Gabriel Duarte da Silva

ARTIFICIAL NEURAL NETWORKS APPLIED TO UNMANNED AERIAL VEHICLE CONTROL

Undergraduate research report for Group of Intelligent Materials and Systems (GMSINT).

São Paulo State University (UNESP)
School of Engineering of Ilha Solteira
Mechanical Engineering Department

Supervisor: Douglas D. Bueno

Ilha Solteira, SP
2024

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

The use of Artificial Intelligence (AI) is very prevalent nowadays (**lee2020**, **poola2017**, **rabunal2006**). This area of statistics neither is new nor started just now with the autonomous cars and voice assistants (**muthukrishnan2020**), but it is clear that in recent years, it has been increasingly gaining more popularity. This happens mainly because of the advances that the World Wide Web has had over the years (**leiner2009**, **cohen-almagor2011**), from dial-up internet connection back in the eighties to now, with broadband internet and smartphones equipped with 5G connection. Another factor is that in the past, the cost to obtain a large capacity of storage memory was significantly more expensive than it is now, making it cheaper and easier to acquire memory for storing information today (**goda2012**). With the amount of data available, the evolution of internet, and storage capacity, it is now not difficult to obtain, maintain and analyze databases to make decisions (**duan2019**).

AI application is everywhere and today, more than ever, it is easy to realize that. Whether it is to receive multimedia recommendations on streaming platforms, as is the case with Netflix, YouTube, Spotify, and many others (**chan-olmsted2019**), or to make predictions on the financial market and sports betting (**milana2021**, **kollar2021**, **hubacek2019**), AI is behind the scenes making all the magic happen. Evidently there is nothing truly magical about them; it is pure mathematics combined with a programming language that produces the algorithm capable of performing these tasks (**goodfellow2016**, **aurelien2022**, **raschka2015**, **raschka2022**). The launch of ChatGPT–3, and shortly thereafter ChatGPT–4, has shown the power of those technologies and how they can change the way people do things (**biswas2023**, **biswas2023a**, **lund2023**, **baidoo-anu2023**).

In the application of smart systems, the use of AI is widespread in Structural Health Monitoring (SHM), which is heavily used in the aerospace and civil fields, (**azimi2020**, **ye2019**). The level and the complexity of the AI to be applied to monitor the structure, whether is going to use deep learning and artificial neural networks (ANNs) or simpler machine learning methods like regressions, are determined by the problem itself and the results desired (**farrar2012**). In some cases, standard methods using numerical techniques may not be feasible, especially when dealing with extensive data analysis. Thus, opting for the AI approach becomes an alternative to obtain the necessary monitoring results in a more practical manner (**smarsly2007**, **sun2020**).

Still in this context, but in the field of Unmanned Aerial Vehicle (UAV), the use of AI can be combined to integrate UAV through wireless communication networks (**lahmeri2021**). This integration proves valuable in the agriculture sector (**ahirwar2019**) incorporating technologies such as the Internet of Things (**verdouw2016**, **tzounis2017**). Furthermore, AI applications can be subtle, such as utilizing a built-in MATLAB function to make a simple ANN to determine the final pose of a UAV based on the initial pose and the forces applied on it (**geronel2023**). It can be more sophisticated, involving the use of machine and deep learning algorithms to predict materials properties, design new materials, discover new mechanisms and control real dynamic systems (**guo2021**, **assilian1974**).

It is clear, therefore, that AI can transit into different fields, including entertainment, business, health care, marketing, financial, agriculture, engineering, among others (**ruiz-real2020**, **yu2018**, **davenport2019**, **verma2021**, **mhlanga2020**, **pannu2015**, **ghatrehsamani2023**). The use of Big Data not only clarifies the scenario to be studied but also supports the formulation of strategic decisions (**jeble2018**, **koscielniak2015**). The improvement in internet and hardware capabilities (**baji2018**), alongside the ease of storing data at accessible costs, encourages the AI use due to the benefits it can provide.

1.1 Motivation

With the 4.0 industry, the engineering evolution is growing bigger every year (**meindl2021**). Solving engineering problems the traditional way may not be the best solution due to its non-triviality to complex systems and their mathematical modeling.

2 LITERATURE REVIEW

This chapter deals with the history, the main concepts and some practical cases of SHM inside the industry and academic area, besides showing how it may be used in the railway crack detection context. Next, in the dynamic field, it will be studied the main mechanical concepts to get the necessary understanding to an UAV motion as well the basics to know how an UAV can be controlled. Then, it will be shown the mathematics behind the algorithms of deep learning that will be implemented in the **??**. Finally, the way how the algorithms are going to be implemented and the tools necessary to achieve the desired ANN.

2.1 Unmanned Aerial Vehicle Control

2.1.1 Usage

An UAV has several applications, going from the simplest to the most sophisticated. It can be used since for entertainment, like toys; commercially, to record big shows in arenas; surveillance, to monitor places; and also in engineering, aiding in various context to improve some processing.

