UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DEPARTMENT OF INFORMATION ENGINEERING

Ingegneria Elettronica

A novel framework for vehicle functions identification by exploiting machine learning techniques

Supervisor
Prof. **Nicola Trivellin**

Co-supervisor
Dr. **Domenico Natella**

Candidate
Leonardo Lecco
ID 1233999

Academic Year 2021/2022

Contents

Abstract

Nowadays vehicles architectures exploit various automotive network protocols that bring information between the implemented Electronic Central Units (ECUs). Exchanged data are encoded and only Original Equipment Manufacturers (OEMs) and T1 (Tier One) producers know their meaning and how decode them. A software model will be developed in order to detect vehicles functions without having database files associated to network signals. Furthermore, the model will behave like an ECU by producing output signals related to input ones. Machine Learning techniques will be exploited, in particular Clustering task will be exploited to understand not a priori known vehicle functions and a Neural Network will be implemented to emulate an ECU behavior. Signals will be grouped in five different types of vehicle functions and the model will predict the ECU's output data with high accuracy. Applications concerning the developed project are, in primis, to fix up possible vehicles electronics faults. In addiction, vehicle predictive maintenance could be done. Another application, could be to check by OEMs if T1 manufacturers comply the required specification.

Introduction

Nowadays, in vehicles are increasingly implemented new Electronic Central Units (ECUs) with new functions. The most popular automotive network protocols are CAN, FlexRay, LIN and MOST. Messages exchanged by ECUs have a different structure from a network protocol to another. In the chapter 1 will be explained in detail frame structures and operations of the above mentioned protocols, starting from an overview of the automotive prototyping process description.

Network messages are encoded. To decode them, the so called *database files* are needed. Usually, only Original Equipment Manufacturers (OEMs) and T1 (Tier One) have the database files. It might be very useful to know messages and signals meaning without owning database files in order to fix up possible vehicles electronics faults. In addiction, vehicle predictive maintenance could be done.

In chapter 3, a system able to decode network messages without database files will be implemented. The system will firstly group messages read from a set of CAN lines based on some intrinsic features of their frames. In particular, signals will be grouped in the five following membership categories: High Voltage batteries and power electronics systems, suspensions and vehicle dynamic controls, dashboard and driver's vehicle interface, driver's comfort and lighting system, and powertrain and engine controls. Then, the system will be trained in order to detect the relationship between input and output signals of a sample ECU. This task could be useful to understand an ECU behavior, seen as a black box.

In order to implement the system, Machine Learning techniques described in chapter 2 will be exploited. In particular, clustering task will be exploited to understand not a priori known vehicle functions by implementing k-means algorithm. For the task of emulating an ECU behavior, a classification neural network will be implemented. For the neural network model choice, various hyperparameters will be tried and the best training model will be chosen.

Chapter 1

Automotive control units integration

The following chapter firstly describes the automotive prototyping process, in order to understand in which phase automotive software applications are developed and in which phase they are implemented in new vehicles. Then, an overview of automotive network protocol typologies is given. It will focus mostly on CAN and FlexRay protocols since they will be taken into account in chapter 3.

1.1 Automotive prototyping process

1.1.1 V-model

Starting from the idea of a new vehicle, the model at the basis of the vehicle realization is the V-model. The V-model is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.

As shown in figure 1.1, the left side of the graph represents *verification* processes while the right side represents *validation* processes. Each phase belonging to the verification process has an associated phase in the validation process.

1.1.2 Verification and validation processes

The PMBOK guide[1], also adopted by the IEEE as a standard (jointly maintained by INCOSE, the Systems engineering Research Council SERC, and IEEE Computer Society) defines the **verification process** as:

The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition.

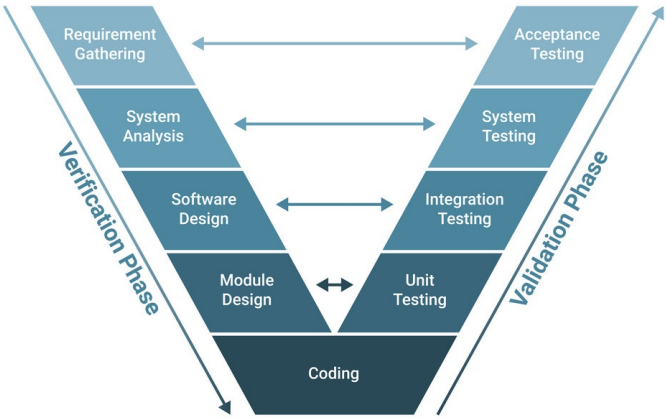


Figure 1.1: Graph of V-model phases

The different Verification phases in a V-Model are explained in detail below.

Requirement gathering This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing. At this stage will be defined the tasks that the automotive functions and features will perform and how.

System analysis The next step is the actual design and development of the function/feature for which the requirement was gathered in the first phase. In the development phase, the functionalities are designed and tested using the model-based development environment[1]. The function/functionality needs to be tested as it is being developed, ensuring that bugs and errors are fixed early on.

Software Design Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

Module Design In this phase, the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and

[1]In the model-based development environment, simulation tools can simulate real-world scenarios. The potential bugs and errors are highlighted and rectified during this phase. These tests are called Model in the Loop (MIL) since the testing is done in a controlled environment using models. Once the development process and testing are complete and the results are satisfactory, the model – a block diagram – is then sent to the software development team.

errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

The PMBOK guide[1], defines the **validation process** as:

The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers.

The different validation phases in a V-Model are explained in detail below.

Unit Testing Unit tests designed in the design phase module are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing Integration testing is associated with the architectural design phase. Since there are different ECUs, during the software integration phase will be checked their correct behavior and the correct interaction between them. The legacy code will be replaced with the proper software version.

System Testing System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. There will also be checked that the newly integrated function won't affects any other modules' functioning. The testing here is done in a physical environment i.e., in-vehicle test. The most important aspect is ensuring that the implementation was correctly done without unwanted consequences.

Acceptance Testing Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

1.1.3 Maturity stages

Maturity level assurance is a control method within project management, initiated by the customer[2]. By applying specified rules, both the suppliers of products and the customer's internal organization are involved jointly at an early stage in the product creation process. Additional work and delays can be avoided where the elements of maturity level assurance are consciously integrated at an early stage in project management.

The principal objective of the maturity level assurance method is to improve the launch quality, delivery quality and field quality of the products supplied, by harmonizing the contents and operations in the supply chain. By employing a structured procedure which interrogates the status of defined measurement criteria relating to the associated maturity levels, which in turn are oriented to the milestones of the overall project plan established by the automobile manufacturer, the agreed quality of the products to be supplied is assured.

Maturity levels are aligned both with the project schedule for the vehicle (or the customer's product) and the project-specific timings for the products to be delivered. They can easily be integrated into the associated product creation processes established by the OEMs or the suppliers.

The maturity of the hardware components increases during the pre-production phase until the component reaches production quality (PPAP). The maturity level is typically indicated on four different stages explained below.

Testing mules are functional vehicles that have the aim to illustrate basic functions. They belong to the first stage of prototyping and they implement only limited and very basic functions. The use on public roads is not permissible in order to guarantee driving safety. Purpose of use is to do laboratory tests, simulation, and to obtain confirm of the functional principles.

Prototypes are functional and drivable vehicles made with auxiliary tools that provide sufficient operational safety for initial tests under driving conditions inside the vehicle and on the test bench. Use on public roads is possible with a special permit and once street clearance has been granted by the competent authority. Purpose of use is to testing the prototype in order to verify that product requirements have been met. Are also carried out durability tests including detection and elimination of remaining vulnerabilities.

Zero series (0S) are components made with production tools, and no technical restrictions that have an impact on driving safety are allowed. Use on public roads is possible with a special permit and once street clearance has been granted by the competent authority. Purpose of use is to testing for achieving technical release.

Series Of Production (SOP) are produced entirely with production tools and under full production conditions (parts made with production tools). Samples with which the production process and product approval process is carried out, with the aim of achieving a release for full production (PPA process).

1.2 Network protocols

1.2.1 Typologies of network protocols

From firsts vehicle model, who were born on the last years of the 19th century, to the 70s, main goal during years was to improve mechanical components. Starting from 70s to nowadays, with the coming of the information age an

exponential increase in the number of electronic systems occurred in vehicles. This intensive growing of electronic systems concerns ADAS, safety, infotainment and mechanical replacement systems.

Regarding safety task, one of the main purposes of electronic systems is to assist the driver to control the vehicle through functions related to the entire vehicle systems. For example, regarding steering, traction (i.e., distribution of the driving torque), braking systems, such as the ABS (Anti-lock Braking System), ESP (Electronic Stability Program), EPS (Electric Power Steering), active suspensions, ADAS, such as ACC (Adaptive Cruise Control), engine control and so on. Another reason for using electronic systems is to control devices in the body of a vehicle such as lights, wings, doors, windows and, recently, entertainment and communication equipment (e.g., radio, dashboard display, devices wireless connections, navigation systems, etc.).

In the early days of automotive electronics, each new function was implemented as a stand-alone ECU. With the exponential growing of different functions, this approach quickly proved to be insufficient. The main issue of this approach is the high number of ECUs, which leads point-to-point links between them. This high amount of connections implies too much weight, cost, complexity and reliability of the wiring, since it requires a number of wiring connections of the order of n squared[2], where n is the number of ECUs. These issues motivated the use of networks where the communications are multiplexed over a shared medium, which consequently required defining rules, called protocols, for managing communications and, in particular, for granting bus access.

In 1986, Robert Bosch GmbH introduced the Controller Area Network (CAN) serial bus system at the Society of Automotive Engineers (SAE) congress [3], which was integrated in vehicles starting from the 1990s. Today, the CAN bus protocol has become the most widely used network in automotive systems. Since different needing of bandwidth, cost and dependability[3], in 1994, the Society for Automotive Engineers (SAE) defined a classification for automotive communication protocols [5]. Four different classes subsist based on data transmission speed and functions that are distributed over the network.

- **Class A** networks are used to transmit simple control data with low-cost technology. They have a data rate lower than 10 Kbit/s.
- **Class B** networks are dedicated to supporting data exchanges between ECUs in order to reduce the number of sensors by sharing information. They operate from 10 Kbit/s to 125 Kbit/s.
- **Class C** networks are used for applications that need high speed real-time communications. They operate from 125Kbit/s to 1Mbit/s.

[2]If each node is interconnected with all the others, the number of wiring links between ECUs is $\sum_{i=1}^{n} i = n(n+1)/2 \in O(n^2)$, which leads hundreds of millions connections by considering that nowadays in an entire vehicle are implemented thousands of different functions.

[3]The quality of being able to be trusted and being very likely to do what people expect [4].

- **Class D** networks are used for applications that need ultra-high speed real-time communications. They operate at a speed over 1Mbit/s.

Example of Class C network is high-speed CAN, used for powertrain and for chassis domain, while an example of Class D networks is MOST, involved on multimedia data, and FlexRay, for safety and critical applications.

