. “Smart Feature Selection for Fault Detection in the MEMS Sensor Production Process Using Machine Learning Methods”. In: *Proceedings of the 2nd International Conference on Industry 4.0 and Artificial Intelligence (ICIAI 2021)*. Atlantis Press, 2022, pp. 21–25. ISBN: 978-94-6239-528-2. DOI: 10.2991/aisr.k.220201.005. URL: <https://doi.org/10.2991/aisr.k.220201.005>.
- [3] Chandradip Patel and Patrick McCluskey. “Modeling and simulation of the MEMS vibratory gyroscope”. In: May 2012, pp. 928–933. ISBN: 978-1-4244-9533-7. DOI: 10.1109/ITHERM.2012.6231524.
- [4] L. Sujatha and B. Preethi. “Design and Analysis of MEMS Gyroscope”. In: 2010.
- [5] Majid Dadafshar. “Accelerometer and Gyroscopes Sensors: Operation, Sensing, and Applications”. In: 2019.
- [6] Xu Ru et al. “MEMS Inertial Sensor Calibration Technology: Current Status and Future Trends”. In: *Micromachines* 13.6 (2022). ISSN: 2072-666X. DOI: 10.3390/mi13060879. URL: <https://www.mdpi.com/2072-666X/13/6/879>.
- [7] Zhibo Ma et al. “Key Processes of Silicon-On-Glass MEMS Fabrication Technology for Gyroscope Application”. In: *Sensors* 18.4 (2018). ISSN: 1424-8220. DOI: 10.3390/s18041240. URL: <https://www.mdpi.com/1424-8220/18/4/1240>.
- [8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly detection: A survey”. In: *ACM Comput. Surv.* 41 (2009), 15:1–15:58.

- [9] Zhiyuan Cao et al. “Scalable Distribution Systems State Estimation Using Long Short-Term Memory Networks as Surrogates”. In: *IEEE Access* PP (Jan. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2967638.
- [10] N.E. Nawa and T. Furuhashi. “Fuzzy system parameters discovery by bacterial evolutionary algorithm”. In: *IEEE Transactions on Fuzzy Systems* 7.5 (1999), pp. 608–616. DOI: 10.1109/91.797983.
- [11] Csongor Márk Horváth et al. “Bacterial Memetic Algorithm Trained Fuzzy System-Based Model of Single Weld Bead Geometry”. In: *IEEE Access* 8 (2020), pp. 164864–164881. DOI: 10.1109/ACCESS.2020.3021950.
- [12] Charles Phiri et al. “Fuzzy rule-based model for outlier detection in a Topical Negative Pressure Wound Therapy Device”. In: *ISA Transactions* (Feb. 2021).
- [13] Charles Phiri et al. “Fuzzy rule-based model for outlier detection in a Topical Negative Pressure Wound Therapy Device”. In: *ISA Transactions* 117 (Feb. 2021). DOI: 10.1016/j.isatra.2021.01.046.
- [14] Dimitri Bertsekas. “Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey”. In: *Optimization* 2010 (July 2015).
- [15] Tong Zhang and Bin Yu. “Boosting with early stopping: Convergence and consistency”. In: *The Annals of Statistics* 33.4 (2005). DOI: 10.1214/009053605000000255. URL: <https://doi.org/10.1214/009053605000000255>.
- [16] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014).
- [17] Rabee Rustum et al. “Sustainability Ranking of Desalination Plants Using Mamdani Fuzzy Logic Inference Systems”. In: *Sustainability* 12 (Jan. 2020), p. 631. DOI: 10.3390/su12020631.
- [18] Rabee Rustum. “Modelling activated sludge wastewater treatment plants using artificial intelligence techniques (fuzzy logic and neural networks)”. In: 2009.
- [19] SN Sivanandam, Sai Sumathi, SN Deepa, et al. *Introduction to fuzzy logic using MATLAB*. Vol. 1. Springer, 2007.
- [20] János Botzheim et al. “Fuzzy Rule Extraction by Bacterial Memetic Algorithms”. In: *International Journal of Intelligent Systems* (Mar. 2009).

- [21] Manolis Lourakis. “A Brief Description of the Levenberg-Marquardt Algorithm Implemented by levmar”. In: *A Brief Description of the Levenberg-Marquardt Algorithm Implemented by Levmar* 4 (Jan. 2005).

List of Figures

2.1	MEMS gyroscope [3]	7
2.2	Acceleration with a single moving mass [5]	9
2.3	Feature correlations and projections	12
2.4	Correlation matrix	13
2.5	Ground truth anomalies	14
2.6	Autoencoder architecture [9]	15
2.7	Anomalies and the corresponding threshold values	15
2.8	Explainability using reconstruction error	16
2.9	Anomaly projections with different Σ values	17
3.1	Individual representation in the abstract space using Bézier curve	19
3.2	Abstract representation, Bézier curves	22
3.3	Bacterial Mutation [12]	24
3.4	Gene Transfer	25
3.5	Training models with different architectures	27
3.6	Error through evolution, $\lambda = 10^{-4}$	28
3.7	Error through evolution, $\lambda = 10^{-1}$	28
3.8	Error meshgrids	29
3.9	Solutions, abstract representations, Bézier curves	30
4.1	Fuzzy logic inference system [17]	31
4.2	Trapezoidal fuzzy set	32
4.3	Mamdani inference and center of gravity method	33
4.4	Bacterial Memetic Algorithm [20]	35
4.5	FRBS and the encoded bacterium	35
4.6	Linear regression using different number of rules	43
4.7	Fuzzy rules for linear regression, model with 3 rules	43
4.8	Fuzzy rules for classification	44

4.9	Rules for classification	44
4.10	Feature correlation	45
4.11	Training curves	46
4.12	Found anomalies	46
4.13	Fuzzy explainability	47

List of Tables

3.1	Examples	22
3.2	Solutions	30
4.1	Errors and the corresponding number of rules	41
4.2	Error drops	41
4.3	Fuzzy rules	47
4.4	Values of anomalies	47

List of Algorithms

1	Bacterial Evolutionary Algorithm [11]	18
2	Bacterial Mutation [11]	24
3	Gene Transfer [11]	25
4	Bacterial Memetic Algorithm [11]	34
5	Levenberg-Marquardt Algorithm [11]	36
6	Optimal Number of Rules [12]	41