Due to its portability and autonomy, it can be used to facilitate the delivery o medicines. In this sense, UAV can be used for transportation of medical goods in critical times, where other means of transportation may not be feasible. In the final of 2019, COVID-19 pandemics spread throughout the world, making it difficult to deliver patients their needed medicines (**ramakrishnan2023**, **mcphillips2022**). Besides, risks are inherent to the transportation and in come countries, like the USA, UAV usage may be restricted (**thiels2015**). A strategical way to use them is also welcome.

In the agriculture context, in order to boost the productivity, UAV can be used to remotely sense the farming, obtaining information on the state of the fields with non-contact procedures, like nutrient evaluation and soil monitoring; or even for aerial spraying, using pesticide to prevent damages in the plantation (**delcerro2021**).

The main reason for its adoptions is the mobility, low maintenance costs, hovering capacity, ease of deployment, etc. It is widely used for the civil infrastructure, gathering photographs faster than satellite imagery and with better quality. Combining those benefits with AI can be a powerful tool for the future (**sivakumar2021**).

2.1.2 Control Equations

Considering the UAV a quadcopter, as the **??** shows, **geronel2023<empty citation>**, based on the work of **fossen1994<empty citation>**, described the equation of motion for a quadcopter with a payload as being:

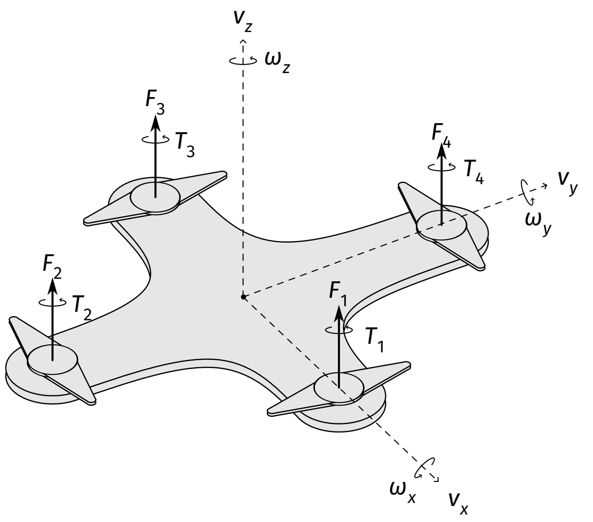
$$\mathbf{M}_{\eta_c}(_c)\ddot{}_c + \mathbf{C}_{\eta_c}(, _c)\dot{}_c + \mathbf{g}_{\eta_c}(_c) + \mathbf{K}_{\eta_c}(_c)_c = {}_{\eta_c}(_c) + \mathbf{F}_d \tag{2.1}$$

where $\mathbf{M}_{\eta_c}(_c)$ is the inertial matrix; $\mathbf{C}_{\eta_c}(,_c)$ is the Coriolis matrix; $\mathbf{g}_{\eta_c}(_c)$ is the gravitational vector; $\mathbf{K}_{\eta_c}(_c)$ is the stiffness matrix; $_{\eta_c}(_c)$ is the control torque; $\mathbf{F}_d$ is the gust vector; and is the velocity generalized coordinate in the body-frame. The **??** can be represented in the state space form as:

$$\dot{x}_s = \mathbf{A}_c x_s(t) + \mathbf{B}(u)(t) + \mathbf{X} \tag{2.2}$$

where $x_s = \left\{_c \quad _c\right\}^\mathsf{T}$ is the state vector; $\mathbf{B}(u)(t)$ is the input vector; $\mathbf{X}$ is the state vector of gravity; and $\mathbf{A}_c$ and $\mathbf{B}$ are the dynamic and input matrices, respectively.

Figure 1 – Quadcopter Dynamic Scheme. F_i and T_i, ($i = 1,2,3,4$), are the forces and the torque applied in the propeller, respectively. ω_j and v_j, ($j = x,y,z$), are the momentum and the velocities applied in the UAV, respectively. The payload is not represented in the figure.



Source: prepared by the author.

All non-explicit matrices and the development of the equations are shown in the **geronel2023<empty cita**
work.

2.1.3 Control Algorithm

geronel2023<empty citation> developed a MATLAB algorithm to control the quadrotor, as a *white box* method. It controls the UAV in three different trajectories: rectangular, circular and linear. Given as the input vector, which represents the position controller $U_1(t)$ and the attitude controller $U_2(t)$, $U_3(t)$, $U_4(t)$, it is able to give a complete overview of the quadrotor's motion. The algorithm provides the state space vector $\mathbf{x}_s$ with the quadrotor position and angles, as their derivatives.

$$\tau = \left\{U_1 \quad U_2 \quad U_3 \quad U_4\right\}^\mathsf{T} \tag{2.3}$$

$$\mathbf{x}_s = \left\{x \quad y \quad z \quad \phi \quad \theta \quad \psi \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi}\right\}^\mathsf{T} \tag{2.4}$$