1.2.2 Time-triggered and event-triggered protocols

One of the main objectives of the design step of an in-vehicle embedded system is to ensure a proper execution of the vehicle functions, with a pre-defined level of safety, both in the normal functioning mode and when some components fail or when the environment of the vehicle creates perturbations. Networks play a key role in maintaining the embedded systems in a *safe* state since most critical functions are now distributed and need to communicate. Thus, the different communication systems have to be examined for this objective. In particular, messages transmitted on the bus must meet their real-time constraints, which mainly consist on bounded jitters and bounded response times.

In automotive systems, there are two kind of communications: *event-triggered* and *time-triggered*. Event-triggered means that messages are transmitted in correspondence to the changing of a signal or a significant event. In this type of communication, the system has permission to take into account, as quickly as possible, any asynchronous events. An example could be to turn on the inside light when a car door is opened. The communication protocol must define a policy to grant access to the bus in order to avoid collisions. For instance, the strategy used in CAN is to assign a priority to each frame and to give the bus access to the highest priority frame. Event-triggered communication is very efficient in terms of bandwidth usage since only necessary messages are transmitted. Furthermore, the evolution of the system without redesigning existing nodes is generally possible which is important in the automotive industry where incremental design is a usual practice.

In time-triggered communications, frames are transmitted at predetermined time instants. This is well-suited for messages periodic transmission as it is required in distributed control loops. Each frame is scheduled for transmission at one pre-defined interval of time, usually termed a slot, and the schedule repeats itself indefinitely. This medium access strategy is referred to as Time Division Multiple Access (TDMA). A property of time-triggered protocols is that missing messages are immediately identified. This is useful to detect, in a short and bounded amount of time, nodes that are presumably no longer operational. The first negative aspect is the inefficiency in terms of network utilization and response times with regard to the transmission of aperiodic messages (i.e. messages that are not transmitted in a periodic manner). A second drawback of time-triggered protocols is the lack of flexibility even if different schedules (corresponding to different functioning modes of the application) can be defined and switching from one mode to another is possible at run-time. There exist also networks that support both event-triggered and time-triggered

types of communication, for example FlexRay, Time-Triggered Controller Area Network (TT-CAN) and Flexible Time-Triggered on Controller Area Network (FTT-CAN).

1.3 In-vehicle communication systems

In this section there will be shown the most important and diffused network protocols. In particular there will be explained CAN bus, FlexRay, LIN and MOST protocols. These kind of protocols differ to each other in the triggered typology, in data rate and in cost. CAN and FlexRay are used predominantly in driver assistance and functional safety applications because of their reliability and robustness. MOST is used predominantly for infotainment applications because of its high throughput.

1.3.1 CAN network

Nowadays, Controller Area Network (CAN) is the most diffused protocol for in-vehicle communication. CAN is a priority based protocol implemented as a SAE class C network for real-time control in chassis and powertrain domains (at 250 or 500KBit/s), or as a SAE class B network for the electronics in the body domain, usually at a data rate of 125Kbit/s. The priority based feature leads to an event-triggered communication typology.

DATA LINK LAYER	LLC – LOGICAL LINK CONTROL	• Acceptance filtering • Overload notification • Recovery management
	MAC – MEDIUM ACCESS CONTROL	• Data encapsulation decapsulation • Frame coding • Error detection • Serialization Deserialization
PHYSICAL LAYER	PHYSICAL SIGNALING	• Bit encoding decoding • Bit timing/ synchronization
	PHYSICAL MEDIUM ATTACHMENT	• Driver/ Receiver characteristics
	PHYSICAL DEPENDENT INTERFACE	• Connectors Wires

Figure 1.2: CAN protocol layers

In figure 1.2 is shown the layer structure of CAN bus. Regarding physical layer, CAN bus consists on a pair of twisted wires, called *CANL* (CAN Low) and *CANH* (CAN High). The electrical characteristics of the CAN bus cable restrict the cable length according to the selected data rate. This kind of cabling leads to the bus a greater robustness towards electric disturbances and electromagnetic interference, therefore it is ideal for safety critical applications.

Table 1.1: CAN bit timing

Bit-Rate	Bus length
1 Mbit/s	25 m
800 Kbit/s	50 m
500 Kbit/s	100 m
250 Kbit/s	250 m
125 Kbit/s	500 m
50 Kbit/s	1000 m
20 Kbit/s	2500 m
10 Kbit/s	5000 m

A typical cabling length related to data rate is reported in table 1.1. The shorter the length, the greater the data rate. The signal must be able to propagate to the most remote node and return back before the bit value is decided. This requires the bit time to be at least twice as long as the propagation delay which limits the data rate.

CAN bus must be terminated at both ends with 120Ω resistors matching the impedance of the cable. Sometimes, the termination resistor is split in two 60Ω resistors and an additional capacitor is connected to the ground. An advantage of this bus configuration is that new subsystems can be added to a system without modification of already existing system. This leads that each subsystem can be upgraded with new hardware and software at any time, encouraging the aftermarket purchase and installation of new subsystems.

To ensure the freshness of the exchanged data during the bus working, it is important that the protocol can ensure bounded response times of frames. For this reason, each message is assigned an identifier, unique to the whole system. This leads two facts: giving priority for transmission and allowing message filtering upon reception.

Since each frame corresponds to a message, each frame has a unique identifier (ID) which are transmitted together. The priority of a message is given by the numerical value of its identifier. The lower the numerical value of an ID, the higher the priority. CAN messages are divided in two groups differing in the size of the identifier. *Standard CAN* (or CAN 2.0A) has 11 bit identifiers, which leads to $2^{11} = 2048$ different IDs. *Extended CAN* has 29 bit identifiers, which leads to 2^{29} different IDs. In-vehicle communications exploits standard CAN since it provides a sufficient number of different identifiers, i.e., different messages. Bit stuffing is an encoding method that enables resynchronization when using Non-Return-to-Zero (NRZ) bit representation where the signal level on the bus can remain constant over a longer period.

The mechanism by which is determined which peripheral will communicate on the bus, is called bit-wise arbitration. The involved network participants (nodes) observe the signal level bit by bit at the configured sample point. Since the identifier is transmitted most significant bit first, the node with the numerically lowest identifier numeric value will gain access to the bus. This

happens in accordance with the wired-and-mechanism, by which the dominant state overwrites the recessive state. Nodes with recessive transmission and dominant observation lose the competition for network access. All those losers automatically become receivers of the currently transmitted CAN data frame, with the highest priority. Nodes that have lost the arbitration will attempt to re transmit their CAN data frame after the current transmission is finalized and the network is available again[6].

CAN requires the physical layer to implement the logical *and* operator: since lower identifiers have higher priority, a bit-per-bit logical *and* is applied on different IDs in order to determine the bit-wise arbitration.

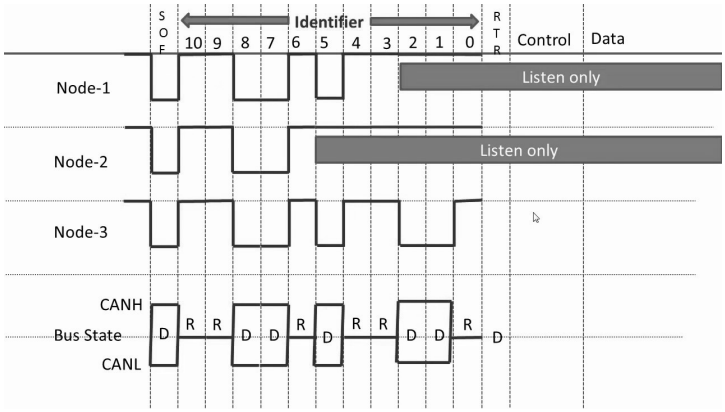


Figure 1.3: CAN arbitration example

An example of bit-wise arbitration is reported in figure 1.3. At bit 5^{th} comparison, Node-1 loses the arbitration. At bit 2^{nd} comparison, Node-2 loses and Node-3 wins the arbitration.

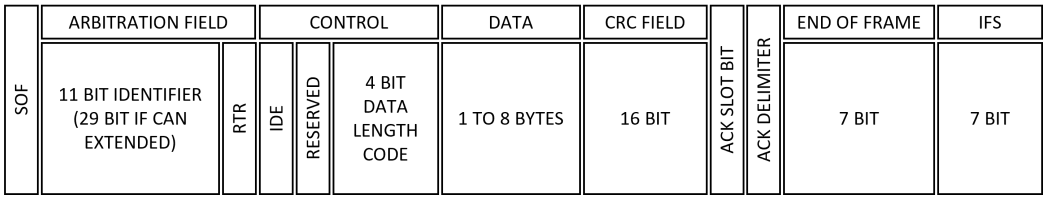


Figure 1.4: CAN frame structure

In figure 1.4 is shown the structure of a CAN data frame.

- **SOF**: Start of Frame bit marks start of message and it is used to synchronize nodes on the bus.
- **ID**: identifier is a 11 bit (or 29 bit if CAN extended) field. It establishes priority of messages.
- **RTR**: Remote Transmission Request bit is dominant if the frame is a *data* frame (node requires information from another remote node) or recessive if the frame is a *remote* frame (the node receives a request and all the nodes receive reply).

- **IDE**: Identifier Extension bit indicates standard CAN frame is being transmitted instead of an extended CAN frame.
- **Reserved** bit for possible use by future standard amendment.
- **DLC**: Data Length Code indicates number of bytes to be transmitted over the CAN bus.
- **Data**: Contains up to 64 bits grouped in different bytes of transmitted application data.
- **CRC**: Cyclic Redundancy Check, used for error detection. It holds checksum for application data preceding to it.
- **ACK**: Acknowledgement field is grouped in 2 bits and contains first bit as acknowledgement bit and second bit as delimiter. Each node uses this field to show integrity of its data. Node receiving correct message overwrites this bit in original received message with dominate bit as mentioned above to indicate error free message has been transmitted. The node receiving erroneous message leaves this bit as recessive. Moreover it discards the message and hence prompts the sending node to re-transmit the message after re-arbitration process.
- **EOF**: End of Frame is a 7 bits size field that marks end of CAN frame (or message).
- **IFS**: Interframe Space is a 7 bits size field that contains time required by controller to move correctly received frame to its proper position in message buffer area.

All nodes on the CAN bus must be synchronized to sample every bit on the CAN bus at the same time. In order to adjust the on-chip bus clock, the CAN controller may shorten or prolong the bit time period. The maximum value of these bit time adjustments are termed the Synchronization Jump Width (SJW). Types of synchronizations that take place during a communication three. *Hard synchronization* occurs on the recessive-to-dominant transition of the start bit after a period of bus idle. The bit time is restarted from that edge. *Resynchronization* occurs when a bit edge doesn't occur within the synchronization segment in a message. One of the Phase Segments are shortened or lengthened with an amount that depends on the phase error in the signal; the maximum amount that may be used is determined by the Synchronization Jump Width parameter. *Adjustment* occurs when a transition does not take place at the exact time the controller expects it, the controller adjust its nominal bit rate accordingly.

