

University of Miskolc
Faculty of Mechanical Engineering and Informatics
Department of Information Technology



No code machine learning platform

Master thesis

Author: **Youcef Guichi**
Registration code: **DXC2E1**

MSc degree in Computer Science Engineering
2022

University of Miskolc
Faculty of Mechanical
Engineering and Informatics



Department of
Information Technology
Miskolc-Egyetemváros, 3515
Hungary

Major: Computer Science and
Engineering
Level: Master

Thesis number:
GEIAL/ DXC2E1/MSc/2022

National Uni. Id: FI 87515

THE S I S S P E C I F I C A T I O N

Youcef Guichi

Registration code: DXC2E1
Candidate for MSc degree in Information Engineering

Subject of thesis: **Machine Learning**

Title of thesis: **no code machine learning platform**

Detailed specification:

- 1. Give an overview of time series forecasting methods, techniques and models.*
- 2. Implement some machine learning algorithms for time series analysis and prediction.*
- 3. Develop a web-based platform, where users can*
 - upload their data files,*
 - choose an algorithm and set the parameters,*
 - run the analysis, and*
 - view the results.*

Supervisor:
Erika Baksáné Varga, PhD

Affiliation, position:
University of Miskolc, associate professor

Date of issue: **01-Feb-2022**

Deadline for submission: **29-Apr-2022**

Miskolc, _____

Prof. Dr. László Kovács
Head of Department

1. *Place of internship:* _____
2. *Supervisor of internship:* _____
3. *Modifications to the thesis:* *required (should be attached separately)*
not required (underline as appropriate)

Miskolc, _____ *Supervisor*

4. *Dates of consultation:*
 (1) _____
 (2) _____
 (3) _____
 (4) _____

Date
Supervisor/Consultant

5. *Thesis submission:* *accepted / not accepted (underline as appropriate)*

Miskolc, _____

Consultant
Supervisor

6. *Thesis contains:*

..... pages,
..... appendices,
..... CD attachment,
..... other attachment.

7. *Thesis:* *approved / not approved (underline as appropriate)*

Name and affiliation of Opponent: _____

Miskolc, _____

Head of Department

8. *Grades:* *Opponent:* _____

Department: _____

Final Examination Board: _____

Miskolc, _____

Chairman of FEB

Declaration of Authorship

I, ***Youcef Guichi***, Neptune code: ***DXC2EI***, MSc student of the Faculty of Mechanical Engineering and Informatics, University of Miskolc, being acutely aware of my legal liability, hereby confirm, declare and certify with my signature, that the assignment, entitled:

No code machine learning platform

except where indicated by referencing, is my own work, is not copied from any other person's work, and is not previously submitted for assessment at University of Miskolc or elsewhere, and all sources (both the electronic and printed literature, or any kind) referred to in it, have been used in accordance with the rules of copyright.

I understand that a thesis work may be considered to be plagiarized if it consists of

- Quoting word by word or referring to literature with either no quotation marks or no proper citation.
- Referring to content without indicating the source of reference.
- Representing previously published ideas as one's own.

I, hereby declare that I have been informed of the term of plagiarism, and I understand that in the case of plagiarism my thesis work is rejected.

Miskolc, _____ (Day) _____ (Month) _____ (Year)

Student's signature

Table of content

<i>Chapter 1: Introduction</i>	8
1.1 Introduction	8
1.2 Motivation	9
1.3 Objectives	9
<i>Chapter 2: Literature Review and Technological Background</i>	10
2.1 Time series	10
2.2 Stationary and nonstationary time series	11
2.3 Time-series components	11
2.3.1 Seasonality	11
2.3.2 Irregular	12
2.3.3 Trend	12
2.4 Machine learning	13
2.4.1. Naive Bayes classifier	13
2.4.2 Principal Component Analysis	14
2.4.3. Decision Tree	15
2.4.4 K-means	16
2.5 Deep learning	17
2.5.1: Going so deep	18
2.5.2 RNN	19
2.5.3 LSTM	21
2.5.3 GRU	22
<i>Chapter 3: Design and Implementation</i>	24
3.1 Tune-Quick Design and Conception	24
3.2 Tune-Quick used technologies and methods	24
3.2.1 Front-end	24
HTML	24
CSS	25
Bootstrap	25
3.2.2 Back-end	25
Python	25
Flask	26

<i>Keras</i>	26
<i>Asyncio</i>	27
<i>3.3 Tune-Quick Parser</i>	27
<i>3.4 Queue</i>	31
<i>3.5 Tune-Quick Models</i>	31
<i>3.5.1 Model class</i>	32
<i>3.5.2 GRUModel class</i>	36
<i>3.5.3 LSTMModel class</i>	37
<i>3.6 Tune-Quick app module</i>	37
<i>3.7 Tune-Quick Case Study</i>	38
<i>3.7.1 Dataset</i>	38
<i>3.7.2 User Interface</i>	39
<i>3.7.3 Results</i>	40
<i>4. Summary</i>	43

List of figures

Figure 1: Example of Cardano prices within a period of time	10
Figure 2: Stationary and non-stationary time series example.....	11
Figure 3: Example of seasonality	12
Figure 4: Example of a trend	12
Figure 5: Naive base classifier illustration	14
Figure 6: PCA illustration	15
Figure 7: Decision Tree illustration.....	16
Figure 8: K-means illustration	17
Figure 9: Artificial neural network architecture	18
Figure 10: Recurrent Neural Network Architecture	20
Figure 11: RNN forward pass equations	20
Figure 12: Distinction between RNN and ANN	20
Figure 13: LSM Block Equations	21
Figure 14: LSTM Block Architecture	22
Figure 15: GRU Block equations	23
Figure 16: GRU Block Architecture	23
Figure 17: Tune-Quick	24
Figure 18: The algorithms supported by Tune-Quick	28
Figure 19: Run_model coroutine.....	29
Figure 21: models class diagram	31
Figure 22: Clean_and_prepare_dataset fuction	32
Figure 23: Split_data function.....	33
Figure 24: Predict function	34
Figure 25: Plot_predictions function.....	35
Figure 28: LSTMModel class.....	37
Figure 29: App module	38
Figure 30: Dataset sample.....	39
Figure 31: Tune-Quick Interface.....	39
Figure 32: Inputs example.....	40
Figure 33: Tune-Quick backend logger example	41
Figure 34: The predicted chart.....	42

Chapter 1: Introduction

1.1 Introduction

In this century, data has become the definition of practically everything; the more data you have, the more understanding and control you have, the current world is in a variety of stages; some countries have a large amount of data, while others have a medium amount, and still, others have none.

Moreover, it turns out that different majors have different scales in terms of the amount of data available; for example, image classification has a lot more data, but object detection has a lot less.

Also, data science is a big field with a lot of concepts, approaches, methodologies, and algorithms, and it is a discipline where we analyze all kinds of data and strive to make it useful.

The focus of this thesis will be on how to ***assist users to use machine learning without having to code***. The thesis will address classical machine learning and time series analysis, and then the Tune-Quick platform will be introduced in chapter 3.

Machine learning platforms that provide no-code capabilities enable companies and users to use machine learning without advanced domain knowledge or training. These tools enable those with no formal expertise in software development to create machine learning applications.

It is very helpful for startups and small businesses to try out the taste of machine learning, it is also helpful if your budget is not enough to hire a complete team of data scientists, or you want from your team to focus on complex problems.

Users can utilize machine learning to make data predictions without writing any code, As shown in the figure below a comparison between traditional ML steps and no-code ml steps.

Tradition Machine learning development steps



No code Machine learning developments steps



1.2 Motivation

Tuning machine learning algorithms involves a lot of repetition in the code, from defining a set of layers to determining which parameters are appropriate for obtaining accurate results. What if we could assist researchers by creating a platform that allows them to try a variety of algorithms and parameters without having to code?

1.3 Objectives

The goal of this thesis is to develop a web-based platform that allows researchers to deliver end-to-end machine learning solutions without the need for coding, allowing them to focus more on the trials and testing of the solutions, making the process faster and more efficient. They will also save time by not having to deal with coding errors.

The main objectives are the following:

- Allow a wide spectrum of people to experiment the capabilities of machine learning algorithms.
- Create a user interface that makes the process simple.
- For businesses, it saves time and money.
- Very useful in fundamental machine learning challenges, allowing you to focus on more difficult problems.

Chapter 2: Literature Review and Technological Background

This chapter will go over the algorithms and principles that the Tune-Quick platform will automate later.

2.1 Time series

Time series analysis is a method for examining the performance of a variable or a combination of variables over a period of time.



Figure 1: Example of Cardano prices within a period of time [1]

As can be seen in the figure above, the x -axis represents time and the y -axis represent the price of ADA token per time unit, generally in time series applied to stock market or crypto market, the goal is to track the price of an asset during a time period, and thus help in understand the pattern of a particular asset and then we try to perform predictions, which will helps us to make better decisions for which asset we should invest.