2.2 Artificial Neural Networks

2.2.1 Deep Learning

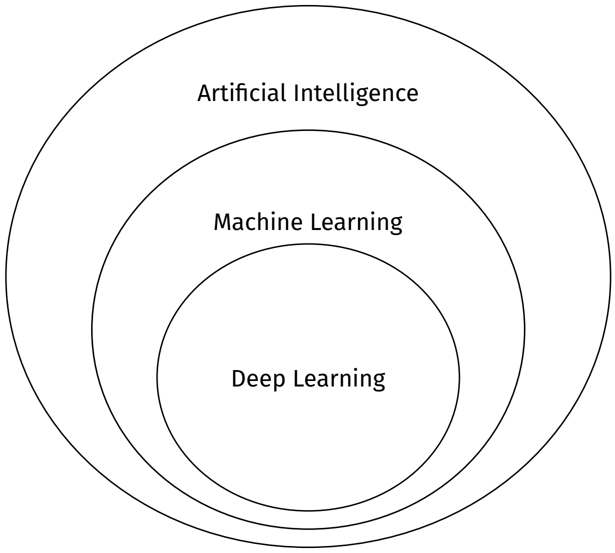
The concepts of deep learning studied in this section is going to be based on the work of **goodfellow2016<empty citation>**, **haykin1999<empty citation>** and the documentation of PyTorch.

There are several definitions of AI (**winston1992**), but the computer scientist **mccarthy2007<empty** defines it as:

> *[...] the science and engineering of making intelligent machines, especially intelligent computer programs [...] it is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.*

The big area of study is the AI and it includes several branches like fuzzy logics, robotics and machine learning. The later one, in turn, is another field with also some branches and one of them is the deep learning. However, all the three terms can be interchangeable in the major context.

Figure 2 – The Relationship Between Artificial Intelligence, Machine Learning, and Deep Learning. Deep learning is a specific field inside the artificial intelligence umbrella.



Source: prepared by the author.

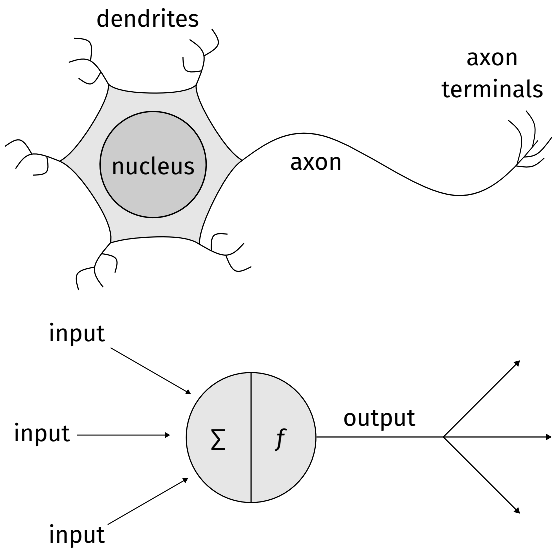
The deep learning history goes back to the 1940s and it had several names over the years. It was called by *cybernetics* (1940s–1960s), *connectionism* (1980s–1990s), and from 2006 until now is known as deep learning. The deep learning models were engineered systems inspired by the biological brain and they were denominated ANN. One of the motivations of the neural perspective was to understand that the brain provides a proof by example that intelligent behavior is possible and try to reverse engineer the computation principals behind the brain, duplicating its functionality. Today it goes beyond the neuroscientist perspective and it is more of general principle of learning multiple levels of composition.

DL dwells in the programming sphere. The approach, however, it is not like the traditional programming scripts and models. To automate stuff, there are three main parts: (i) the input data, (ii) the rule (function) and (iii) the output data. In oth types there are two of three parts available, but different ones for each other. In the traditional programming, there is the input data and the rule, for the algorithm output the data. For DL, there is the input data and the output data, for the algorithm provides the rule. A good analogy is cooking: in the traditional programming context, one has the ingredients and the recipe to make the main course; in the deep learning context, one has the ingredients and the main course to discover the recipe.

2.2.2 Neural Networks Models

A ANN is machine learning a model that simulate a biological ANN to make a machine learns as the human being learns. ANN are the heart of deep learning and there are several models of them, each one most suitable for different kind of problems. Some of them are multi-layer perceptron (MLP), convolutional neural network, recurrent neural network, among others.

Figure 3 – Artificial and Biological Neuron. Dendrites are corresponding with the inputs; nucleus with sums and activation functions; and axon terminals with the outputs.