1.3.2 FlexRay network

The FlexRay Consortium, whose includes BMW, DaimlerChrysler, General Motors, Motorola, Philips, and Bosch companies, developed FlexRay protocol. The goal of this consortium was to develop an OEM-independent, deterministic

and fault tolerant FlexRay communication standard, which each member of the consortium can use without having to pay licensing fees[7].

FlexRay communication protocol is a time-triggered protocol. Since in time-triggered protocols activities are driven by the progress of time while event-triggered activities are driven by the occurrence of events, in general, dependability is much easier to ensure using a time-triggered bus. For this reason, time-triggered communication systems are adopted for X-by-Wire applications.

In this category, multi-access protocols based on TDMA (Time Division Multiple Access) are particularly well suited; they provide deterministic access to the medium (the order of the transmissions is defined statically at the design time), and thus bounded response times. The three principal TDMA based networks that could act as gateways or for supporting safety critical applications are FlexRay, TTCAN and TTP/C (although the last one is no more used in vehicle applications).

The FlexRay network is very flexible with regard to topology and transmission support redundancy[4]. FlexRay is a TDMA based networks, thus it could behaves as gateway or it can support safety critical applications. At the MAC level, FlexRay defines a communication cycle as the concatenation of a time-triggered (or static) window and an event-triggered (or dynamic) window.

The *time-triggered* window uses a TDMA MAC protocol with the particularity that might possess several slots, but the size of each slot is the same. During the *event-triggered* communication cycle, the protocol is FTDMA (Flexible Time Division Multiple Access): the time is divided into so-called mini-slots. Different nodes can send frames in the same slot but in different cycles, this is called *slot multiplexing*. Each node possesses a given number of mini-slots and it can start the transmission of a frame inside each of its own mini-slots. A mini-slot remains idle if the node has nothing to transmit, that leads a loss of bandwidth.

A FlexRay frame is composed by three parts: the header, the payload segment and the CRC[8]. *Header* is 5 bytes (40 bits) long and includes the following fields:

- Status Bits (5 bits)
- Frame ID (11 bits)
- Payload Length (7 bits)
- Header CRC (11 bits)
- Cycle Count (6 bits)

The Frame ID defines the slot in which the frame should be transmitted and is used for prioritizing event-triggered frames. The Payload Length contains

[4]The definition of *transmission redundancy* given in the standard ISO26262 is: *The information is transferred several times in sequence. The repetition is effective only against transient failures.*

the number of words which are transferred in the frame. The Header CRC is used to detect errors during the transfer. The Cycle Count contains the value of a counter that advances incrementally each time a Communication Cycle starts.

Payload contains the actual data transferred by the frame. The length of the FlexRay payload or data frame is up to 127 words (254 bytes), which is over 30 times greater compared to CAN. *Trailer* contains three 8-bit CRCs to detect errors.

In the automotive context where critical and non-critical functions will increasingly co-exist and inter operate, this flexibility can prove to be efficient in terms of cost and re-use of existing components if missing fault-tolerance features are provided in a middle ware layer.

1.3.3 LIN network

Local Interconnect Network (LIN) is a low cost serial communication system used as SAE class A network. The low-cost objective is achieved not only because of the simplicity of the communication controllers but also because the requirements set on the micro-controllers driving the communication are reduced (i.e., low computational power, low-cost electrical components, small amount of memory). Typical applications involving these networks include controlling doors (e.g., door locks, opening/closing windows) or controlling seats (e.g., seat position motors, occupancy control). LIN is developed by a set of major companies from the automotive industry (e.g., DaimlerChrysler, Volkswagen, BMW and Volvo) and is already widely used in production cars.

A LIN cluster consists of one *master* node and several *slave* nodes connected to a common bus. For achieving a low-cost implementation, the physical layer is defined as a single wire with a data rate limited to 20Kbit/s due to EMI (Electromagnetic Interference) limitations. The master node decides when and which frame shall be transmitted according to the schedule table. The schedule table is a key element in LIN. It contains the list of frames that are to be sent and their associated frame-slots thus ensuring determinism in the transmission order.

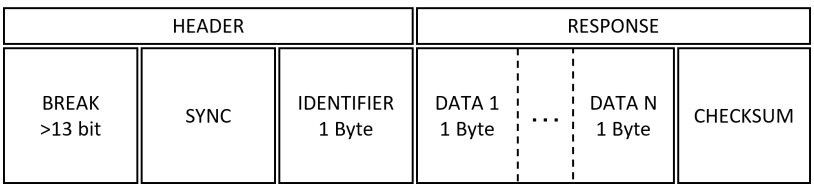


Figure 1.5: LIN frame structure

Any node interested can read a data frame transmitted on the bus. In figure 1.5 is shown the structure of a LIN data frame.

- **Break**: Is composed by 13 dominant bits followed by a break delimiter of one recessive bit. This serves as a start-of-frame notice to all nodes on the bus.
- **Sync**: It allows slave devices that perform automatic baud rate detection to measure the period of the baud rate and adjust their internal baud rates to synchronize with the bus.
- **ID**: It provides identification for each message on the network and ultimately determines which nodes in the network receive or respond to each transmission. All slave tasks continually listen for ID fields, verify their parities, and determine if they are publishers or subscribers for this particular identifier. The LIN bus provides a total of 64 IDs. IDs 0 to 59 are used for signal-carrying (data) frames, 60 and 61 are used to carry diagnostic data, 62 is reserved for user-defined extensions, and 63 is reserved for future protocol enhancements.
- **Data**: Is transmitted by the slave task in the response. This field contains from one to eight bytes of payload data bytes.
- **Checksum**: The LIN bus defines the use of one of two checksum algorithms to calculate the value in the eight-bit checksum field. Classic checksum is calculated by summing the data bytes alone, and enhanced checksum is calculated by summing the data bytes and the protected ID.

LIN defines five different frame types: unconditional, event-triggered, sporadic, diagnostic and user-defined[9]. *Unconditional* frames are the usual type of frames used in the master-slave dialog and are always sent in their frame slots. *Sporadic* frames are frames sent by the master, only if at least one signal composing the frame has been updated. Usually, multiple sporadic frames are assigned to the same frame-slot and the higher priority frame that has an updated signal is transmitted. An *event-triggered* frame is used by the master willing to obtain a list of several signals from different nodes. A slave will only answer the master if the signals it produces have been updated, thus resulting in bandwidth savings if updates do not take place very often. If more than one slave answers, a collision will occur. The master resolves the collision by requesting all signals in the list one by one. *User-defined* frames may carry any type of information.

1.3.4 MOST network

MOST (Media Oriented System Transport) is a serial communication system for transmitting audio, video and control data via fiber-optic cables[12]. It is a high-speed multimedia network protocol optimized by the automotive industry, used also in other applications. This technology was introduced in 1998 by the MOST Cooperation[10], that was founded as a German civil law partnership by BMW, Daimler Benz, Becker and OASIS Silicon System. Audi joined the founding group after the foundation.

The MOST specification not only defines the Physical Layer and Data Link Layer; it covers all seven layers of the ISO/OSI Reference Model for data communication[12]. Uniform interfaces simplify implementation of the MOST protocol in multimedia devices. A MOST network, usually laid out as a ring, may include up to 64 MOST devices. Due to its plug&play functionality it is not very difficult to either add or remove a MOST device. Given suitable topological conditions stars may also be laid out - and double rings may be used for safety critical applications.

A MOST frame consists of header, synchronous data channel, asynchronous data channel, control frame, and trailer[11]. *Header* consists on the start of frame. *Synchronous channel* is for transmission of synchronous data, sound and movie data. *Asynchronous channel* is for transmission of asynchronous data and internet packet data. *Control data* field is for hardware control data and turn on or turn off the device in MOST network. *Trailer* is the end of MOST frame and it has parity bits to check MOST frame.

Chapter 2

Machine Learning techniques

In this chapter will be explained needful Machine Learning (ML) techniques to implement tasks on chapter 3. An overview on what is a machine learning model will be firstly explained, by focusing on how to design it starting from the basis. Then, the clustering task will be explained by introducing clustering typologies and by focusing on k-means algorithm. Finally, neural networks will be treated by showing their structures and by describing their main hyperparameters typologies.

2.1 Machine learning introduction

2.1.1 Machine learning model

Machine Learning is the subset of artificial intelligence[1] (AI) that focuses on building systems that learn or improve performance-based on the data they consume[13].

Algorithms are the engines that power machine learning. In general, two major types of machine learning algorithms are used today: *supervised learning* and *unsupervised learning*. The difference between them is defined by how each learns about data to make predictions.

Supervised learning
Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training data set, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly[14].

Unsupervised learning
Unsupervised machine learning algorithms are used when the information used

[1] Artificial intelligence is a broad term that refers to systems or machines that emulate human intelligence.

to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from data sets to describe hidden structures from unlabeled data[14].

Machine learning algorithms has access to:

Domain set $\mathcal{X}$	Set of objects to make prediction about
Label set $\mathcal{Y}$	Set of possible labels
Training set $\mathcal{S} = \mathcal{X} \times \mathcal{Y}$	Finite sequence of labeled domain points in $\mathcal{X} \times \mathcal{Y}$

Prediction rule, or simply *predictor*, is the output of a machine learning algorithm. It consists on the rule learned by the algorithm by which it assign a label to an unlabeled object. It can be defined *error of prediction rule* the probability of not predict the correct label on a random data point. Goal of machine learning algorithms is to minimize the error of prediction rule.

A machine learning training model is a process in which a machine learning algorithm is fed with sufficient training data to learn from[15]. In the following, most crucial steps involved in creating a machine learning model are pointed[16].

1. **Defining the problem:** Defining the problem statement is the first step towards identifying what an ML model should achieve. This step also enables recognizing the appropriate inputs and their respective outputs.

2. **Data collection:** It is necessary to investigate and gather data that can be used to feed the machine. This is an important stage in the process of creating an ML model because the quantity and quality of the data used will decide how effective the model is going to be. Data can be gathered from pre-existing databases or can be built from the scratch.

3. **Data preparation:** The data preparation stage is when data is profiled, formatted and structured as needed to make it ready for training the model. This is the stage where the appropriate characteristics and attributes of data are selected. This is also at the stage where data is categorized into two groups - one for training the ML model and the other for evaluating the model. Pre-processing of data by normalizing, eliminating duplicates and making error corrections is also carried out at this stage.

4. **Assigning appropriate model:** Picking and assigning a model or protocol has to be done according to the objective that the ML model aims to achieve. The choice of models largely depends on the type of data that is being used.

5. **Model training:** This is the stage where the ML algorithm is trained by feeding datasets. This is the stage where the learning takes place. Consistent training can significantly improve the prediction rate of the

ML model. The weights of the model must be initialized randomly. This way the algorithm will learn to adjust the weights accordingly.

6. **Evaluating and defining measure of success:** The machine model have now to be tested. This helps assess the accuracy of the model. Identifying the measures of success based on what the model is intended to achieve is critical for justifying correlation.