2.2 Stationary and nonstationary time series

Time series have mainly two types, where the time series can be stationary, that means that the data have constant mean, variance and covariance, where nonstationary time series have a volatile mean, variance and covariance as shown in the figure below:

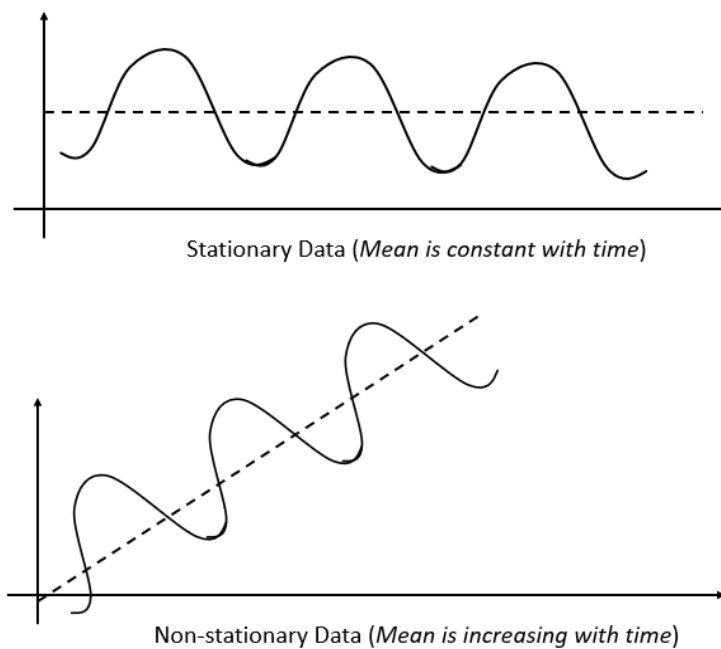


Figure 2: Stationary and non-stationary time series example[2]

2.3 Time-series components

Time series have 3 core components: seasonal, irregular, and trend; each component is suitable for a particular event.

2.3.1 Seasonality

Seasonality is derived from the word season, which means that something happens in a regular and predictive way, for example, the number of travelers increases in summer and decreases in winter and this happens each year, as has been shown in figure 3 the peaks are similar and easy to predict.

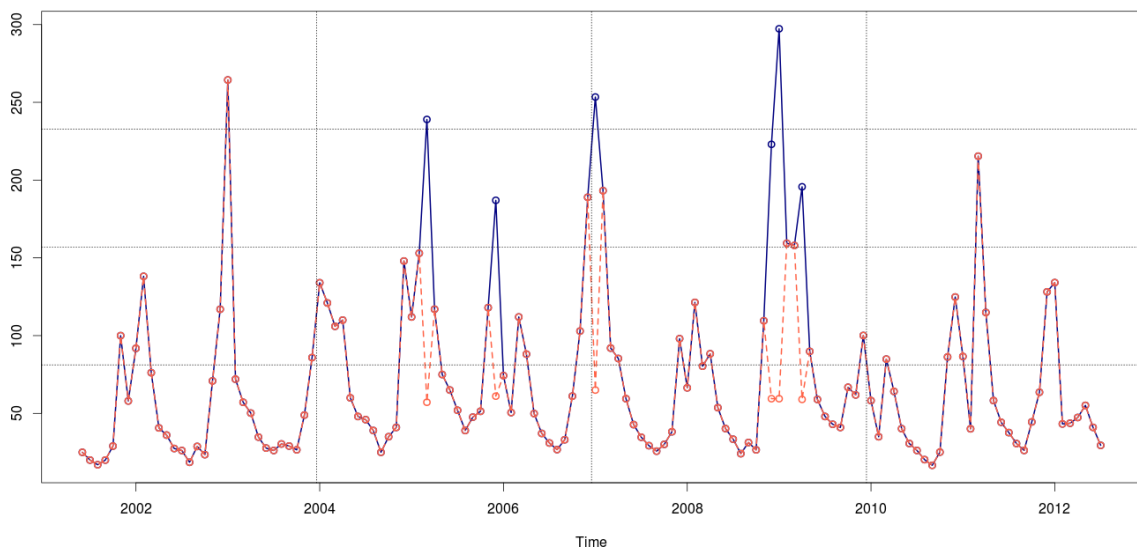


Figure 3: Example of seasonality [3]

2.3.2 Irregular

Irregular is exactly the inverse of seasonal, where the target is neither seasonal nor predictable, like the stock market and crypto market, as has been shown in figure 1, irregular has entirely different and unpredictable picks, which make it hard to do statistical modeling for it.

2.3.3 Trend

Trend is when a value decreases or increases within a long time period, decreasing refers to a downtrend, and increasing refers to an uptrend.

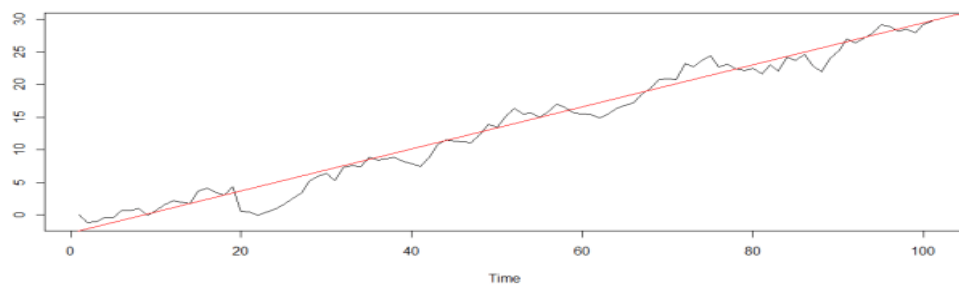


Figure 4: Example of a trend [4]

2.4 Machine learning

Is increasingly being talked about in the world of computing, and which relates to the field of artificial intelligence. Also called statistical learning, this term refers to a process of development, analysis, and implementation leading to the establishment of systematic processes. Simply put, it is a kind of program that allows a computer or machine to learn automatically so that it can perform a number of very complex operations [5].

The objective is to make the machine or computer capable of providing solutions to complicated problems, by processing an astronomical amount of information [6]. This thus offers an opportunity to analyze and highlight the correlations that exist between two or more given situations, and to predict their different implications.

Machine Learning is broken down into 2 steps: a training phase (we learn on a part of the data) and a verification phase (we test on the second part of the data). We will therefore have 3 phases: representation, evaluation, and optimization. The representation phase consists of finding the most suitable mathematical model. There are a large number of models. The evaluation measures the gap between the model and the reality of the test data. Finally, optimization aims to reduce this gap[5][6].

2.4.1. Naive Bayes classifier

Naive Bayesian classification methods are a set of supervised machine learning algorithms based on the application of Bayes' theorem with the assumption of a strong "naive" independence between each pair of features.

In other words, a naive Bayesian classifier assumes that the existence of a feature for a class is independent of the existence of other features!

Bayes' theorem allows us to calculate the conditional probability by the following formula

$$P(B_i|A) = P(A|B_i) * P(B_i) / P(A)$$

$P(B_i)$: the a priori probability of belonging to class B_i .

$P(A)$: the prior probability of A. It is also called the marginal probability of A.

$P(A|B_i)$: the conditional probability of seeing the input vector A knowing that the class is B_i .

Finally, the Bayesian estimator is:

$$C(A) = \operatorname{argmax}_{i \in \text{class}(m)} P(A|B_i) * P(B_i) / P(A)$$

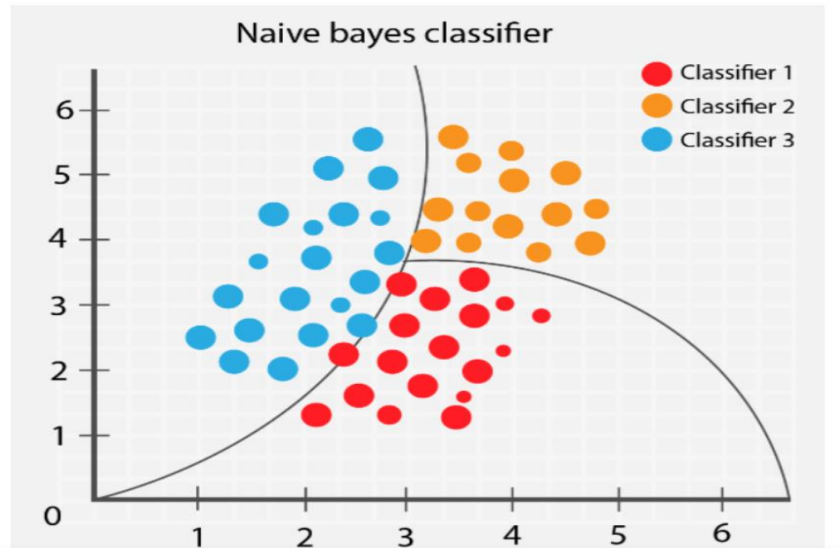


Figure 5: Naive base classifier illustration [7]

2.4.2 Principal Component Analysis

We can turn a group of correlated variables into uncorrelated variables called principal components using principal component analysis.

Its goal is to reduce the number of variables in order to reduce the amount of information that is redundant and the problem's complexity [8].

PCA is frequently used to describe and visualize data in a variety of fields.