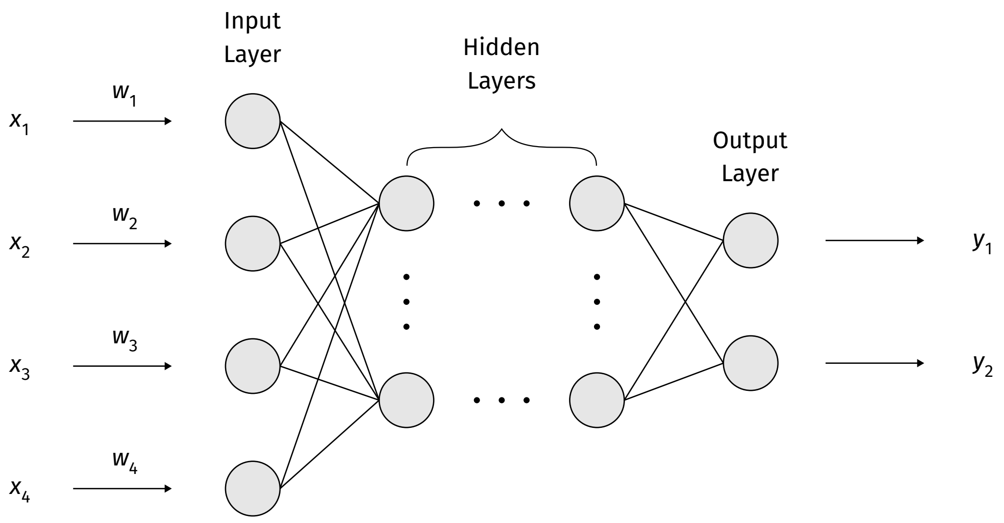
Source: preapared by the author.

Multi-layer Perceptron

A MLP is a important class of ANN. It consists of a set of sensorial units that compose the *input layer*; one or more *hidden layers*; and an *output layer*. The input signal propagates forward through the network, layer by layer. They are used to solving complex problems, with the supervised training with the *error back-propagation* algorithm.

The learning by back-propagation consists of two steps through the layers of the perceptron: a forward pass (propagation) and a backward pass (back-propagation). In the forward pass, an input vector is applied to the sensorial nodes of the network and it propagates through the network, layer by layer; in this step the weights are fixed. During the backward pass, the weights are fit accordingly through a loss function (see **??**). This error signal is propagated through the network in the opposite direction of the synaptic connections. The weights are adjusted to make that the network output gets closer of the wanted output. **??** represents a MLP.

Figure 4 – Visual Representation of a MLP. Let x_i be the input vector and y_i the output vector, $i \in \mathbb{N}$. The weights w_{x_i} are specific for each input data. Input and output data can have multiple entries.



Source: prepared by the author.

The three main features of the MLP are:

- Non-linear activation function. It is commonly used a smooth non-linear activation function, like rectifier function (ReLU):

$$y_j = \begin{cases} x, & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

  Or sigmoid function:

$$y_j = \frac{1}{1 + \exp(-v_j)} \tag{2.6}$$

  where v_j is the weighted sum of all input layers with their respective weights of the j neuron; and y_j is the output of the neuron.

- Hidden layers. They allow the network to learn complex tasks, extracting progressively the most significantly features of the input vector.

- Connectivity. High level of connectivity, determined by the network synapses.

These features, plus the ability to learn from the experience of the training, that makes the MLP so powerful, however, they are also responsible for its deficiency. First, the non-linearity and the high connectivity makes hard the theoretical analysis of an MLP; second, the hidden layers make it more difficult to visualize the learning processing. The learning process is harder because the search must be conducted in a much bigger space of possible functions.

2.2.3 Loss Function

The *loss function*, also called *cost function* or *error function*, is the one used measure the error between the predicted output of an algorithm and the real target output. There are several loss functions suitable to different kind of situation. For each distributed data there is one that fits better. Many kinds of them are available and must be analyzed the most proper one to each case. The choice of what loss function should be picked will depend on not only the data and its pattern, but also the computational processing and the cost attached to it.

Regression

Although regression problems do not require deep learning to create a satisfactory model, naturally it is possible to do so. For regression problems, common loss functions adopted are the MAE and MSE (**bussab2017**):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.7}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.8}$$

where n is the sample size; y_i is the predicted output; and $\hat{y}_i$ is the real target.

?? shows a linear data and how the domain of the loss function is obtained for a linear regression.

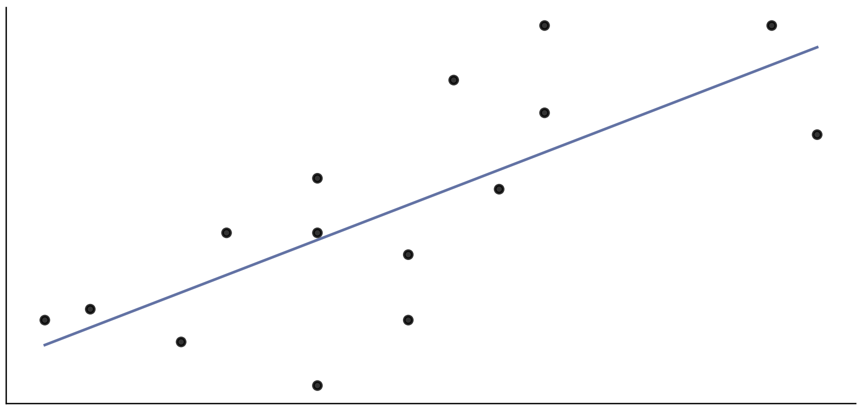
2.2.4 Optimizer

The optimizer is an algorithm that updates the model in response to the output of the loss function, that is, it aids to minimize the loss function. As the loss function minimizes, the model is getting closer to the target values and, hence, closer to the real pattern.