7. **Parameter tuning:** Selecting the correct parameter that will be modified to influence the ML model is key to attaining accurate correlation. The set of parameters that are selected based on their influence on the model architecture are called *hyperparameters*. The process of identifying the hyperparameters by tuning the model is called parameter tuning.

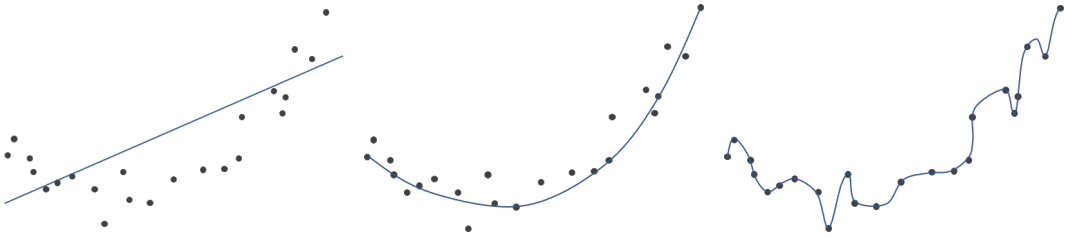
2.1.2 Underfitting and overfitting

There are two situations of which the model's performance might suffer from. *Overfitting* and *underfitting* are the two biggest causes for poor performance of machine learning algorithms.

Overfitting refers to a model that models the training data too well. It happens when a model learns details and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns[17].

Underfitting refers to a model that can neither model the training data nor generalize[2] to new data. An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data. Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting[17].

[2]Generalization is the model's ability to give sensible outputs to sets of input that it has never seen before[18].



(a) Underfitting (b) Good approximation (c) Overfitting

Figure 2.1: Example of curve underfitting, overfitting and a good fit

Figure 2.1a shows an example of underfit model: a too simply curve has been predicted by the algorithm. It has poor performance on the training data and poor generalization to other data. Figure 2.1c shows an example of overfit model: data has been learned too much by the algorithm. It has good performance on the training data, but poor generalization to other data. Figure 2.1b represent the right balance between underfitting and overfitting situations, that implies a good result.

2.1.3 ML model training

In order to fit as well the machine learning model, the entire dataset will be split in three parts: *training set*, *validation set* and *test set*.

1. Training set

Training set is a data set of samples used during the learning process and it is used to fit the model parameters. Usually, it is made up from the majority of dataset samples.

2. Validation set

Validation set is the samples of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration[19].

3. Test set

The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

Supervised learning algorithm can be divided in two main categories, based on the type of prediction it want to be implemented. These two predictors are called *classification* and *regression* tasks.

Classification is the process of predicting the class of given data points. Classification predictive modeling is the task of approximating a mapping function from input variables to discrete output variables. Label set is a finite set of discrete values. It could be divided in two typologies, *binary* classification and *multi-class* classification. Binary classification refers to those tasks which can give either of any two class labels as the output. Generally, one is considered as the normal state and the other is considered to be the abnormal

state[20]. The most common algorithms which are used for binary classification are K-Nearest Neighbours, Logistic Regression, Support Vector Machine, Decision Trees, Naive Bayes. Multi-class classification problems have no fixed two labels but can have any number of labels. The most common algorithms which are used for Multi-Class Classification are K-Nearest Neighbours, Naive Bayes, Decision trees, Gradient Boosting, Random Forest.

Regression algorithms attempt to estimate the mapping function from the input variables to numerical continuous output variables. Since output is a real value, regression tasks are usually exploited to predict quantities or sizes. The most common algorithms which are used for regression are linear regression, decision tree, support vector regression, Lasso regression and random forest.

Unsupervised learning algorithms can be categorized into two types of problems - *clustering* and *association*.

Clustering is a method of grouping objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities[21].

Association rule is an unsupervised learning method used for finding the relationships between variables in a large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy an item (suppose pizza) are also tend to purchase an associated item (for example, beer)[21].

2.2 Clustering

2.2.1 Clustering types

A particular focus will be done for clustering task since this technique will be exploited in chapter 3.2. Clustering is an important concept when it comes to unsupervised learning. As told before, clustering mainly deals with finding a structure or pattern in a collection of uncategorized data. Unsupervised learning clustering algorithms will process data and find natural clusters or groups if they exist in the data set[22]. Several approaches to clustering exist. Most important typologies are listed below.

- **Centroid-based** comprehends iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. k-means clustering algorithm is a popular algorithm that falls into this category. In these models, the number of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima[23].

- **Density-based** clustering connects areas of high example density into clusters. This allows for arbitrary-shaped distributions as long as dense areas can be connected. These algorithms have difficulty with data of varying densities and high dimensions. Further, by design, these algorithms do not assign outliers to clusters[24].
- **Distribution-based** clustering approach assumes data is composed of distributions, such as Gaussian distributions. As distance from the distribution's center increases, the probability that a point belongs to the distribution decreases[24]. If data probability distribution is unknown, this type of clustering is not recommended.
- **Hierarchical** clustering starts by treating each observation as a separate cluster. Then, it repeatedly executes the following two steps: identify the two clusters that are closest together and merge the two most similar clusters. This iterative process continues until all the clusters are merged together. The main output of hierarchical clustering is a dendrogram[3], which shows the hierarchical relationship between the clusters[25].

2.2.2 *k*-means algorithm

k-means algorithm, also known as Lloyd's algorithm, clusters data by trying to separate samples in k groups of equal variance, minimizing a criterion known as the *inertia* or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields. k-means algorithm divides a set of samples into disjoint clusters, each described by the mean of the samples in the cluster. The means are commonly called the cluster *centroids*. They are not, in general, points from, although they live in the same space[27].

Given the following quantities

$\mathcal{X} \subset \mathbb{R}^n$	Set of vectors to be clustered
$\boldsymbol{x} \in \mathcal{X}$	Vector to be clustered
k	Number of clusters
C_i, $i \in [1, k] \subset \mathbb{N}$	Clusters
$\boldsymbol{\mu}_i$, $i \in [1, k] \subset \mathbb{N}$	Centroids of the clusters

Task of k-means algorithm is to find cluster centers and allocations in order to minimize the error made by approximating the points with the cluster centers:

$$\boldsymbol{\mu}_i(C_1) = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}\|^2$$

k-means algorithm is suitable to solve the previous task.

[3]A dendrogram is a type of tree diagram showing hierarchical relationships between different sets of data. It maintains a memory of algorithm clusters during each iteration and it stores them in the dendrogram[26].

Algorithm 1 k-means

Require: $\mathcal{X}$, k
Ensure: $\boldsymbol{x}_j \in C_i$, $\forall \boldsymbol{x}_j \in \mathcal{X}$, $i = 1, ..., k$

Select k random centroids

while algorithm *converges* **do**

Each point is associated to the closest centroid:

$$\forall i : C_i = \{\boldsymbol{x} \in \mathcal{X} : i = \underset{j}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{\mu}\|\}$$

Compute new centroids as the barycentre of the associated points:

$$\forall i : \boldsymbol{\mu}_i = \frac{\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}}{|C_i|}$$

end while

Algorithm 1 represents k-means algorithm. In basic terms, the algorithm has three steps. The first step chooses the initial centroids, with the most basic method being to choose k samples from the dataset $\mathcal{X}$. After initialization, k-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The spatial difference between the new centroid and the previous one is computed and the algorithm repeats these last two steps until this value become less than a set threshold. In other words, it repeats until the centroids do not move significantly[27] or until other convergence situations occur.

It convergence could be reached in different situations:

1. Centroids positions and allocations do not change any more.
2. Error improvement is below a fixed threshold between two consecutive iterations.
3. A maximum number of iterations is reached.
4. Error reaches a target value.

By assuming a m-sized dataset in $\mathbb{R}^n$ divided into k clusters and t the number of iteration before reaching the convergence condition, the algorithm complexity is O($ktmn$).

Advantages of k-means approach are:

- It is fast
- It is relatively simple
- It always converges (usually very fast)

Drawbacks of k-means approach are:

- It does not guarantee an optimal solution
- The solution depends on the initial centroids
- k must be known a priori
- Forces spherical symmetry of clusters (in the n dimensional space)

2.3 Neural networks

2.3.1 Neural networks structure

A *neural network* (also called an artificial neural network) is an adaptive system that learns by using interconnected *nodes* or *neurons* in a layered structure that resembles a human brain. A neural network can learn from data, so it can be trained to recognize patterns, classify data, and forecast future events[28].

Neural networks were firstly proposed in 1940-50. Their first practical applications become in the 80-90s, but their practical results were very low. From 2010 on, deep architectures with impressive performances has been implemented.

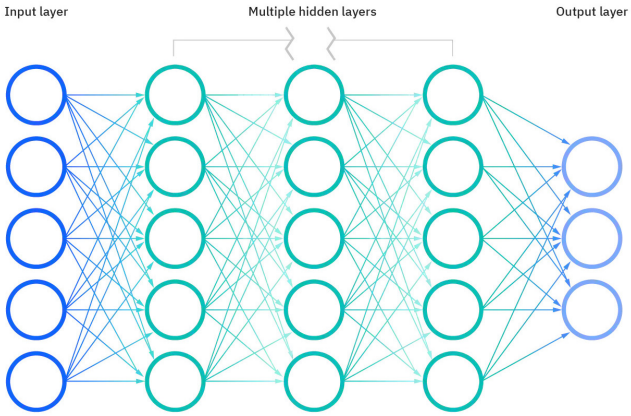


Figure 2.2: Graph of a Neural Network

Typical graph of a neural network is shown in figure 2.2. The Neural Network is constructed from three type of layers:

1. **Input layer:** Initial data for the neural network. This layer will accept the data and pass it to the rest of the network. Number of neurons corresponds to number of inputs.
2. **Hidden layers:** Intermediate layers between input and output layer and place where all the computation is done. Hidden layers are either one or more in number for a neural network. In the above case, the number is 3. Hidden layers are the ones that are actually responsible for the excellent performance and complexity of neural networks. They perform multiple functions at the same time.

3. **Output layer:** Holds the result or the output of the problem for given inputs. Raw images get passed to the input layer and we receive output in the output layer.

A neuron is the basic unit of a neural network. Neurons receive input from an external source or other nodes. Each node is connected with another node from the next layer, and each such connection (arrows in graph 2.2) has a particular weight. Weights are assigned to a neuron based on its relative importance against other inputs[29].

In a neural network, the flow of information through neurons can happen in two different ways[30]:

- *Feedforward Networks:* In this model, signals travel only in one direction, towards the output layer. Feedforward networks have an input layer and a single output layer with zero or multiple hidden layers. They are widely used in pattern recognition.

- *Feedback Networks:* In this model, the recurrent or interactive networks use their internal state (memory) to process the sequence of inputs. In them, signals can travel in both directions through the loops (hidden layer/s) in the network. They are typically used in time-series and sequential tasks.