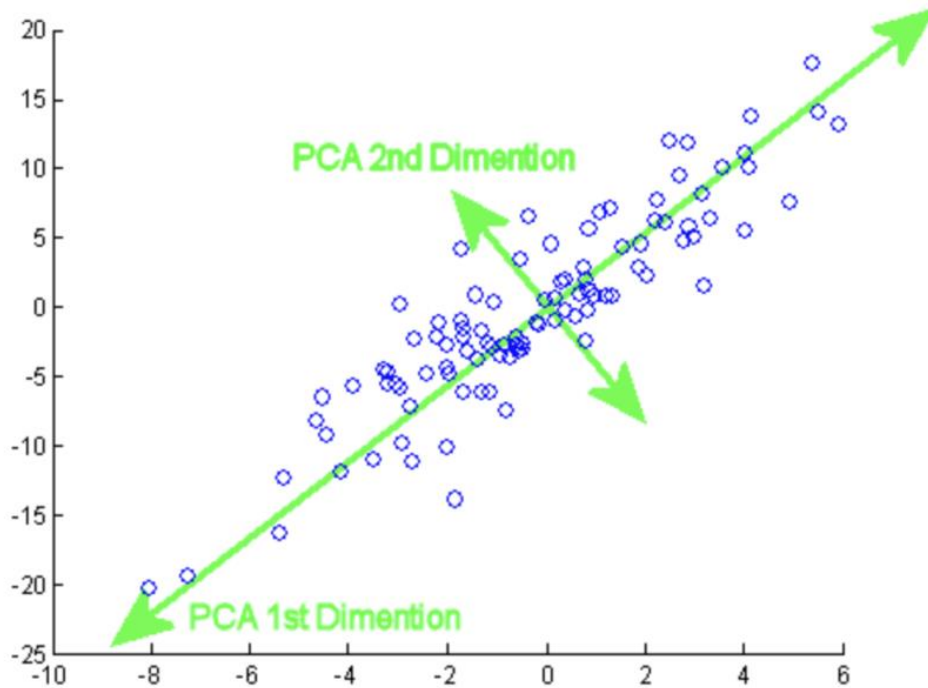


Figure 6: PCA illustration [9]

2.4.3. Decision Tree

A decision tree is a graphical structure in the shape of a tree (with leaves and branches) that depicts a set of options to help in decision-making and classify an input vector X . This method is commonly used in security and data mining.

Each node in the tree has a basic function comparison against a field. The outcome of each comparison is true or false, indicating whether we should go to the node's left or right leaf. Classification and regression trees are other names for decision trees (CART) [10].

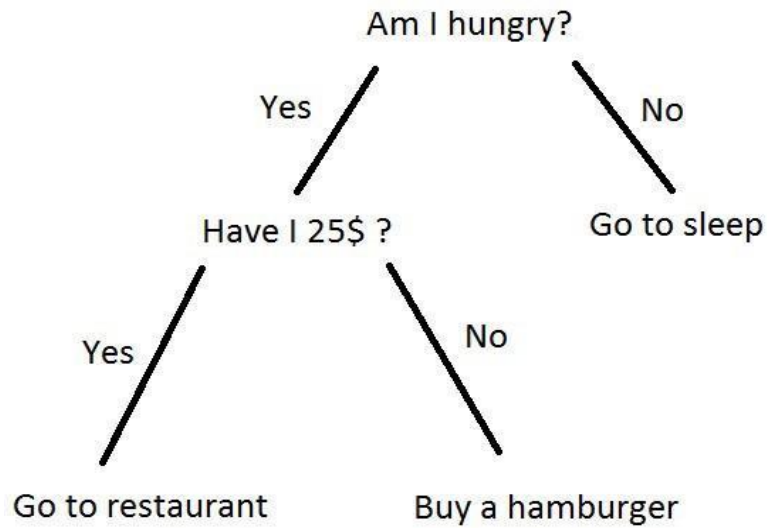


Figure 7: Decision Tree illustration[11]

Advantages:

- Non-linearity
- Support for categorical variables
- Easy to interpret
- Application to regression

Disadvantages:

- Prone to overfitting
- Unstable (not robust)
- High variance

2.4.4 K-means

The K-means algorithm divides a dataset into K unique, non-overlapping clusters in a simple and elegant way [12].

This algorithm is commonly used to solve problems like:

- Service delivery should be improved.

- Recognize the crime scene
- Identify and prevent fraudulent transactions.
- Computer alarms are automatically grouped.

The following is how K-means works:

choose k objects as the initial cluster's center, According to Euclidean or other distance, allocate each of the remaining samples to a division of the center of the cluster closest to it.

As the centroid of the new cluster, compute the mean vector of the samples given to each cluster.

Repeat steps 2 and 3 until the k clusters' centroid points don't change anymore or the total of squared errors is negligible.

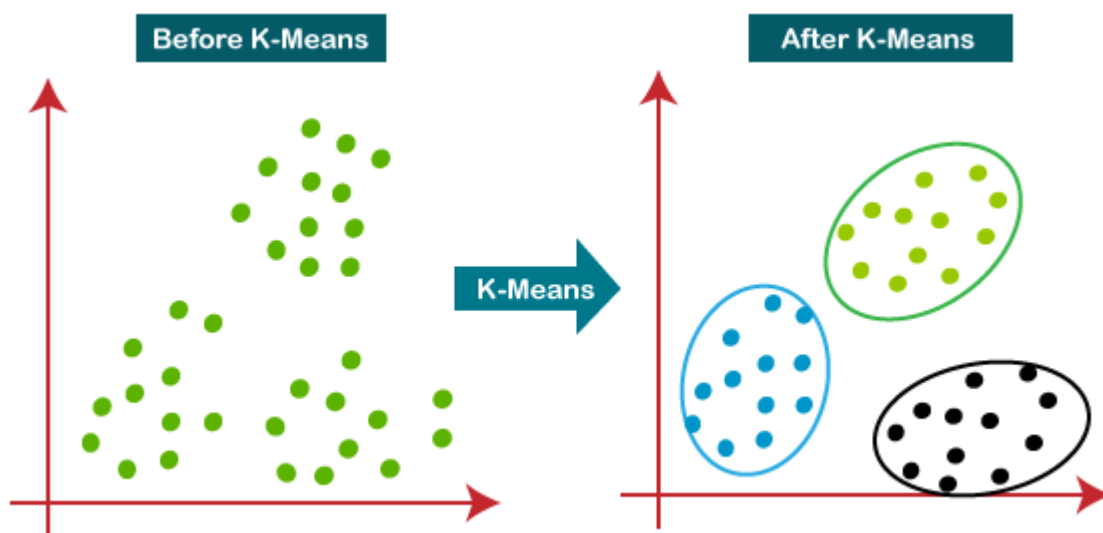


Figure 8: K-means illustration [13]

2.5 Deep learning

is a branch of machine learning that is entirely based on artificial neural networks, which are designed to mimic the human brain. As a result, deep learning is also a form of human brain imitation. We don't have to program everything directly in deep learning [14].

Deep learning is not a new concept. It's been around for quite some time. It's fashionable these days because we don't have nearly as much processing power or as much data as we do now. Deep learning and machine learning have emerged as processing power has increased dramatically over the previous 20 years.

It is a depiction of an individual neuron in the human brain, which has around 100 billion neurons in total, and each neuron is connected by thousands of their neighbors. The question is whether or not these neurons can be replicated in a computer. As a result, we develop an artificial neural network, which consists of nodes or neurons. In the buried layer, we have neurons for input values and others for output values, with many coupled neurons in between.

2.5.1: Going so deep

A neural network is a type of brain software or virtual computer that consists of thousands of calculation-based units (neurons). Logic and decision-making units (neurons, perceptrons) connect input and output data through a sophisticated network (network, brain) capable of generating complicated decisions.

These systems were originally referred to as Artificial Neural Networks (ANN) to distinguish them from biological systems. They usually have a set of input and output data (input/output layer), a small network of neurons, and multiple intermediate strata (hidden layers).

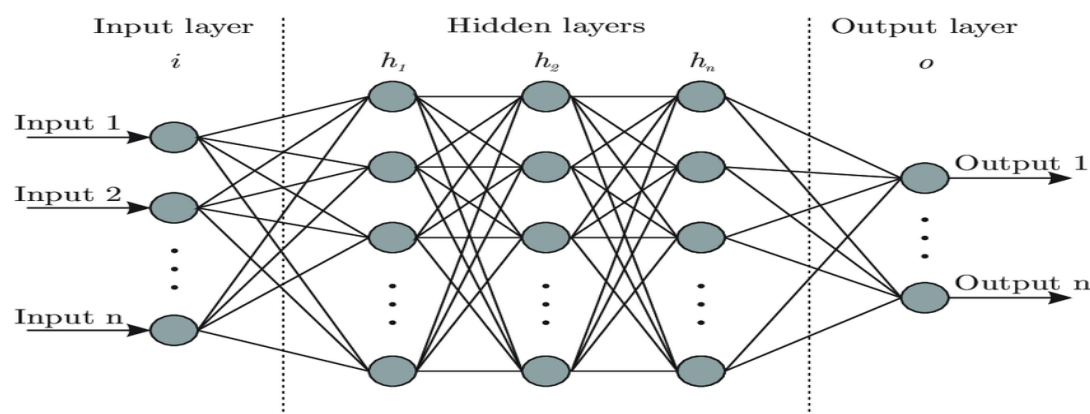


Figure 9: Artificial neural network architecture[15]

Three input units, one output, and two intermediate levels are shown in the example above (hidden layers). You discover that neurons are "highly linked," a crucial characteristic of brain networks. This is what permits complicated interactions, functions, and judgments to exist; without it, input-output relationships would be very basic.

The goal is to replicate the brain's ability to learn and detect complex connections, rather than to develop a precise model of the brain.