Gradient Descent

The *gradient descent* is one of the main algorithm (**nesterov2004**) that optimizes the model and many important ones are based on it, like the SGD. The goal is to get the minimum,

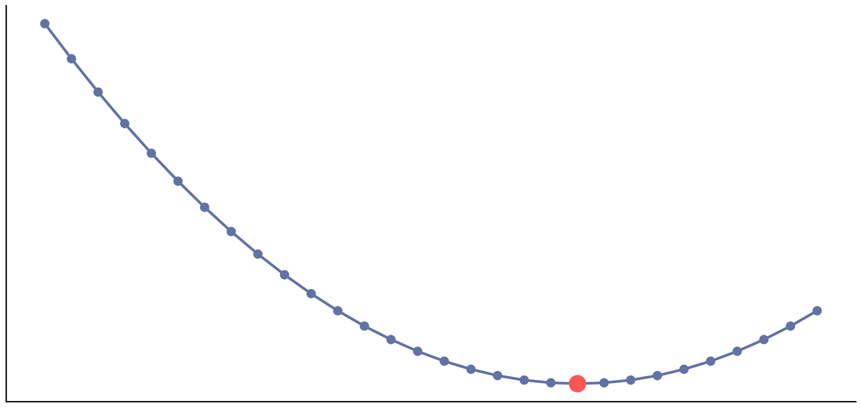
Figure 5 – Loss Function for Linear Regression. The loss function take all the distances between the predicted (dots) and the target value (curve) to verify if the model is in the right path. The lower the distance, the better the model. The *y*-axis represents the output data and the *x*-axis represents the input data.



Source: prepared by the author.

as the error (loss) between the predicted and the target data is null. This would mean that the model fits to the pattern of the data.

Figure 6 – Gradient Descent Process. Loss function can be represented in a two-axes plan. Depending on the data, it is not possible to represent graphically due to its multi dimension. Each point represents the learning step. When the gradient descent reaches the minimum of the loss function, it means that the model may be accurate. Note that the gradient descent can reach a local minimum of the function and not the global minimum necessarily. The *y*-axis represents the loss function and the *x*-axis represents the weight values. Red dot represents the minimum value of the loss function.



Source: prepared by the author.

The gradient descent is a powerful algorithm that reduces the loss function, minimizing the error between the predicted value and the target value.

Since the gradient of a function gives the direction of the steepest ascent of a function, and it is orthogonal to the surface at a determined point, it seems reasonable that moving in the perpendicular direction gives the maximum increase of the function (**stewart2016**). On the other hand, the negative of the gradient may be used to find the opposite, that is, the steepest descent of the function, or the minimum decrease. If the steps given to the direction of the negative gradient of the function are small, there is a good chance to get minimum value of the function. However, if the steps are too long, the chance to pass by the minimum value is high (**nielsen2015**). These steps are called *learning rate* and should be chosen wisely.

This way, let $\mathbf{x}$ be the entry vector with the predicted data and L the loss function adopted for some deep learning model, and ϵ the learning rate, the gradient descent is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \epsilon \nabla L(\mathbf{x}_t) \tag{2.9}$$

In determined cases, it is possible to avoid running the iterative algorithm and just go directly to the critical point by solving $\nabla L(\mathbf{x}_t) = 0$ for $\mathbf{x}$.

Stochastic Gradient Descent

As seen, gradient descent is a powerful tool to minimize the loss function, however, for large data, the cost of operation is very high, and its use is not feasible. The main idea of SGD is that the gradient is an expectation. Later, the data is divided in subsets, also called *mini-batch* and then the gradient is performed over them. The mini-batch size is chosen to be a relatively small numbers of examples. The data inside each subset may be considered redundant, that is why it uses one single value of the subset to compute the gradient descent. This way, the process is considerable better for computational resources.

The SGD can be written as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\epsilon}{m} \sum_{i=1}^{m} \nabla L(\mathbf{x}_t; p^{(i)}, q^{(i)}) \tag{2.10}$$

where m is the mini-batch size; and $\nabla L(\mathbf{x}; p^{(i)}, q^{(i)})$ is the gradient of the loss function with respect to the parameter vector $\mathbf{x}$ for the i^{th} example $(p^{(i)}, q^{(i)})$ in the mini-batch.