2.3.2 Neurons activation functions

When all the node values from the input layer are multiplied (along with their weight) and summarized, it generates a value for the first hidden layer. Based on the summarized value, the first layer has a predefined *activation function* that determines whether or not this node will be activated and how active it will be. More precisely, activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it[31]. The purpose of the activation function is to introduce non-linearity into the output of a neuron. Thus, applying an activation function is needed to make the network dynamic and add the ability to it to extract complex and complicated information from data and represent non-linear convoluted random functional mappings between input and output[32].

There exist several types of activation functions that determine neurons activation. The most important activation functions are:

1. **Binary Step function**

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

While creating a binary classifier binary activation function are generally used, but binary step function cannot be used in case of multiclass classification in target carriable. Also, the gradient of the binary step function

is zero which may cause a hindrance in back propagation step. The main drawback of the binary step function is that it had zero gradient because there is no component of x.

2. **Linear function**

$$f(x) = \alpha x$$

The linear activation function is directly proportional to the input. The value of variable α can be any constant value chosen by the user. The gradient is not zero, but a constant value which is independent of the input value x. It can be noted that, if α is unitary, it is as no activation function is being implemented. There isn't much benefit of using linear function because the neural network would not improve the error due to the same value of gradient for every iteration. Furthermore, the network will not be able to identify complex patterns from the data.

3. **Sigmoid function**

$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid is the most widely used activation function as it is a non-linear function. Sigmoid function is continuously differentiable and it looks like a smooth S-shaped function. It is centered in $(0, 0.5)$ and it has an horizontal asymptote towards 0 for $x \to -\infty$, and an horizontal asymptote towards 1 for $x \to +\infty$. Furthermore, sigmoid function is not symmetric about zero, which means that the signs of all output values of neurons will be same. This issue can be improved by scaling the sigmoid function.

4. **Hyperbolic Tangent function**

$$f(x) = tanh(x) = 2sigmoid(2x) - 1$$

Hyperbolic tangent function ($tanh$) function shape is similar to the sigmoid function, but it is symmetric to around the origin. It is centered in the origin and it has an horizontal asymptote towards -1 for $x \to -\infty$, and an horizontal asymptote towards 1 for $x \to +\infty$. This results in different signs of outputs from previous layers which will be fed as input to the next layer. As compared to the sigmoid function, the gradient of $tanh$ function is more steep. Hyperbolic tangent is preferred over sigmoid function as it has gradients which are not restricted to vary in a certain direction and also, it is zero centered.

5. **ReLU function**

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Rectified Linear Unit (ReLU) function is a non-linear activation function which is widely used in neural networks. The upper hand of using ReLU function is that all the neurons are not activated at the same time. This implies that a neuron will be deactivated only when the output of linear transformation is zero. ReLU is more efficient than other functions because as all the neurons are not activated at the same time, rather a certain number of neurons are activated at a time.

6. **Parametric ReLU function**

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

Parametric ReLU is a variant of Rectified Linear Unit with better performance and a slight variation. It resolves the problem of gradient of ReLU becoming zero for negative values of x by introducing a new parameter of the negative part of the function. If α is set to a small value (as 0.01), it behaves as leaky ReLU function, but here a is also a trainable parameter. For faster and optimum convergence, the network learns the value of α.

7. **Softmax function**

$$\sigma(\boldsymbol{x})_j = \frac{e^{z_j}}{\sum_{\boldsymbol{k=1}}^{K} e^{z_k}}$$

Softmax function is a combination of multiple sigmoid functions. It takes as input a vector $\boldsymbol{z}_i$ of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. Since the sigmoid function returns values in the range $[0, 1]$, these can be treated as probabilities of a particular class data points. Softmax function unlike sigmoid functions, which are used for binary classification, can be used for multiclass classification problems.The function, for every data point of all the individual classes, returns the probability. When a network or model for multiple class classification is built, the output layer of the network will have the same number of neurons as the number of classes in the target. Usually, softmax function is applied at neurons of output layer.

It was seen that overfitting is one of the most issues during the train of a neural network. Since it cannot be completely eliminated, there are many methods to avoid, or significantly reduce, overfitting problem. These methods are described below.

1. **Dropout**
   At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections. Thus, each iteration has a different set of nodes and this results in a different set of outputs. This leads to avoid that the output depends too much on single neurons[33].

2. **Regularization**
   Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on the unseen data as well. A regularization term is added to the loss function with the aim to penalize big weights. Regularization parameter λ determines how much the regularization term affects the process. Regarding the λ parameter, if its value is too low, the model will be more complex and risk of overfitting situation might occur[34]. The model will learn too much about the particularities of the training data, and won't be able to generalize to new data. On the other hand, if lambda value is too high, the model will be simple, but the risk of underfitting situation might occur. The model won't learn enough about the training data to make useful predictions.

3. **Early stopping**
   Early stopping is a kind of cross-validation strategy in which a part of the training set is kept as the validation set. When performance on the validation set is getting worse, the training on the model will be immediately stopped. *Patience* parameter denotes the number of epochs with no further improvement after which the training will be stopped.

4. **Data augmentation**
   A simple way to reduce overfitting is to increase the size of the training data. Theoretically, it is not possible to increase the size of training data as the labeled data was too costly. New numerically calculated samples are added to the dataset in order to *virtually* increase the number of training samples. Let's consider, for example, the case in which it is dealt with images. There are a few ways of increasing the size of the training data by applying random transformations – rotating the image, flipping, scaling, shifting, etc. This usually provides a big leap in improving the accuracy of the model.

Chapter 3

Developed system

The following chapter describes the developed system in three different sections. First section shows how data have been acquired from vehicle networks and which softwares have been used for different tasks, depending on the needs. Second section describes the method implemented for the CAN signals clustering by exploiting machine learning clustering techniques and algorithms. The third part concern task of prediction of an ECU output data by emulating the ECU behavior via software. For this task will be implemented a classification neural network in order to understand the input/output data relationship.

3.1 Data acquisition

3.1.1 Networks analysis software

In order to acquire data from various vehicle networks, CANoe software has been used. CANoe is a software tool for development, test and analysis of individual ECUs and entire ECU networks. It supports network designers, development and test engineers throughout the entire development process – from planning to system-level test. Versatile variants and functions provide the appropriate project support. CANoe leads to have an interface with Ethernet, MOST, FlexRay, LIN and CAN networks. Therefore, its versatile functions and configuration options are used successfully by OEMs and suppliers worldwide[35].

In CANoe software, it is possible to customize the main screen. For the implemented task, it was set a configuration including *Measurement setup*, *Write*, *Trace* and *Simulation setup* windows. In *Write* window, are printed system errors, warnings and messages, so it is useful to recognise system and hardware connection issues.

As shown in figure 3.1, from the *Simulation setup* section it is possible to upload CAN and FlexRay databases. These databases are owned by vehicles original equipment manufacturer (OEM) and they are different for every electronic vehicle architecture. Each CAN line has its own database, took with

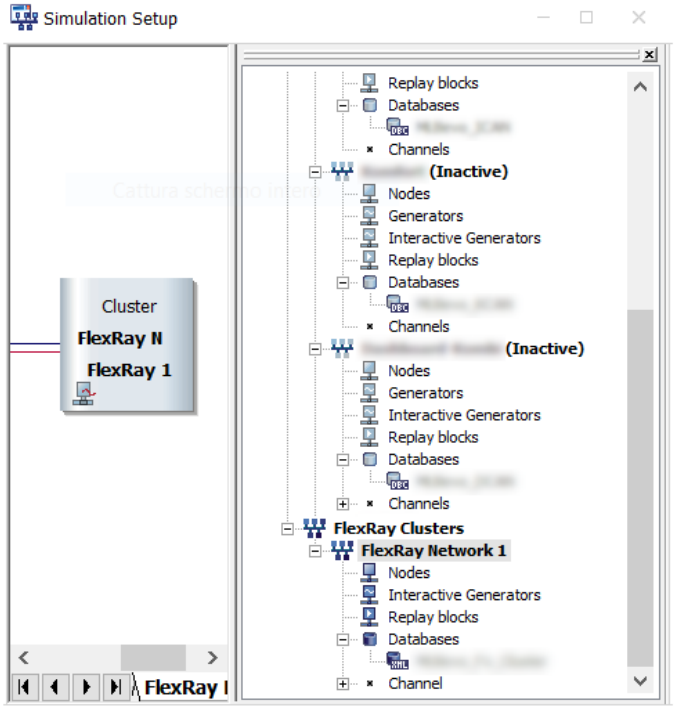


Figure 3.1: CANoe *Simulation setup* window

.dbc file extension. FlexRay has its own database, took with *.xml* file extension. *Simulation setup* section also leads to associate a specific database to a physical CAN or FlexRay channel line.

Figure 3.2, shows the *Measurement setup* window. Many blocks concerning measurement tasks appears in the right side of the window. Each block can be used by applying signals filters. Filters are useful to delete from measurements useless signals that could create confusion. The *CFB* filter concern CAN signals, while the *FFB* filter concern FlexRay signals. Each block links to another measurement window. *Graphics* block links to a window that leads to print signals graphs. It is useful in order to visualize signals shapes.

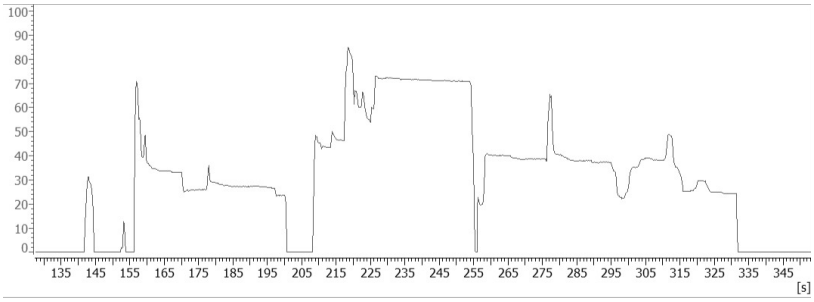


Figure 3.3: Graph of brake oil pressure signal

An example of a signal that was intentionally changed during an acquisition is the oil pressure of the brake pedal, whose graph is reported in figure 3.3. *Logging* block leads to insert a link of the folder in which save the acquisition.

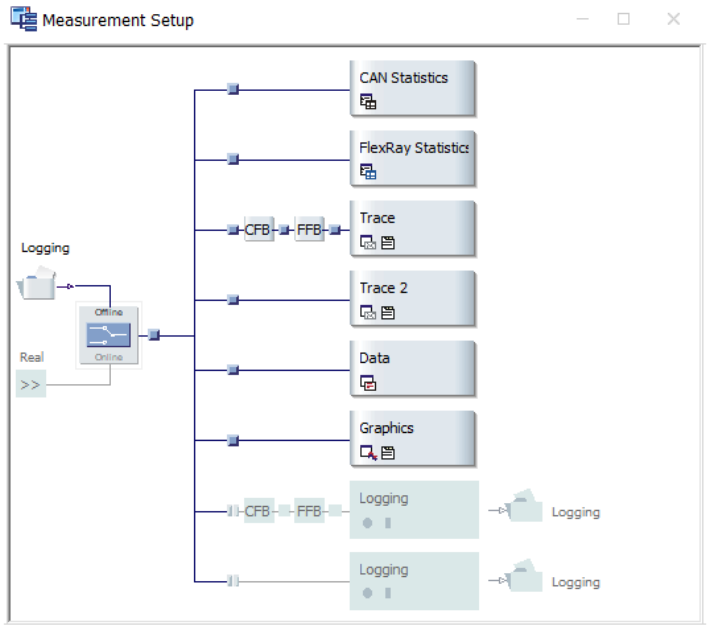


Figure 3.2: CANoe *Measurement setup* window

In figure 3.2, that shows the *Measurement setup* window, there are two different *Logging* blocks. In the first one were applied both CAN and FlexRay filters in order to acquire only signals of interest. The second one has no filters in order to acquire all signals possible. Then, it store them in case one of them want to be analyzed in the future for any reason. *Trace* block is fundamental because it links to a window in which are shown signal values, features and descriptions.