2.5.1 Artificial neuron?

The concept of an artificial neuron has yet to be fully defined. The logic unit of a processor is made up of transistors; we could also find a "wired" neural network there, but it would have to be "adaptive" and "learnable." Indeed, a neuron's reaction to incoming impulses must be able to change over time as it learns. This is referred to as "weighting," in which a neuron assesses (weights) multiple input variables in order to produce the desired output variable. This is why neurons are commonly thought of as mathematical functions that connect input and output variables.

Neurons adjust their weighting behavior and refine the output outcomes based on the input factors during the learning phase. As a result, each neuron must receive feedback from the overall result. As a result, we can argue that a neural network's input and output variables are known, but the values of the neurons, particularly in the buried layers, are unknown. We're dealing with a so-called "black box."

A neural network that has not been trained in isolation does not "know" anything and offers the user with random, even chaotic outputs. Only a well-trained system can deliver the intended outcome. If the problem is basic, it can be solved with a simple program that is easier to debug. We'll utilize a neural network to solve a more difficult problem, which we'll train with enormous data sets. Each neuron can produce complex output variables and respond to input variables in a linear or non-linear manner. This is a subtle aspect, because neurons must be able to respond to all possible scenarios in order to deliver a satisfactory output.

This might mean one of two things: either the neural network programmer is aware of all conceivable internal connections, or he has designed the network in such a complicated fashion that he has covered "all" possibilities.

2.5.2 RNN

Recurrent neural networks, or RNNs, are a type of neural network that is commonly employed in deep learning. RNNs are particularly suited for processing sequential data since they use past outputs as new inputs [14]. They usually adopt the following shape:

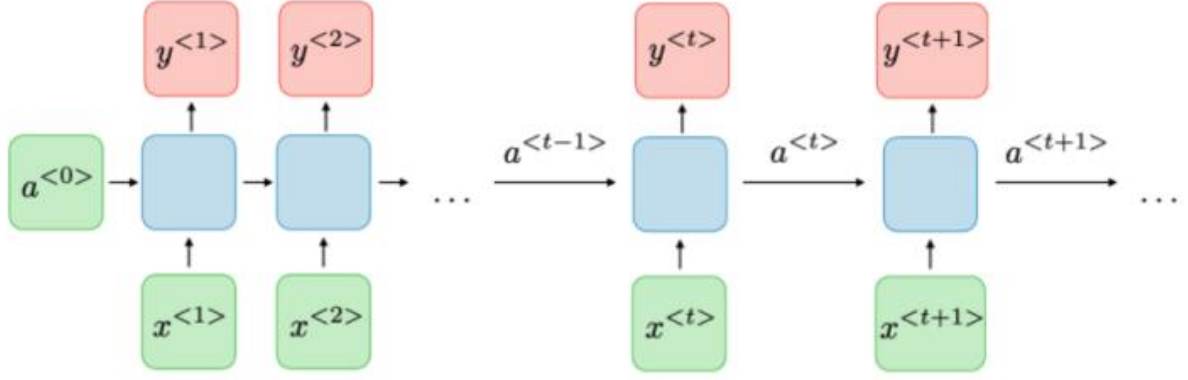


Figure 10: Recurrent Neural Network Architecture [16]

The forward pass is modelled by the following equations at each moment t :

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad (1)$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \quad (2)$$

Figure 11: RNN forward pass equations [14]

Unlike a traditional ANN (Artificial Neural Network) neural network, where the output is solely dependent on the input values, we can clearly see the recurring aspect in these calculations (the calculation at time t is based on the information provided at time $t-1$, which is itself calculated from the information provided at time $t-2$, etc.). This distinction is best illustrated in the following figure:

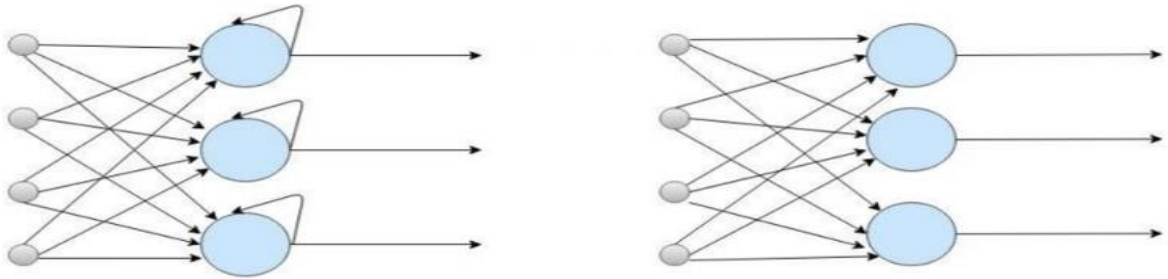


Figure 12: Distinction between RNN and ANN [17]

Although RNNs are more convenient for processing sequential input than a traditional ANN design, they are extremely difficult to train to handle long-term dependencies due to the removal of the gradient (Gradient Vanishing) [16]. Gradient explosions can also happen, but they are extremely rare. New RNN variations have been presented in the literature to address these flaws.

2.5.3 LSTM

S. Hochreiter et al [6] suggested LSTM to solve the gradient vanishing problem, which was later enhanced in an article by F. Gers et j. Schmidhuber [7]. The LSTM unit depicted in the following Figure is the main component of an LSTM architecture for overcoming the gradient vanishing problem. It's a collection of gates and cells that work together to achieve a final output. Equations are used to model an LSTM forward pass (3-8).

$$f_t = \sigma(W_f.x_t + U_f.h_{t-1} + b_f) \quad (3)$$

$$i_t = \sigma(W_i.x_t + U_i.h_{t-1} + b_i) \quad (4)$$

$$\tilde{c}_t = \tanh(W_c.x_t + U_c.h_{t-1} + b_c) \quad (5)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \quad (6)$$

$$o_t = \sigma(W_o.x_t + U_o.h_{t-1} + b_o) \quad (7)$$

$$h_t = o_t * \tanh(c_t) \quad (8)$$

Figure 13: LSM Block Equations [14]

Where σ is the sigmoid function, f_t is the Forget gate activation vector, i_t is the input gate activation vector, o_t is the output gate activation vector, d_t is the activation vector cell input, c_t is the cell state, h_t is the LSTM unit output vector, all W and U are weights, b is a bias vector, and $*$ is the hadamard product symbol. During the training phase, the weights W , U , and b biases must be learned.

Let's start with the cell's current state, which contains two sorts of data:

- the old information to keep from the previous state c_{t-1} , adjusted using the forget gate f_t , which determines the percentage of information to keep by calculating a value between 0 (discard completely) and 1 (keep completely).
- the new information to be included in the cell's state calculated using the input gate i_t , and the cell's activation calculated using equations 4 and 5.

The final value is then determined in two parts: first, a candidate value is calculated using equation 7, and then that value is changed to match the memory cell value to generate the final result.

Because the cell state is employed in the final computation, LSTM is highly useful in situations where data must be kept and used later (long-term dependency). The subject at the beginning of the phrase determines the conjugation of the verb to be employed in the middle or even at the end of the sentence, which is a simple example illustrating this issue.

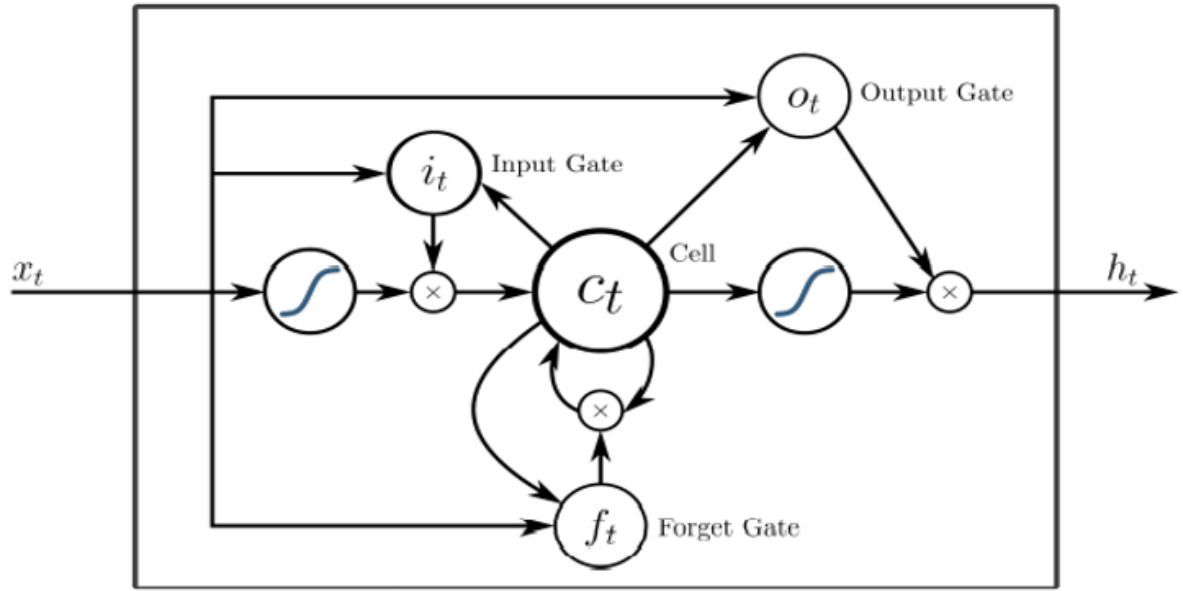


Figure 14: LSTM Block Architecture [18]

2.5.3 GRU

Cho and Al proposed the closed recurrent unit GRU (Gated Recurrent Unit) [8] in 2014 to overcome the problem of the disappearance of the gradient in standard recurrent networks, as well as to present architecture with less parameters to train than an LSTM. The GRU, like the LSTM, is the foundation of a GRU architecture. Equations (9-12) model a forward pass of the GRU unit:

$$z_t = \sigma(W_z.x_t + U_z.h_{t-1} + b_z) \quad (9)$$

$$r_t = \sigma(W_r.x_t + U_r.h_{t-1} + b_r) \quad (10)$$

$$\tilde{h}_t = \tanh(W_h.x_t + U_h(r_t * h_{t-1}) + b_h) \quad (11)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (12)$$

Figure 15: GRU Block equations [14]

Where the sigmoid function is σ , the update gate activation vector is z_t , the reset gate activation vector is r_t , the candidate vector is \tilde{h}_t , and the GRU output vector is h_t . W and U are weights, b is the bias vector (weights and biases are learned during the learning process), and $*$ is the Hadamard product sign.