Yet, nowadays, with the amount of data, many techniques are still applied in SGD as creating an automatic adaptive learning rates which achieve the optimal rate of convergence (**darken1991**) and the momentum technique to improve it (**sutskever2013**).

Adam

Adam, shown in **??**, is an algorithm developed by **kingma2017<empty citation>** for first-order gradient-based optimization of stochastic objective functions, like the loss function, as seen in the **??**. It is based on adaptive estimates of low-order moments and computationally efficient, requiring little computational memory. Adam is a strategical choice when using large data or parameters and with very noisy/sparse gradients (**kingma2017**).

Algorithm 1 – Adam Algorithm. Good default setting are $\alpha = 0.001,\ \beta_1 = 0.9,\ \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Operations on vectors are element-wise.

```
Require: $\alpha$: step size
Require: $\beta_1,\ \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates
Require: $f(\theta)$: loss function
Require: $\theta_0$: initial parameter
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $t \leftarrow 0$
  while $\theta_t$ not converged do
      $t \leftarrow t + 1$
      $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
      $m_t \leftarrow b_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
      $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
      $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$
      $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$
      $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$
  end while
        return $\theta_t$
```

3 METHODOLOGY

There are two main AI perspectives and according to **winston1992<empty citation>**:

> *The engineering goal of artificial intelligence is to solve real-world problems using artificial intelligence as an armamentarium of ideas about representing knowledge, using knowledge and assembly systems. The scientific goal of artificial intelligence is to which ideas about representing knowledge, using knowledge and assembly systems explains various sorts of intelligence.*

The current objective is to approach AI from an engineering perspective. Hence, the code implementation will be pragmatic, and the lines of code will not be fully explained. The framework methods will not be elaborated on either, but their documentation is reasonably comprehensive for individuals with prior programming knowledge, especially in Python and MATLAB. Links to documentation will be provided whenever possible.

3.1 Softwares

3.1.1 MATLAB

The standard in the engineering industry, MATLAB is a powerful toolbox that can be used to several activities. Since applications in fields like medicine and biology (**demirkaya2009**), like image processing, until the most complex problems in engineering (**bansal**), that involve matrix operation and control simulation, MATLAB has lots of tools that aid to solve problems.

geronel2023<empty citation> used MATLAB to develop their algorithm, hence the data generation will be done with it.

3.1.2 PyTorch

A framework is a group of libraries for a programming language that implements a lot of tools to facilitate some tasks. The programming language used for the ANN training is Python. There is a lot of deep learning frameworks available and the most popular ones are TensorFlow (**abadi2016**) and PyTorch (**paszke2019**). While the first one was developed by Google and released in 2015 the second one was developed by Meta (Facebook), although it is now under the Linux Foundation umbrella, and released in 2016, being both open-source. Many companies, like Uber (**goodman2017**) and Tesla (**pytorch2019**), use PyTorch in their AI team, while companies like Coca-Cola uses TensorFlow (**tensorflow2018**). This means that both are trustful frameworks to rely upon their built-in functions. PyTorch will be used to develop all the ANN.

3.2 Data Generation

Since the script of **geronel2023<empty citation>** provides the control torque ($_\eta$) related to the state-space ($\mathbf{x}_s$) through dynamic and control equations. The ANN goal will take $\mathbf{x}_s$ as the input vector and predict the $_\eta$ vector as output. Modifications in the script are minimal the necessary to generate a thousand trajectories in a loop and adding random noise to each one. The time is a discrete vector with 200 s and step 0.01, therefore the time vector has 1×20000 dimension.

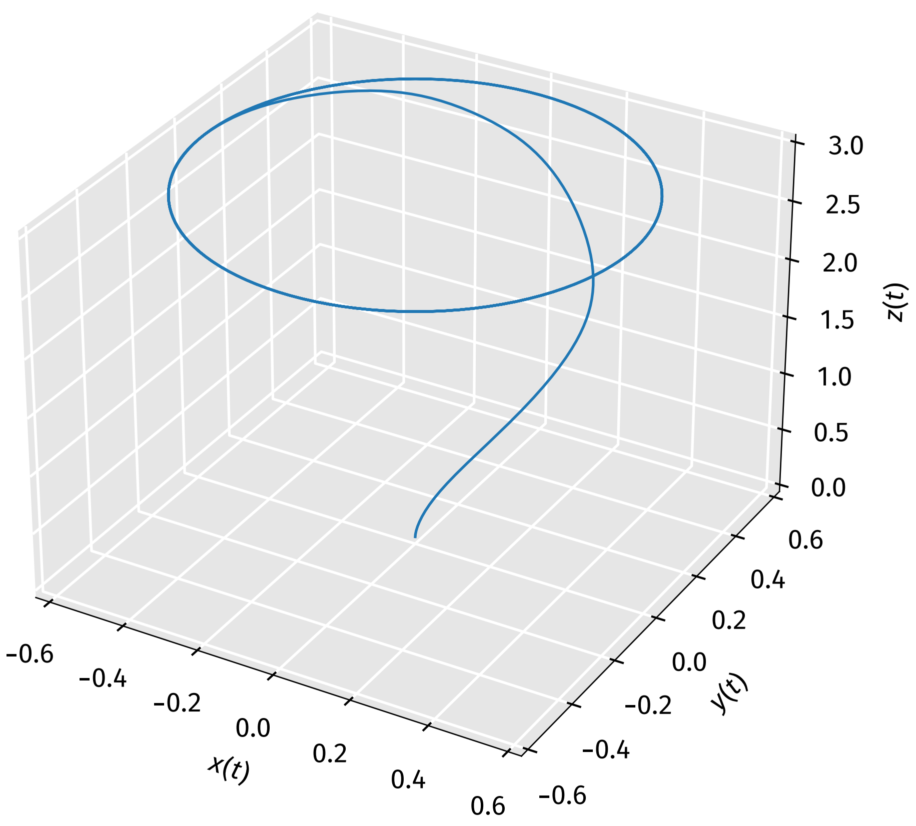
The output vector $\mathbf{T}$ has 20000×4 dimension the and the input vector $\mathbf{X}_s$ has 20000×12 dimension, where each line represents specific point in time:

$$\mathbf{T} = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.1}$$

$$\mathbf{X}_s = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.2}$$

Circular trajectory was arbitrary selected as shown in **??**. Then, it was generated 1000

Figure 7 – Trajectory of the UAV. Circular trajectory was chosen and the thousand other ones generated are only variations with noise added.



Source: adapted from **geronel2023<empty citation>**

different trajectories with random noise for each trajectory. Input and output vector were stored in a MATLAB variable and exported through the `.mat` extension to be used inside the Python environment using *scipy.io.loadmat* SciPy method.

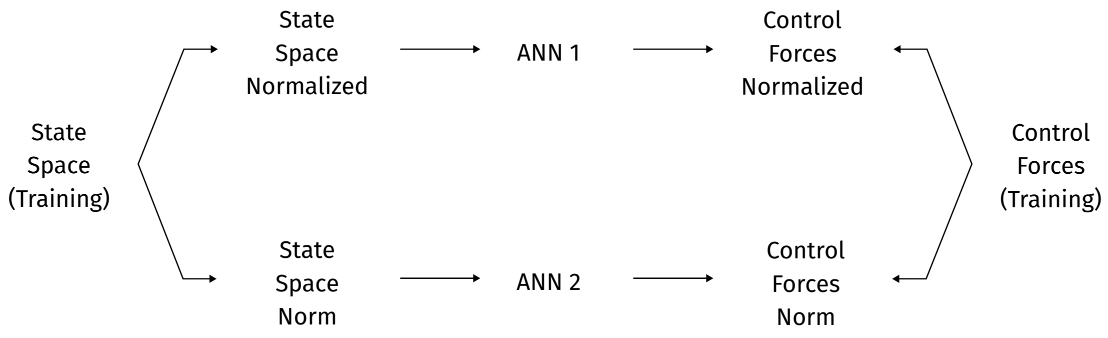
3.3 Algorithm Overview

The first approach for the ANN is to use the raw data, both for input and output and do the training. Even though it works, it does not give the proper result, due to the scale for each force and state-space columns. Therefore, the preprocessing of the data is mandatory to get the best results. Then, all input and output data were normalized in order to get them all standardized.

The normalization is in L2 form, from the *sklearn.preprocessing.normalize* method, applied in each matrix column. From the trained ANN, all input data should be normalized and naturally the output also will be normalized. However, the control forces (output data) can not be normalized to be useful, but there is no "denormalized" correspondent matrix to the output data from the ANN.

To solve this problem, a second ANN was created to be able to denormalize the output data. When preprocessing the data, as the normalization is done, both norms of the input and output data are stored and the second ANN is made from them. The **??** show how the data and the ANNs are related for the training. When the training and the validation is done, the ready-to-use model will perform as shown in the **??** scheme.

Figure 8 – Data and ANN relation. The training data in the extremes are the ones generated by the white box parametric model.

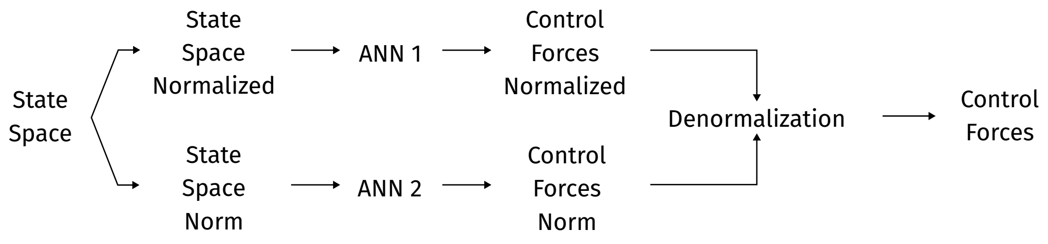


Source: prepared by the author.