In *Trace* window, shown in figure 3.4, some areas have been blurred in order to comply corporate privacy. Numbers in the left side of the window, under *Time* label, represent the current time. *ID* label indicates identifiers of CAN messages and FlexRay PDUs. *Name* label indicates the message or PDU name. Under *Data* label are reported entire messages/PDUs data, grouped in bytes. Blurred areas next to symbols similar to a blue tilde, show all selected signal names (left part), signal values (central part) and signal descriptions (right part). In case no CAN or FlexRay database has been added to the *System setup* section, only IDs and start bit values are shown instead of message and signal names.

3.1.2 Vehicle hardware interface

Physical connection between vehicle networks and CANoe software occurs exploiting a Vector VN7640 device.

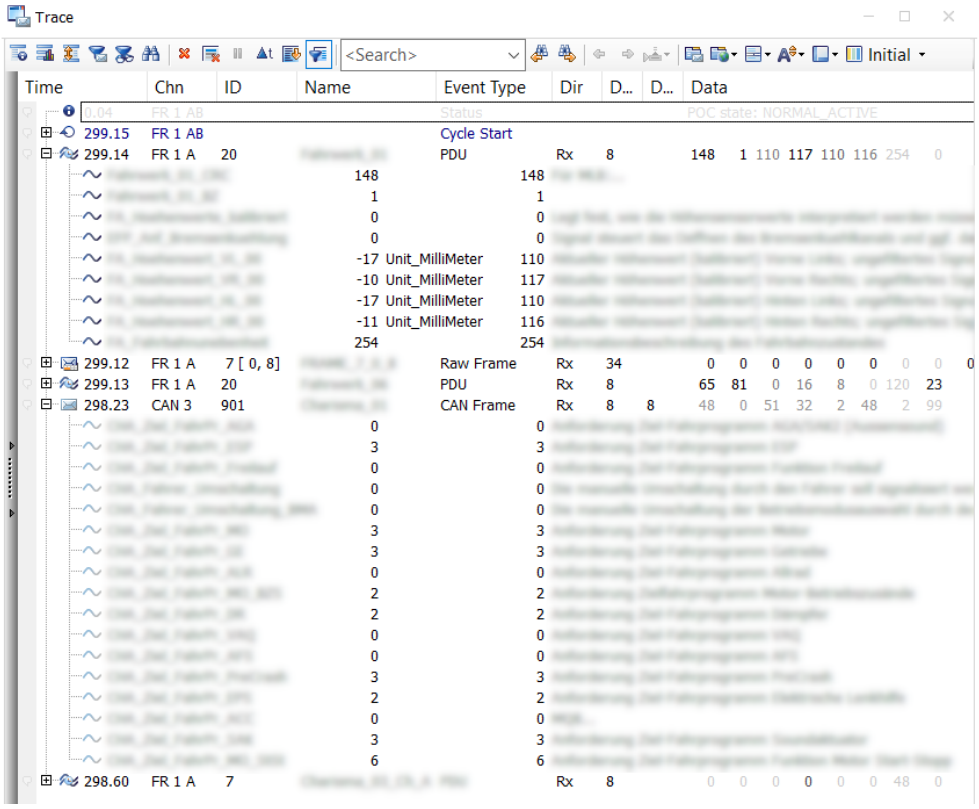


Figure 3.4: CANoe *Trace* window

Figure 3.5: Vector VN7640 device

Vector VN7640 is shown in figure 3.5. It is a network interface device with USB interface for CAN FD, LIN, FlexRay, K-Line, J1708 and Ethernet. It is well suited for this task because it interfaces vehicle networks with CANoe software. It has four input channels. Three channels are dedicated to CAN lines and one is dedicated to FlexRay bus. The amount of four input channels leads to acquire a high quantity of different signals, depending on the needs.

3.1.3 Data processing software

As data processing software, it will be used Matlab by MathWorks. The choice of Matlab was done because of the easily interface with CANoe software. From CANoe it is possible to convert logging files from *.blf* to *.mat* format, both file types readable by Matlab. Matlab is also well suited to processing vehicle networks data because of its *Vehicle Network* and *Statistics and Machine Learning* toolboxes, both exploited in this project. *Vehicle Network Toolbox* contains many useful instruction for interface Matlab to CANoe that have been exploited. One of these, is *blfread* instruction that leads to upload a *.blf* format logging file to Matlab and associate a *.dbc* format CAN database to it. Matlab was chosen also for machine learning applications because of the simplicity to find documentation on the web.

3.2 CAN signals clustering

3.2.1 System description

As told in section 3.1.1, with CANoe software it is possible to read data from vehicle networks. A CANoe user could know signals meaning by uploading appropriate CAN and FlexRay databases. These databases associate each message ID (in case of CAN protocol) or each PDU ID (in case of FlexRay protocol) and each signal belonging to those IDs with a specific signal name and description. Having these database files is fundamental in order to understand what kind of signals are exchanging ECUs through network buses. Unfortunately, database files are owned by vehicles OEMs and they are different for every electronic vehicle architecture, so not always it is possible to have them.

The aim of the developed system is to read data from vehicle network and to detect their function without associating them a database. This task might be useful in order to understand fault error messages in case OEMs do not make databases available. Another advantage could be to understand a specific vehicle architecture by doing reverse engineering. Since different signals in network protocols are in the hundreds, it is unlikely to assign a specific function to each read signal. It is more probable to assign them to a macro category depending on what function they belongs. For example, understand if signals belongs to engine functions, lighting system, and so on.

3.2.2 Signals choice

In order to group messages by their function, it is useful to find common and similar features on different network messages. These similar features might be found on frame data structure. Since different protocols have different frame

structure, for simplicity only one network protocol will be taken into account. Since CAN protocol is the most diffused protocol in the automotive field, only signals belonging to this protocol will be taken into account.

In order to find similarities on CAN data it is fundamental to understand what kind of features are available on read data from CAN buses without associating database files. Exploiting the *blfread* function in Matlab, a generic uploaded logging file presents following features that are available from the user:

- *Time*: Time instant of current sample.
- *ID*: CAN identifier of current sample.
- *Extended*: Tells if current sample has an extended ID or not.
- *Name*: Current sample message name associated at previous ID.
- *Length*: Length of current message expressed in number of bytes. It varies from 1 to 8.
- *Data*: Array containing current sample data. It contains a number of bytes given from *Length* field.
- *Signals*: Signal names belonging to the current message and their corresponding values.

Without associate a CAN database to the logged file, only *Time*, *ID*, *Extended* and *Length* fields are available. In this application, extended ID messages corresponds to Bedien und Anzeigeprotokoll (BAP) signals, that is a transport protocol developed and used only by Volkswagen Group, used for exchanging information between different devices on the car network[36]. Thus, a first discard might be done for message whose ID is extended because it cannot be generalized to other OEM vehicles.

A second discard might be done for signals that won't vary their value during an acquisition. Some signals may not be implemented in the network because they are under development and they will be available in a second release, or because they are obsolete and they won't be used anymore. As data set for the clustering task, it will be taken into account only signals that, during an entire acquisition, will change their value at least once.

Data Byte 1	0	1	0	1	1	0	1	1
Data Byte 2	1	0	1	0	1	0	0	0
Data Byte 3	1	0	0	0	1	0	1	1
Data Byte 4	0	1	0	1	0	0	0	0
Data Byte 5	0	1	0	0	0	1	1	1
Data Byte 6	1	0	0	1	0	1	0	0
Data Byte 7	0	1	0	0	1	0	1	0
Data Byte 8	1	0	1	0	1	0	0	1

(a) Data field without database

Data Byte 1	0	1	0	1	1	0	1	1
Data Byte 2	1	0	1	0	1	0	0	0
Data Byte 3	1	0	0	0	1	0	1	1
Data Byte 4	0	1	0	1	0	0	0	0
Data Byte 5	0	1	0	0	0	1	1	1
Data Byte 6	1	0	0	1	0	1	0	0
Data Byte 7	0	1	0	0	1	0	1	0
Data Byte 8	1	0	1	0	1	0	0	1

(b) Data field with database

Figure 3.6: CAN data field structure example

If database files were be used, the CAN frame data field could be divided into different signals, as shown in the example in figure 3.6b. That information might be very valuable, but since database files are not added for initial specifications, data field will appear as shown in the example in figure 3.6a. Ideally, would be an excellent solution consider features concerning different signal lengths or start bits[1], but due to the fact that signal division is not known, the only way to emulate a signal division is to divide data for their bytes. It could be a good compromise in order to find similarities between different messages. Another feature by which grouping similar data might be frame IDs. From now on, these ID's bytes taken into account will be called **signal bytes**.

3.2.3 Signals acquisition

As described before, not all signal bytes are implemented in a specific vehicle network. To optimize the choice of signal bytes taken into account, the approach used is to acquire the highest possible number of byte signals that change during an acquisition. The fact that they vary implies that they are currently implemented in the network and they are active signals. In order to comply the previous consideration, an acquisition by changing the highest possible number of driver's interface commands inside the vehicle is done. In particular, signals coming from four[2] different CAN lines are acquired and merged together. These four CAN lines were chosen by considering CAN lines that carry messages with different functions because they are the most representative of signals that have different functions. For simplicity, during the acquisition only commands at driver's interface that are easily accessible by the driver without moving the vehicle have been made to vary. For an example, have been made to vary commands such as internal and external lights, windscreen wiper, radio volume, windows, air conditioner, brake pedal, doors and so on. The aim is to vary the highest number possible of signals - both similar signals in order to find similarities and different signals in order to detect different clusters associated to different vehicle functions. Thus, no filter has been applied to the logging file during the acquisition.

Once uploaded the logging file in Matlab, all signal bytes belonging to the logging file were scanned. All signal bytes whose values don't change during the entire acquisition were discharged. Remaining values were merged in a unique database whose entries are *IDs* and *signal bytes*.

A visual representation of the database is given in figure 3.7. It is recalled that, since IDs are made up from 11 bits and data fields are made up from 8 bytes, sample ID values goes from 1 to $2^{11} = 2048$, while sample signal byte values goes from 1 to 8.