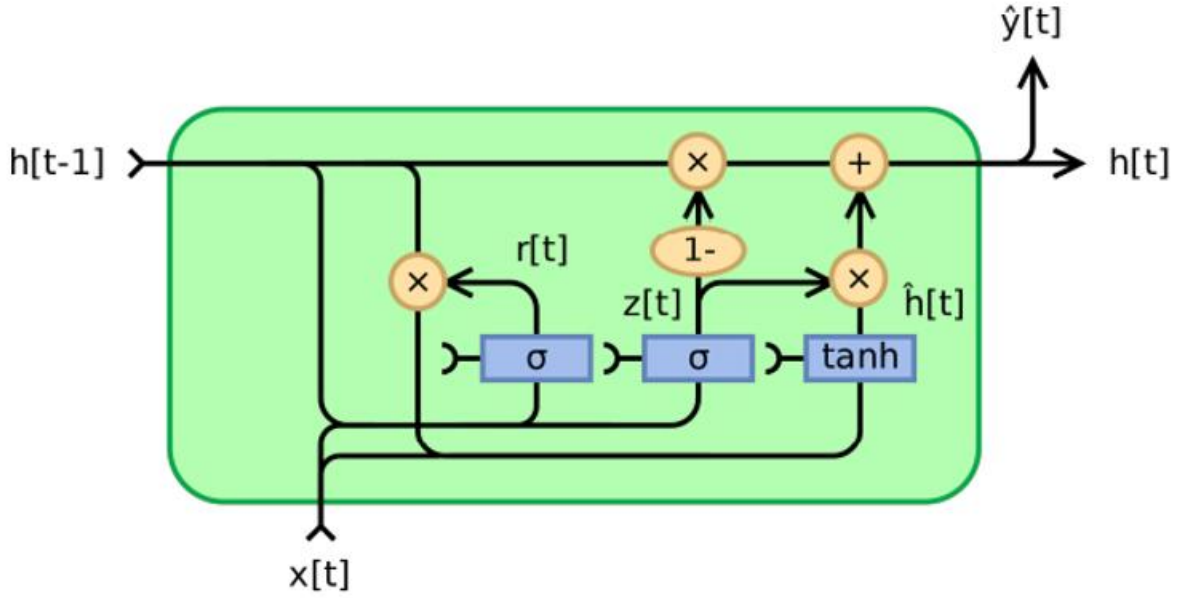


Figure 16: GRU Block Architecture [14]

Chapter 3: Design and Implementation

3.1 Tune-Quick Design and Conception

Tune-Quick is a web-based platform created to help in the creation of **end-to-end machine learning solutions**, the following diagram explains what is tune quick in a high level

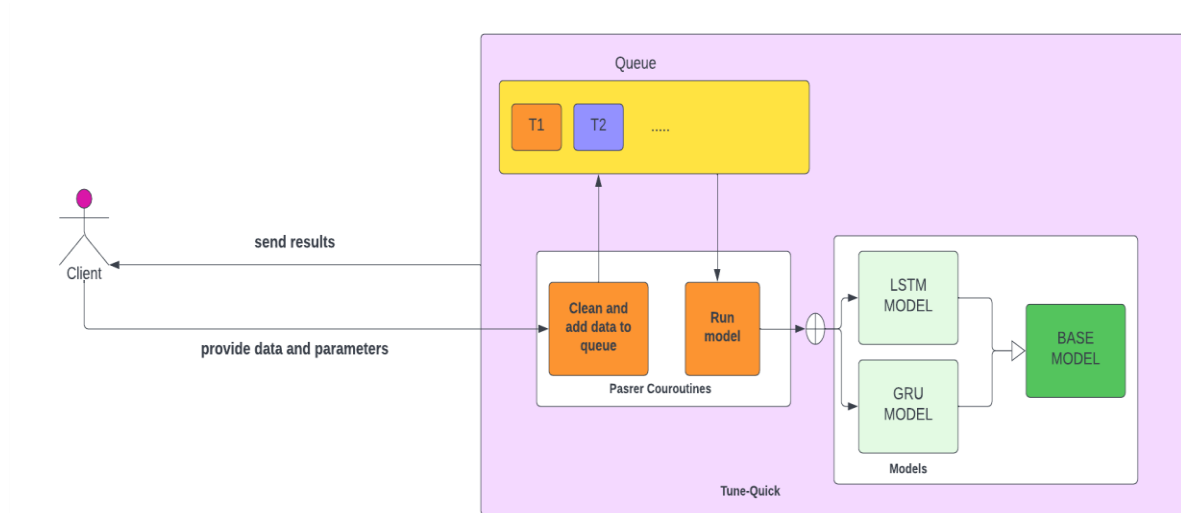


Figure 17: Tune-Quick

The figure above explains a high-level abstraction of how tune-quick is designed, in this chapter, there will be a detailed explanation of the technologies and patterns used as well as an explanation for all the parts of tune-quick.

3.2 Tune-Quick used technologies and methods

Several methods and technologies exist in the current world, only a few of them have been chosen based on the requirement and the need of this project.

3.2.1 Front-end

HTML

Stands for HyperText Markup Language and is a descriptive computer language. It is, more accurately, a data format for formatting Web pages on the Internet. It enables the creation of hypertext as well as the integration of multimedia resources into the content.

HTML is what allows a website creator to control how their web pages' content appears on a screen via the browser. It is based on a tag system that allows you to title, subtitle, bold, etc., text as well as include interactive components like images, links, and videos. The HTML language is easier for search engine crawl robots to understand than the JavaScript language, which is also used to make pages more interactive.

CSS

Cascading Style Sheets (CSS) is an acronym for Cascading Style Sheets. It is a style language with a very simple syntax yet exceptional performance. CSS is, after all, concerned with the formatting of HTML information.

So, whether you want to modify the color of your background, the font size of your text, the alignment and margins of your objects, and a variety of other things, CSS is there to help.

Bootstrap

It is a framework that contains all kinds of HTML and CSS-based design templates for various functions and components like navigation, grid system, image carousels and buttons.

Although Bootstrap saves developers time by repetitively dealing with templates, its primary purpose is to create responsive sites. It allows a website's user interface to perform optimally on all screen sizes, whether on small-screen phones or large-screen desktop computers.

3.2.2 Back-end

Python

Python is a free and open-source programming language created by Guido van Rossum in 1991. He gets his name from the Monty Python's Flying Circus television show.

It's an interpretive programming language, which means it doesn't need to be compiled to work. An application known as a "interpreter" allows you to run Python code on any computer. This allows you to see the results of a code modification right away. However, this makes this language slower than a compiled language like C.

Python, as a high-level programming language, allows programmers to concentrate on what they are doing rather than how they are doing it. As a result, writing programs takes less time than writing programs in another language.

Flask

is a simple and easy-to-use Python micro-framework for building scalable web apps. Flask relies on the Jinja templating engine and Werkzeug's WSGI toolkit.

- **Pandas**

is an open-source software library for manipulating and analyzing data in the Python programming language. It's powerful, adaptable, and simple to use.

The Python language now has the ability to load, align, manipulate, and even merge data thanks to Pandas. When the back-end source code is written in C or Python, performance is very outstanding.

Keras

is a Python language-based neural network API. It's a free library that runs on top of frameworks like Theano and TensorFlow.

Keras was built by Google employee François Chollet and is designed to be modular, quick, and simple to use. It allows you to develop Deep Learning models in a straightforward and intuitive manner.

A Keras Model is a self-contained sequence or graph. To develop new models, there are various fully adjustable components that can be combined.

One of the benefits of modularity is the ease with which additional capabilities may be added as distinct modules. Keras is thus extremely adaptable, making it ideal for research and development.

Asyncio

asyncio is a library for writing async/await-style concurrent programs. Several Python asynchronous frameworks, like asyncio, provide high-performance network and web servers, database connection libraries, distributed task queues, and more.

3.3 Tune-Quick Parser

The parser is a class that has the following

- Add model and dataset to the queue coroutine



```
async def add_model_with_dataset_to_queue(self, queue):
    try:
        if self.request.method == "POST":
            dataset = pd.read_csv(
                io.StringIO(self.request.files["csv-file"].stream.read().decode("UTF8")),
                newline=None)
            seq = Sequential()
            model = self.algorithms[self.request.form["algorithm"]](dataset, seq)
            queue.put_nowait(model)
    except Exception as err:
        raise err
```

Figure 18: *add_model_with_dataset_to_queue* coroutine

the figure shows the code for the *add_model_with_dataset_to_queue*, the coroutine will run only if the request method is **POST** because basically the function waiting for data and the parameters entered by users, then we store the data entered by the user in the dataset variable, also we pick up the suitable algorithms from the algorithms dictionary basically **LSTM OR GRU** for now.



Figure 18: The algorithms supported by Tune-Quick

After reading the data and choosing the correct algorithm(model), the coroutine will send the model to the queue as a task that it will be pick up by another process and run it.

- Run model coroutine

The run_model coroutine will pick up a task from the queue, which is in our case a model, then it will task the functions associated to that model in the following order:

1. Clean and prepare the dataset for the model
2. Split the data into the train and test set
3. Start training which takes the parameters from the users like the number of epochs, the optimizer type, and batch size...
4. Predict the results
5. Plot predictions and send the results to the front end

Also the user can choose the results chat title, x-axis, and y-axis



```

async def run_model(self, queue):
    if self.request.method == "POST":
        # get model from the queue
        model = await queue.get()
        logger.info('cleaning ans preparing the data...')
        try:
            model.clean_and_prepare_dataset()
            model.split_data()
        except Exception as err:
            raise err

        logger.info('training started...')
        model.train(
            optimizer=self.request.form["optimizer"],
            loss=self.request.form["loss-function"],
            epochs=int(self.request.form["epochs"]),
            batch_size=int(self.request.form["batch-size"]))
        logger.info('training done')
        model.predict()
        self.chart = model.plot_predictions(self.request.form["chart-title"], self.request.form["x-
axis"],
                                           self.request.form["y-axis"])

```

Figure 19: Run_model coroutine

- main coroutine



```

async def main(self):

    queue = asyncio.Queue()
    task1 = asyncio.create_task(self.add_model_with_dataset_to_queue(queue))
    task2 = asyncio.create_task(self.run_model(queue))
    await asyncio.gather(*[
        task1, task2]
    )

```

Figure 20: Main coroutine

The main steps are the following:

1. Create and initialize the queue
 2. Create a task for add model with data set coroutine
 3. Create task for run model
 4. And then gather them
- That means the two coroutines will run concurrently.

3.4 Queue

Queues are a powerful way to communicate between processes, basically, the aim is to make the coroutines independent from each other, so whenever a person submits a dataset and an algorithm it will be sent to the queue and many users can do this at the same time so each one will wait his turn, then the run model coroutine will pick up task from the queue and execute it and send the results and then pick up a new task from the queue while the queue is not empty.

3.5 Tune-Quick Models

Tune-quick has the base class **Model** which has the common attribute and functionalities across the submodels **GRUModel** and **LSTMModel**, the train function which is different from one model to another model, as shown in the class model below

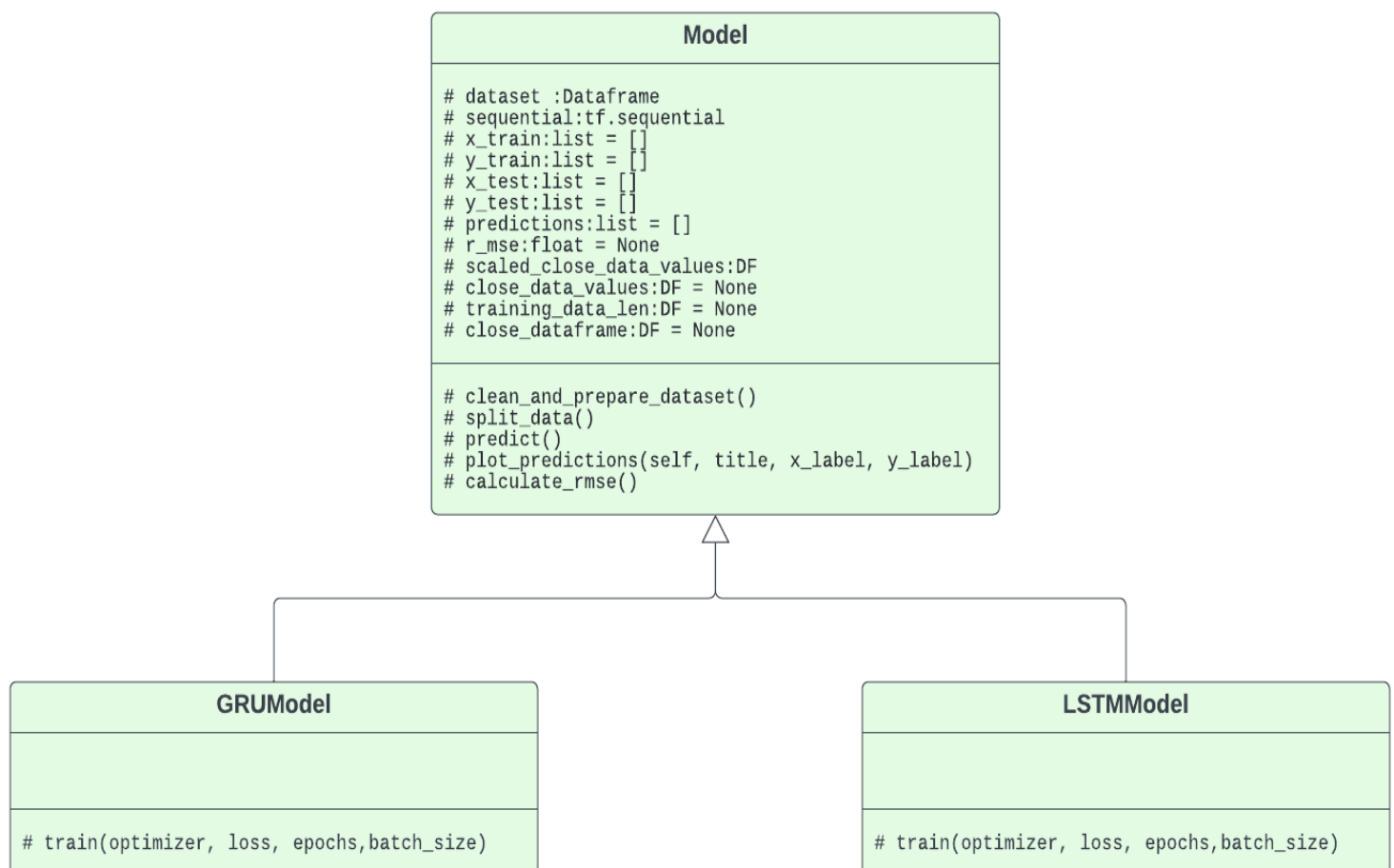


Figure 21: models class diagram

The functions of the model classes are explained below:

3.5.1 Model class

- ***Clean and prepare dataset function***

Clean and transform the data set into the correct format that suits the chosen algorithm later which consist of the following steps:

1. Drop duplicates
2. Drop nan values
3. Transform the date column from text to date
4. Set the index to date because the data is a timeseries

A code editor window with a white background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for a function named clean_and_prepare_dataset. The code uses docstrings, self.dataset for attribute access, pd.to_datetime for date conversion, and self.dataset.set_index to set the index. A logger.info statement is at the end.

```
def clean_and_prepare_dataset(self):  
    """transfer the data to the correct format"""  
    self.dataset.drop_duplicates(inplace=True)  
    self.dataset.dropna(inplace=True)  
    self.dataset["Date"] = pd.to_datetime(self.dataset["Date"])  
    self.dataset["Date"] = self.dataset["Date"].dt.date  
    self.dataset.set_index("Date", inplace=True)  
    logger.info("validation done")
```

Figure 22: *Clean_and_prepare_dataset fuction*

- ***Split data function***

Split the data into train and test it consists of the following steps :

1. Get the Close column which we will use on the training later
2. Take 95% from the actuall data as training data and the rest for the test set
3. Scale the data which is bery important for the training especially like avoiding local optima
4. Split the train into x_train and y_train
5. Then turn both to an array because later the model expect arrays as input

6. Reshape the arrays to suits recurrent neural network to the shape (x-train.shape[0], x-train.shape[1], 1)

```
def split_data(self):
    """split the data into x_train and y_train and scale it"""
    # Create a new dataframe with only the 'Close column
    self.close_dataframe = self.dataset.filter(['Close'])
    self.close_data_values = self.close_dataframe.values

    # Get the number of rows to train the model on
    self.training_data_len = int(np.ceil(len(self.close_data_values) * .95))
    self.scaled_close_data_values = scaler.fit_transform(self.close_data_values)
    train_data = self.scaled_close_data_values[0:int(self.training_data_len), :]

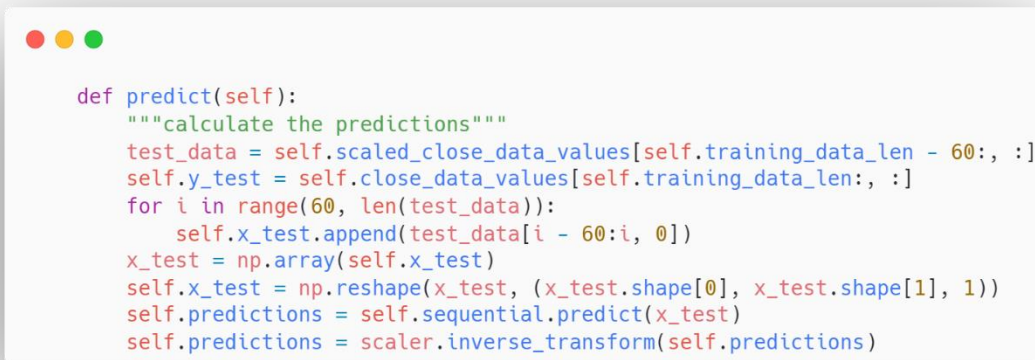
    # Split the data into x_train and y_train data sets
    for i in range(60, len(train_data)):
        self.x_train.append(train_data[i - 60:i, 0])
        self.y_train.append(train_data[i, 0])
    self.x_train, self.y_train = np.array(self.x_train), np.array(self.y_train)
    self.x_train = np.reshape(self.x_train, (self.x_train.shape[0], self.x_train.shape[1], 1))
```