[1]Start bit is the physical position of the first bit of a signal in the data field of its message. Start bit could vary from 1 to 64 since there are 8 bytes

[2]Number of CAN lines by which signals have been acquired are limited by the number of Vector VN7640 device inputs.

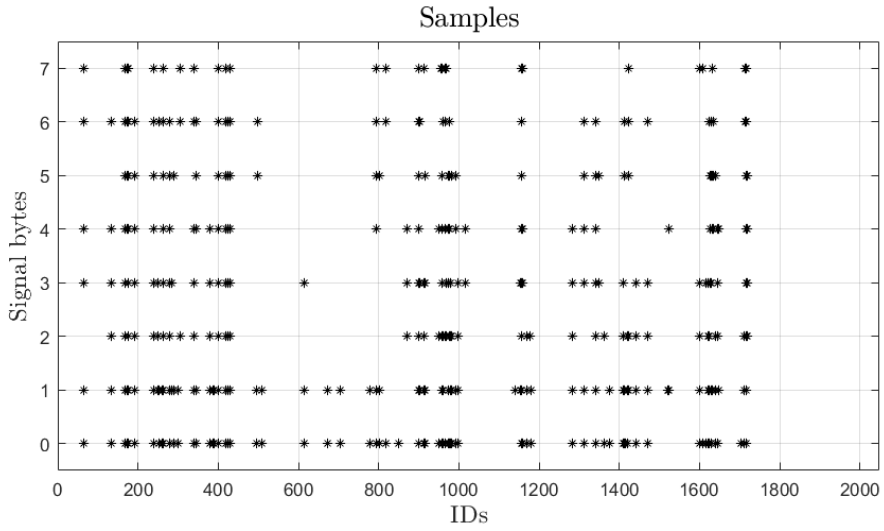


Figure 3.7: Database with IDs and signal bytes entries

3.2.4 Signals clustering

In order to group samples by their similarity, k-means algorithm described in chapter 2.2.2 was implemented. k-means algorithm, requires a parameter k that represents the number of clusters. The number of clusters has been decided by taking into account database files. By analyzing them, it is possible to detect five different macro categories of signal typologies. In the light of these signal typologies, it has been set the parameter $k = 5$, i.e., five clusters will be created by k-means algorithm. These five signals typologies are the following:

1. HV batteries, DC-DC converter and power electronics systems
2. Suspensions and vehicle dynamic controls
3. Dashboard, displays and driver's vehicle interface
4. Driver's comfort and lighting system
5. Powertrain and engine controls

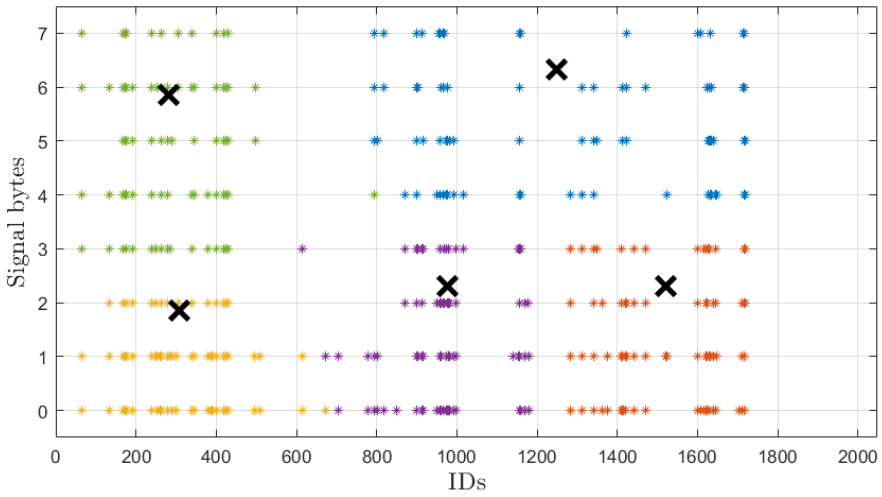
kmeans Matlab function has been exploited to create clusters. Before create clusters, all signal features were normalized to one. This leads to have a *squared* dataset, in which IDs and signal bytes have same relevance.

Centroid positions and number of samples belonging to each cluster are reported in table 3.1.

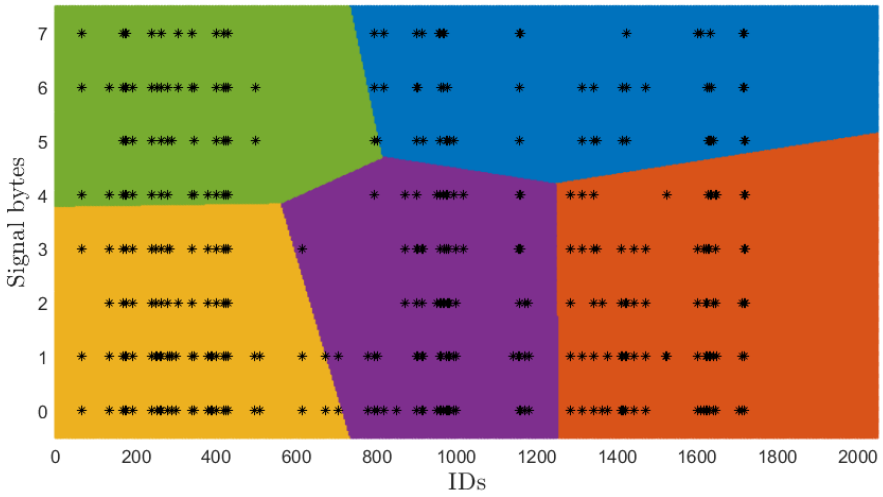
Figure 3.8 gives an overview on how samples have been grouped in $k = 5$ clusters. In figure 3.8a are highlighted clusters centroids, wile in figure 3.8b is highlighted clusters division in different areas.

Table 3.1: Cluster centroids features

Cluster's number	Centroid coordinates	Samples number
1	1247.6, 6.3	87
2	1521.2, 2.3	79
3	308.2, 1.8	79
4	976.3, 2.3	86
5	282.2, 5.8	86



(a) Clusters centroids visualization



(b) Clusters areas visualization

Figure 3.8: Clustered samples

3.2.5 Clusters validation

Validation of obtained results was made by creating a validation data set and assign each sample to clusters given by k-means algorithm. For the validation set were chosen a set of known signals by examining database files. The choice of signals belonging to the validation set was done by considering five sets of signals with similar features. In particular, sixty six signals were taken into account. For example, following signals belonging to the lighting system have been chosen:

- Request for permanent switching on of the main beam headlights
- Left turn side light
- LED activation on the charging button
- The ABS safety warning lights have been turned on
- Reverse light indicator
- The left rear light defective
- Interior lights activation (footwell, doors, etc.)

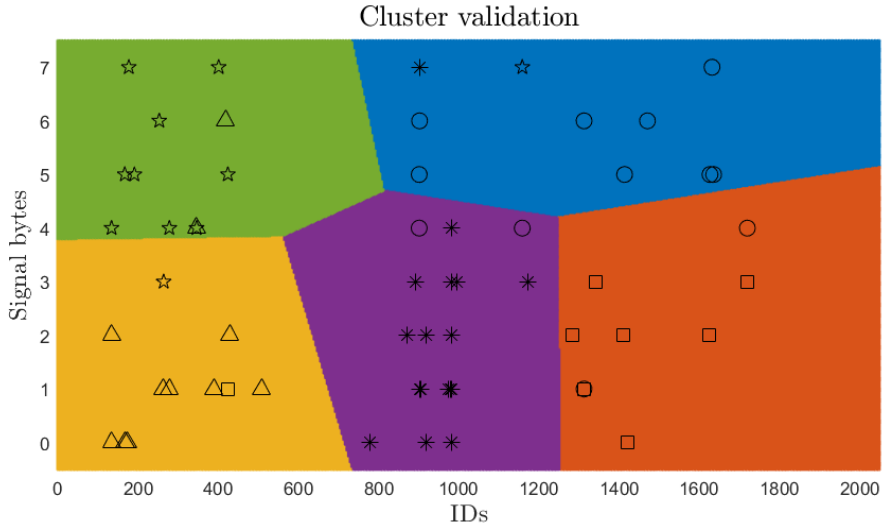


Figure 3.9: Clustered validation set

Figure 3.9 shows the validation set clusterization. In the figure, each signal function has been highlighted with a different sample shape. Most of validation samples with the same shape have been grouped in same clusters. More precisely, sixty samples over sixty six were grouped in same clusters and only six sample belongs to another cluster.

Since the dataset is unlabeled, it is not possible to assure that the all the samples grouped together actually belong to the correct clusters. Thus, a new definition of *correctness* will be established:

It is possible to assert that if almost all samples of a labeled sample data set, each one with the same label, are grouped in the same cluster, they will be clustered in a correct way with a very high probability.

By considering this last definition of correctness, since almost all of similar samples have been grouped together, it is highly probable that they belong to correct clusters. As a percentage, 90.9% samples belong to the correct cluster, while 9.1% belong to a wrong cluster.

3.3 ECU behavior simulation

3.3.1 System description

The goal of the project is to understand the input/output signals relationship of an ECU in order to emulate its behavior. Then, a Matlab model that behaves like the ECU will be implemented and will be tested. The main purpose of the project is to bypass an ECU in case of its fault. By knowing an ECU behavior it is possible to substitute it with a device that hypothetically behaves like it. Another purpose is to understand ECU behaviors and electrical architecture of others vehicle OEMs. The technique adopted in this system is the so called *reverse engineering*, that's the process of studying a product to understand how it is made.

Choice of the ECU was made by taking into account signals that are easily accessible by the user by changing driver's commands. One of most representative ECU that comply what has been said, is the ECU related to driving mode features. Database files were exploited in order to understand which are input and output signals functions. Since the driving mode ECU has lots of input and output signals, to simplify the purpose, only one function representing the task was taken into account. This chosen signal represents a set point to be reached by the driving mode selector. As inputs, only accountable signals for the chosen output signal were considered. They are six signals that could be changed by varying following driver's commands:

1. Soil typology:
    - On-road
    - Sport
    - Race
    - Off-road
    - Sand
    - Snow
2. Vehicle dynamic features:
    - Four wheel drive activation

- Steering wheel maneuverability
- Suspension stiffness

In order to have the widest variety of input and output combinations, a logging by varying all possible above reported user's commands was made. It is not superfluous to specify that input signals are exchanged by exploiting the CAN bus, while the output signal belong to the FlexRay network. Since CAN is an event-triggered protocol and FlexRay is a time-triggered protocol, a time match is needed to match input signals with the corresponding output.

It might make little sense to set up a data set with all input signals time steps because of the repetition of input values combinations during contiguous time steps. As data set samples, it will be taken a numeric array representing the value of all input signals every time one of them changes its value. Match between input and output signals is done by taking the time steps of all data set samples. It will be associate to them, as a label, the value of the output signal corresponding to the first available time step after 50 milliseconds from the input one. It was done in order to let the ECU elaborate input data and to set its output after its own latency time. By approaching in this way, the data set is made up by 123 samples.