Figure 23: Split_data function

- **Predict function**

Calculate the predictions based on the test set it consist of the following steps:

1. Get the rest of the data from the close data values which is about 5%
2. then basically repeats the same process as the splitting in the training data



```
def predict(self):
    """calculate the predictions"""
    test_data = self.scaled_close_data_values[self.training_data_len - 60:, :]
    self.y_test = self.close_data_values[self.training_data_len:, :]
    for i in range(60, len(test_data)):
        self.x_test.append(test_data[i - 60:i, 0])
    x_test = np.array(self.x_test)
    self.x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    self.predictions = self.sequential.predict(x_test)
    self.predictions = scaler.inverse_transform(self.predictions)
```

Figure 24: Predict function

- **Plot predictions function**

Plot and convert the plots to HTML elements in which will be displayed later in the front end, it consists of the following steps:

1. Get the training data
2. Get the validation data
3. plot the predictions
4. For the predictions, the line style is dashed
5. Then it takes the figure and converts it to HTML elements with the help of the mpld3 library

```

def plot_predictions(self, title, x_label, y_label):
    """plot predictions and convert it to html elements"""
    train = self.close_dataframe[:self.training_data_len]
    valid = self.close_dataframe[self.training_data_len:]
    valid['Predictions'] = self.predictions
    fig, ax = plt.subplots(figsize=(16, 6))
    ax.set_title(title, fontsize=15)
    ax.set_xlabel(x_label, fontsize=15)
    ax.set_ylabel(y_label, fontsize=15)
    ax.plot(train['Close'])
    # ax.plot(valid[['Close']])
    ax.plot(valid['Predictions'], linestyle="dashed")
    ax.legend(['Train', 'Predictions'], loc='lower right')
    plt.show()
    chart = mpld3.fig_to_html(fig)
    return chart

```

Figure 25: Plot_predictions function

- **Calculate rmse function**

Which calculates the root mean square error to ensure that the results are valid

```

def calculate_rmse(self):
    """calculate root mean square error"""
    self.r_mse = np.sqrt(np.mean(((self.predictions - self.y_test) ** 2)))

```

Figure 26: Calculate_rmse function

3.5.2 GRUModel class

Gru model inherits from the base class model with extra function which is train and it is different from a model to another model it designs the layer of the GRU model which is the following:

1. It defines 3 layers in each layer there is a 50 GRU units except the last layer which the dense layer where the final output produced.
2. It defines a drop out of 0.2 which skips some nodes to help reduce overfitting
3. It uses mean_squared_error as a default optimizer, an optimizer help to reach the local optima of the problem very quickly.
4. It uses 20 as the default value for a number of epochs

```
class GRUModel(Model):
    def __init__(self, dataset, sequential):
        super().__init__(dataset, sequential)

    def train(self, optimizer: str, loss='mean_squared_error', epochs=20,
              batch_size=32):
        self.sequential.add(GRU(units=50, return_sequences=True, input_shape=(self.x_train.shape[1], 1),
                                activation='tanh'))
        self.sequential.add(Dropout(0.2))
        self.sequential.add(GRU(units=50, return_sequences=True, input_shape=(self.x_train.shape[1], 1),
                                activation='tanh'))
        self.sequential.add(Dropout(0.2))
        self.sequential.add(GRU(units=50, return_sequences=True, input_shape=(self.x_train.shape[1], 1),
                                activation='tanh'))
        self.sequential.add(Dropout(0.2))
        self.sequential.add(GRU(units=50, activation='tanh'))
        self.sequential.add(Dropout(0.2))
        self.sequential.add(Dense(units=1))
        self.sequential.compile(optimizer=optimizer,
                                loss=loss)

    # fitting the model
    self.sequential.fit(self.x_train, self.y_train, epochs=epochs, batch_size=batch_size)
```

Figure 27: GRUmodel class

3.5.3 LSTMModel class

For now, LSTMModel uses a similar architecture to the GRU, but instead of GRU units it uses lstm units.

```
class LSTMModel(Model):  
  
    def __init__(self, dataset, sequential):  
        super().__init__(dataset, sequential)  
  
    def train(self, optimizer: str, loss='mean_squared_error', epochs=1, batch_size=32):  
        self.sequential.add(LSTM(units=50, return_sequences=True, input_shape=(self.x_train.shape[1],  
1)))  
        self.sequential.add(Dropout(0.2))  
        self.sequential.add(LSTM(units=50, return_sequences=True))  
        self.sequential.add(Dropout(0.2))  
        self.sequential.add(LSTM(units=50, return_sequences=True))  
        self.sequential.add(Dropout(0.2))  
        self.sequential.add(LSTM(units=50))  
        self.sequential.add(Dropout(0.2))  
        self.sequential.add(Dense(units=1))  
        self.sequential.compile(optimizer=optimizer, loss=loss)  
        self.sequential.fit(self.x_train, self.y_train, epochs=epochs, batch_size=batch_size)
```

Figure 28: LSTMModel class

3.6 Tune-Quick app module

app module is where the routes of the web app exists, for now the app run on the root /
Routes defined by the decorator @app.route()

index() function consists of the following steps:

1. Initiate a parser object
2. Pass request to the object which has the information submitted by the user
3. Then run the main coroutine which will run the parser coroutines
4. Then the result will be saved in the chart attribute, the it will rendered to the front end to display the result for the end-user.
5. To run the flask app we need to call app.run() function



```
import asyncio

from flask import Flask, render_template, request

from src.parser import Parser

app = Flask(__name__)

@app.route("/", methods=["POST", "GET"])
def index():
    parser = Parser(request=request)
    asyncio.run(parser.main())
    return render_template("index.html", bar_chart=parser.chart)

if __name__ == "__main__":
    app.run(debug=True)
```

Figure 29: App module

3.7 Tune-Quick Case Study

In this case study, a full experiment will be held to cover the capabilities of Tune-Quick

3.7.1 Dataset

Tune-Quick deals with stock market time-series related data, the figure below shows a sample of it:

	A	B	C	D	E	F	G	H
1	Date	Open	High	Low	Close	Volume	Name	
2	1/3/2006	211.47	218.05	209.32	217.83	13137450	GOOGL	
3	1/4/2006	222.17	224.7	220.09	222.84	15292353	GOOGL	
4	1/5/2006	223.22	226	220.97	225.85	10815661	GOOGL	
5	1/6/2006	228.66	235.49	226.85	233.06	17759521	GOOGL	
6	1/9/2006	233.44	236.94	230.7	233.68	12795837	GOOGL	

Figure 30: Dataset sample

Tune-Quick takes the Close column which refers to the last traded price as an input for the models.

3.7.2 User Interface

Tune-Quick uses a web-based interface for the user, the interface supports uploading CSV files, and adjusts the parameters for the models as shown below:

Welcome to TUNE-QUICK, Train deep learning algorithms with 0 code

TUNE-QUICK helps you to explore which algorithm and parameters is the best for your dataset

Upload Your CSV File

Choose File

No file chosen

Choose Algorithm

Choose...

Choose loss function

Choose an optimizer

Enter number of epochs...

Enter chart title...

Enter batch size...

Enter x-axis title...

Enter y-axis title...

Train

Figure 31: Tune-Quick Interface

The user interface supports the following:

- Add a CSV file, it must be the same format as the figure ..
- Choose the optimizer, for now, there is only one optimizer
- Choose the algorithm that you want to experiment for now it supports two algorithms
- Choose the number of epochs
- Choose the batch-size
- Wrote the title for your resulting chart
- Wrote the title of the x-axis
- Wrote the title of the y-axis

3.7.3 Results

Based on the following inputs:

The screenshot displays the TUNE-QUICK web interface. At the top, it says 'Welcome to TUNE-QUICK, Train deep learning algorithms with 0 code' and 'TUNE-QUICK helps you to explore which algorithm and parameters is the best for your dataset'. Below this, there are two main sections. The first section, 'Upload Your CSV File', includes a 'Choose File' button and a text input field containing 'IBM_2006-01-01_to_2018-01-01.csv'. To the right, under 'Choose Algorithm', is a dropdown menu currently set to 'LSTM'. The second section contains several input fields: a dropdown for 'mean_squared_error', a dropdown for 'adam', a text input for '10', a text input for '16' (which is highlighted with a blue border), and three text inputs for chart titles: 'IBM STOCK PRICES PREDICTION', 'Dates', and 'PRICE(USD)'. At the bottom of this section is a large 'Train' button.