3.3.2 Neural network implementation

By analyzing data set values, it is possible to note that both input and output signals values are integer number. In particular, output values, corresponding to the codomain, assume three possible different values. In the light of this, a *classification* neural network will be implemented in order to detect the input/output relationship.

Dataset, which consists on 123 samples, will be randomly split into training, test and validation sets with percentages of 75%, 15% and 10% respectively. To find the best neural network model, a set of hyperparameters will be changed to evaluate the model with the lowest training error possible. In particular, following hyperparameters will be taken into account.

Hidden layers number: [10], [20], [10 10], [10 10 10]

Regularization term λ*:* 0, 10^{-6}, 10^{-4}, 10^{-3}

Activation function: ReLU, Hyperbolic tangent, Sigmoid

Hidden layers choice was made in order to let the neural network do the training both over a single layer with much neurons and much layers with a lowest number of neurons. Regarding the regularization term, it is know that if λ value is too low, the model will be more complex and risk of overfitting situation might occur. On the other hand, if lambda value is too high, risk of underfitting situation might occur. For each combination of above hyperparameters, a different activation function of hidden layers neurons will be tried.

Table 3.2: Training errors with $ReLU$ activation function

Hidden layers VS λ	0	10^{-6}	10^{-4}	10^{-3}
[10]	15.16	12.89	13.54	13.88
[20]	12.54	15.05	14.16	13.82
$[10-10]$	14.22	15.62	15.99	12.86
$[10-10-10]$	12.98	13.81	12.97	13.42

Table 3.3: Training errors with *Sigmoid* activation function

Hidden layers VS λ	0	10^{-6}	10^{-4}	10^{-3}
[10]	14.69	12.70	14.49	15.58
[20]	13.78	14.22	14.38	12.81
$[10-10]$	15.12	12.73	14.20	14.50
$[10-10-10]$	15.36	16.13	14.51	14.70

Table 3.4: Training errors with *Hyperbolic Tangent* activation function

Hidden layers VS λ	0	10^{-6}	10^{-4}	10^{-3}
[10]	13.99	14.38	13.46	13.83
[20]	14.66	15.73	14.76	14.52
$[10-10]$	14.71	14.79	13.43	13.38
$[10-10-10]$	14.96	13.51	13.40	14.24

Table 3.2, table 3.3 and table 3.4 report training errors with ReLU, sigmoid and hyperbolic tangent activation functions respectively. These training errors have been computed by the varying of above reported hyperparameters. Each error has been computed as the mean of 60 training for its hyperparameters combination. In order to have more variety of training and validation samples, for each training, dataset will be divided into training, validation and test sets again. This leads to have a more precise and trusted value for the error.

It is possible to see that the lowest training error over the entire trainings, occurs with hyperparameters combination of:

Activation function: ReLU; *Hidden layers:* [20]; *Regularization term* $\lambda = 0$

As described in chapter 2.3.2, Softmax activation function has been applied at neurons of the output layer.

3.3.3 Model validation

In order to test the model, a second logging on vehicle ECU's input and output signal is done. The previously learned model is used to predict the second logging output. Then, effective output values from the logging file are compared with values predicted by the model.

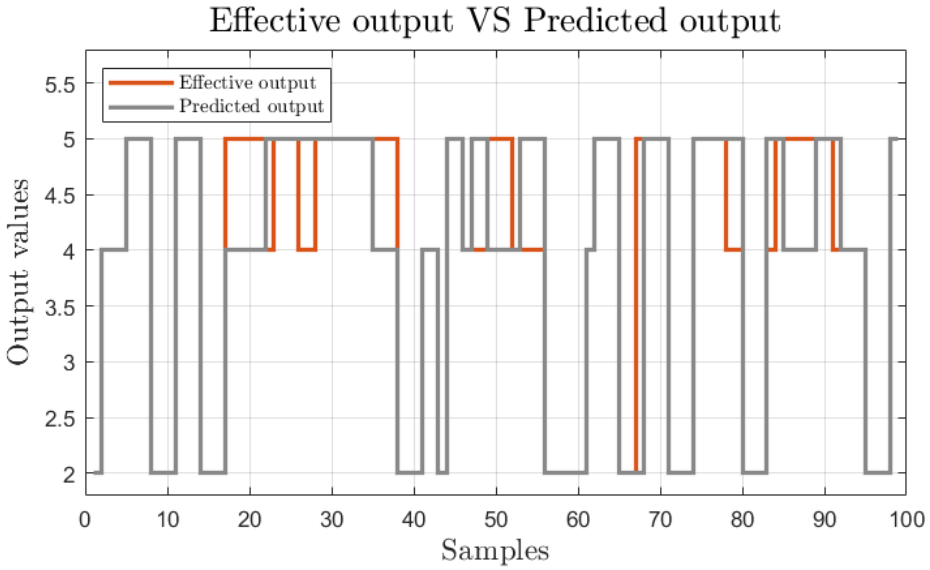


Figure 3.10: Neural network predicted VS real values

Figure 3.10 shows the neural network predicted output values (in gray) compared to the effective output values (in red). It is possible to observe that the two signals are almost equals. More precisely, they differ to the 13%. In other words, the match between the predicted output signal and the real output signal is 87%. It is possible to ascertain that the implemented neural network model works with a good prediction match.

Conclusions

The CAN signal clustering task, described in chapter 3.2, gives excellent results, however, it could be improved. At first, more significant signals might be considered. Since all changing signals during the acquisition have been considered, a more accurate way to choose significant signals is to take into account only signals that change when the driver carry out an action doing a time match between actions and signals. In this way some useless signals are discharged, for example signals concerning CRC (Cyclic Redundancy Check) transmissions. By analyzing data in a more accurate manner, a highest number of clusters could be implemented by detecting more vehicle functions groups. Another improvement might be to change data features weights. By giving a different weights to IDs and/or to signal bytes, a different *shaped* clusters could be implemented. Instead of circular clusters, elliptical clusters will occurs. An upgrade of the system could be the extension of the model to FlexRay and to other vehicle network protocols. Considering the FlexRay network, for example, IDs and Payload field data might be used in order to cluster signals.

Regarding the second task, ECU's behavior simulation, it has been observed that the match between the predicted output signal and the real output signal is 87%. Since it is a good result, it could be improved in various ways. One way is to acquire more and different data in order to have a larger dataset. The dataset size corresponds to 123 samples and for a neural network it could be very small amount. Risk of this under-sized dataset is to fall in underfitting problem. A way to confirm the underfitting problem occurs by observing that, in tables 3.2, 3.3 and 3.4, the training error doesn't vary too much between different hyperparameters. Another improvement might be to take into account some ECU's output signals as model input signals. It might be also considered all input ECU's signals as model input signals in order to detect if they are involved in output behavior. By considering a larger dataset and by taking into account more input signals, test error may highly decrease at the expense of neural network complexity and training times.

Since the model predicts the driving mode ECU behavior, a wrong data prediction does not involve safety critical situations and it might be associated to an Automotive Safety Integrity Level (ASIL) B. Possible consequences of a wrong prediction could be to select an inappropriate driving mode while driving on a critical soil. An example may be selecting a *on-road* mode while driving on an icy soil. In general, selected driving mode is shown in the dashboard, so, it might be checked by the driver the correctness of ECU prediction.

Potential applications concerning the developed project are, in primis, to fix up possible vehicles electronics faults. By understanding vehicle functions by exploiting the developed model might prove useful in case of absence of database files, or in case database files are not updated or they are incomplete. Despite it cannot be understood in a very precise manner a vehicle function, it is possible to stem the fault to a restricted faults family. In addiction, vehicle predictive maintenance could be done. Another application task could be to check by OEMs if T1 (Tier One) manufacturers comply the required specifications.

List of Figures

List of Tables

Bibliography

[1] *IEEE Draft Guide: Adoption of the Project Management Institute (PMI) Standard: A Guide to the Project Management Body of Knowledge (PMBOK Guide)-2008 (4th edition)*, in *IEEE P1490/D1, May 2011 , vol., no., pp.1-505, 30 June 2011*, doi: 10.1109/IEEESTD.2011.5937011

[2] *Maturity level assurance for new parts, Methods, measurement criteria, documentation.* German Association of the Automotive Industry Quality Management Center. 3^{rd} revised edition, October 2021.

[3] Kiencke, U., Dais, S., and Litschel, M., *Automotive Serial Controller Area Network*, SAE Technical Paper 860391, 1986

[4] https://dictionary.cambridge.org/it/dizionario/inglese/dependability

[5] Society of Automotive Engineers. J2056/1 class C application requirements classifications. In *SAE Handbook*, 1994

[6] https://www.can-cia.org/can-knowledge/can/classical-can/

[7] https://elearning.vector.com/mod/page/view.php?id=378

[8] https://www.ni.com/it-it/innovations/white-papers/06/flexray-automotive-communication-bus-overview.html

[9] Nicolas Navet, Françoise Simonot-Lion. Trends in Automotive Communication Systems. Richard Zurawski. *Embedded Systems Handbook: Networked Embedded Systems - 2nd ed.*, Taylor and Francis / CRC Press, pp.13.1-13.24, 2009, Industrial Information Technology Series, ISBN 978-1-4398-0761-3.

[10] Andreas Grzemba, *MOST - The Automotive Multimedia Network*, from MOST25 to MOST150, Franzis

[11] S. Lee, B. Cho, Y. Choi and K. Baek, *Implementation of MOST/-CAN network protocol*, 2011 International Conference on Electrical and Control Engineering, 2011, pp. 5974-5977, doi: 10.1109/ICECENG.2011.6057339.

[12] https://www.vector.com/int/en/know-how/most/#c21358

[13] https://www.oracle.com/data-science/machine-learning/what-is-machine-learning/

[14] https://www.expert.ai/blog/machine-learning-definition/

[15] https://oden.io/glossary/model-training/

[16] https://oden.io/glossary/model-training/

[17] https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[18] https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690

[19] https://machinelearningmastery.com/difference-test-validation-datasets/

[20] https://www.analyticsvidhya.com/blog/2021/09/a-complete-guide-to-understand-classification-in-machine-learning/

[21] https://www.javatpoint.com/unsupervised-machine-learning

[22] https://www.guru99.com/unsupervised-machine-learning.html

[23] https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/

[24] https://developers.google.com/machine-learning/clustering/clustering-algorithms

[25] https://www.displayr.com/what-is-hierarchical-clustering/

[26] https://www.kdnuggets.com/2019/09/hierarchical-clustering.html

[27] https://scikit-learn.org/stable/modules/clustering.html

[28] https://www.mathworks.com/discovery/neural-network.html

[29] https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/

[30] https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/

[31] https://www.geeksforgeeks.org/activation-functions-neural-networks/

[32] *Activation Functions in Neural Networks*, in *International Journal of Engineering Applied Sciences and Technology*, 2020. Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316

[33] https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[34] https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/lambda

[35] https://www.vector.com/int/en/products/products-a-z/software/canoe/#

[36] https://www.protokollix.de/automotive.html