Figure 32: Inputs example

When the train button is clicked the process starts in the backend, Tune-Quick uses a logger instead of the print function to track the output of the system which make it easy to debug when a problem occurs, basically, the following steps happened after submitting your data:

1. Cleaning and preparing data
2. Validate the data
3. Start the training

4. Finish the training and send the results back to the user

```
INFO:root:cleaning ans preparing the data...
INFO:root:validation done
INFO:root:training started...
Epoch 1/10
176/176 [=====] - 13s 48ms/step - loss: 0.0164
Epoch 2/10
176/176 [=====] - 8s 46ms/step - loss: 0.0048
Epoch 3/10
176/176 [=====] - 8s 44ms/step - loss: 0.0038
Epoch 4/10
176/176 [=====] - 8s 44ms/step - loss: 0.0041
Epoch 5/10
176/176 [=====] - 8s 44ms/step - loss: 0.0037
Epoch 6/10
176/176 [=====] - 8s 44ms/step - loss: 0.0036
Epoch 7/10
176/176 [=====] - 8s 44ms/step - loss: 0.0034
Epoch 8/10
176/176 [=====] - 8s 44ms/step - loss: 0.0028
Epoch 9/10
176/176 [=====] - 8s 44ms/step - loss: 0.0027
Epoch 10/10
176/176 [=====] - 8s 44ms/step - loss: 0.0025
INFO:root:training done
```

Figure 33: Tune-Quick backend logger example

The user sees the results as a chart, that contains the training data with the prediction part as well as the titles that have been provided by the user see the following figure:

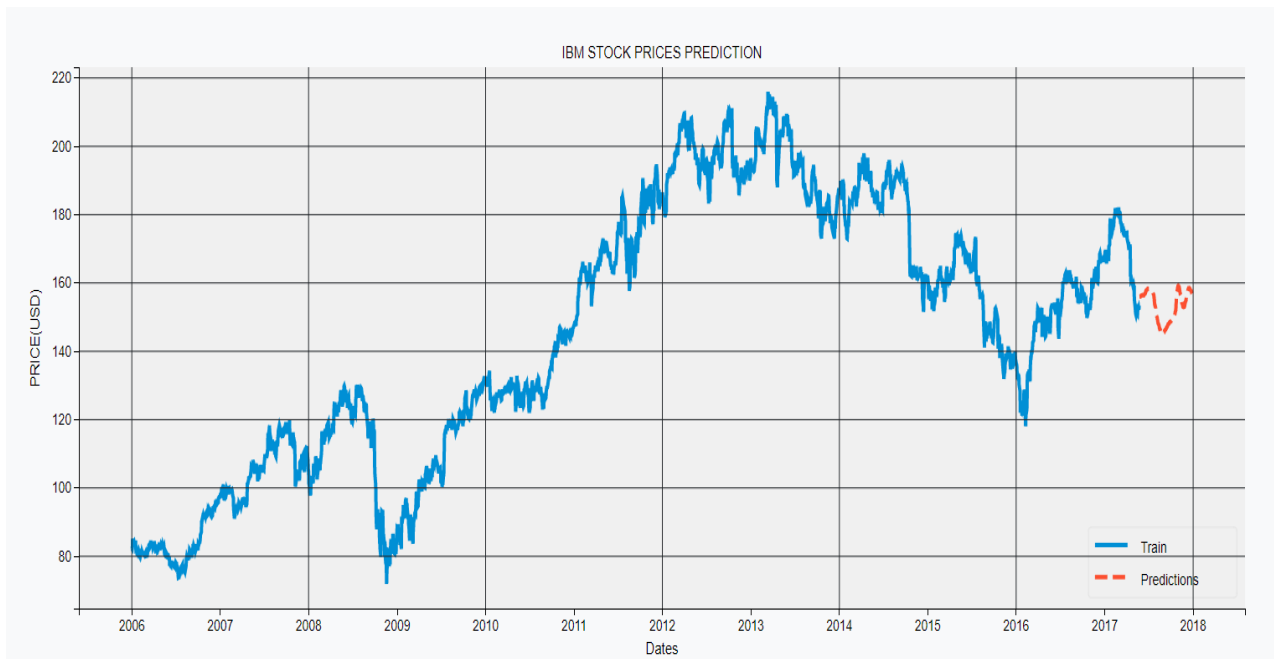


Figure 34: The predicted chart

Tune-Quick ensures that the prediction is practical by using a small interval prediction.

4. Summary

Machine Learning Approach is a heavily repetitive task, where many researchers and developers believe that tuning machine learning algorithms can be automated to some points.

The question was can we automate machine learning solutions pipelines ? and therefore we focus only at experimenting various machine learning algorithms with less effort and less knowledge about AI, then we avoid nonsense repetitions, gain time, and perform a lot of solution within a small amount of time.

Tune Quick now has some services to deliver, the base structure has been established and it works fine, users can try some algorithms in the platform.

Tune Quick help gives you a full scale for your machine learning solutions as follow:

- Upload and Clean the data for you
- Gives you a set of algorithms to try on your data
- Gives you the ability to tune your parameters across your model
- Display the result for you

Tune Quick's next Goals are the following:

- Provide more algorithms
- Provide a drag and drop drawing interface which allows you to draw complex architectures and then the user need to just compile the architecture within Tune-Quick and wait for the results.
- Provide a desktop version of Tune-Quick based on C++, for Computer Vision related problems

References

- [1] Binance. 2022. *0.8762 / ADABUSD / Binance Spot*. [online] Available at: <https://www.binance.com/en/trade/ADA_BUSD> [Accessed 4 May 2022].
- [2] Medium. 2022. *Stationarity in time series analysis*. [online] Available at: <<https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322>> [Accessed 4 May 2022].
- [3] outliers, F., 2022. *Filtering seasonal time series outliers*. [online] Stack Overflow. Available at: <<https://stackoverflow.com/questions/12218584/filtering-seasonal-time-series-outliers>> [Accessed 4 May 2022].
- [4].ORAYLIS. 2022. *Trend in times series analysis - ORAYLIS*. [online] Available at: <<https://www.oraylis.de/blog/2015/trend-in-times-series-analysis>> [Accessed 4 May 2022].
- [5] El Naqa and M. Murphy, "What Is Machine Learning?", *Machine Learning in Radiation Oncology*, pp. 3-11, 2015. Available: 10.1007/978-3-319-18305-3_1 [Accessed 4 May 2022].
- [6] M. Jordan and T. Mitchell, "Machine learning: Trends, perspectives, and prospects", *Science*, vol. 349, no. 6245, pp. 255-260, 2015. Available: 10.1126/science.aaa8415 [Accessed 4 May 2022].
- [7] "Building Naive Bayes Classifier from Scratch to Perform Sentiment Analysis", *Analytics Vidhya*, 2022. [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/03/building-naive-bayes-classifier-from-scratch-to-perform-sentiment-analysis/>. [Accessed: 04- May- 2022].
- [8] Z. Jaadi, "A Step-by-Step Explanation of Principal Component Analysis (PCA)", *Built In*, 2022. [Online]. Available: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. [Accessed: 04- May- 2022].
- [9] "Principal Components Analysis Explained for Dummies - Programmatically", *Programmatically - A Blog on Building Machine Learning Solutions*, 2022. [Online]. Available: <https://programmatically.com/principal-components-analysis-explained-for-dummies/>. [Accessed: 04- May- 2022].
- [10] P. Gupta, "Decision Trees in Machine Learning", *Medium*, 2022. [Online]. Available: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>. [Accessed: 04- May- 2022].
- [11] E. Dezhic, "Understanding decision trees," *Medium*, 11-Jul-2017. [Online]. Available: <https://becominghuman.ai/understanding-decision-trees-43032111380f>. [Accessed: 05-May-2022].
- [12] C. Piech, "K Means," *CS221*. [Online]. Available: <https://stanford.edu/~cpiech/cs221/handouts/kmeans.html>. [Accessed: 05-May-2022].

- [13] S. Agrawal, "To start with K-means clustering," *Medium*, 18-Jan-2021. [Online]. Available: <https://medium.com/analytics-vidhya/to-start-with-k-means-clustering-1c6ee3cb840f>. [Accessed: 05-May-2022].
- [14] J. Brownlee, "What is deep learning?," *Machine Learning Mastery*, 14-Aug-2020. [Online]. Available: <https://machinelearningmastery.com/what-is-deep-learning/>. [Accessed: 05-May-2022].
- [14] Touzani, Y., 2021. *Deep Learning : les réseaux de neurones récurrents (RNN)*. [online] DataValue Consulting. Available at: <<https://datavalue-consulting.com/deep-learning-reseaux-neurones-recurrents-rnn/>> [Accessed 4 May 2022].
- [15] F. Bre, J. M. Gimenez, and V. D. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using artificial neural networks," *Energy and Buildings*, vol. 158, pp. 1429–1441, 2018.
- [16] A. Amidi, "Recurrent neural networks cheatsheet star," *CS 230 - Recurrent Neural Networks Cheatsheet*. [Online]. Available: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>. [Accessed: 05-May-2022].
- [17] P. Walpita, "Recurrent neural networks in deep learning - part 1," *Medium*, 01-Apr-2020. [Online]. Available: <https://medium.datadriveninvestor.com/recurrent-neural-networks-in-deep-learning-part-1-df3c8c9198ba>. [Accessed: 05-May-2022].
- [18] "long short-term memory" *Wikimedia Commons*. [Online]. Available: https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg. [Accessed: 05-May-2022].
- [19] "Gated Recurrent Unit, base type.- Wikimedia Commons", *Commons.wikimedia.org*, 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit,_base_type.svg. [Accessed: 04-May-2022].