3.4 Neural Network Modeling

The ANN 1 is responsible for, from the normalized state space, to return the normalized control forces. The problem is considered as a regression problem for both ANNs. The **??**

Figure 9 – Model in production. The scheme shows how the process returns the control forces from the state space.



Source: prepared by the author.

shows the functional form of the ANN 1. Input and output data are all matrices.

$$f(x, y, \dots, \dot{\theta}, \dot{\psi}) = \langle U_1, U_2, U_3, U_4 \rangle \tag{3.3a}$$

$$f(\mathbf{X}_s) = \mathbf{T} \tag{3.3b}$$

The ANN 2 is responsible for, from the raw data, to get the norms from each input and output data of the ANN 1. The problem is considered as a regression problem. Input and output data are all vectors. Both ANNs have similar parameters, being the main difference the input and output layer size. Characteristics of them are provided in the **??**.

Table 1 – Parameters of the neural networks and their training

Parameters	ANN 1	ANN 2
Input layer neurons	12	1
Output layer neurons	4	1
Hidden layers neurons	128	128
Hidden layers	8	8
Activation function	ReLU	ReLU
Loss function	MSE	MSE
Optimizer	Adam	Adam
Batch size	1	1
Train/test split	0.8/0.2	0.8/0.2
Epochs	500	500

Source: prepared by the author.

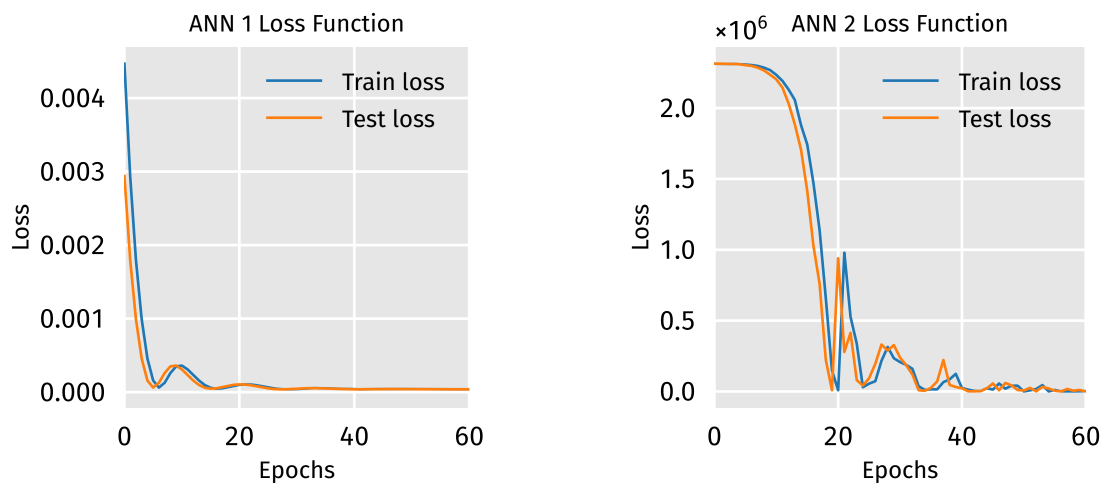
4 RESULTS AND DISCUSSION

This chapter introduces and presents the results obtained with ANNs modeling. Initially, the loss functions are presented with a discussion about them. Subsequently, the forces are analyzed, and suggestions for improvement are provided

4.1 Loss functions

Since there are two ANN, two loss functions were obtained with the training. They are shown in the **??**.

Figure 10 – Loss Functions Behavior. The total number of epochs is 500 and was omitted because they were approximating a straight line.



Source: prepared by the author.

The expected behavior of the loss function is its decay as the epochs increases. **??** shows the loss function decreasing along the epochs for both ANN and both for the train and test data. The data used for training is relatively small (1000) and after splitting the data, for the training, indeed, the ANN only could see the pattern of 800 trajectories. One of the strategies, therefore, was to use a high number of epochs, even though the loss function had already converged. For acknowledgment, the convergence began with the first 10% of the total number of epochs for both ANNs. This way, it was necessary to verify if the ANN was not going to overfit the data. However, as the green curve shows, the test validation also converged, making it much more likely that no overfitting occurred.

4.2 UAV Forces

The normalized forces from neural network ANN 1 go through a comparison with the white box script within the corresponding state space. This analysis is crucial because it reveals the consistency of the model, aiming to show insights into the accuracy of ANN 1 predictions. The results are presented in **??**. The denormalized forces obtained by joining the results of both ANN are also presented in **??**, with the same goal.

The ANN 1 is supposed to determine patterns for the UAV forces based on its state space. To achieve a better recognition of these patterns, normalization was performed to prevent the scale of the values from affecting the ANN, since the input values have different units of measurement among themselves and also differ from the output values.

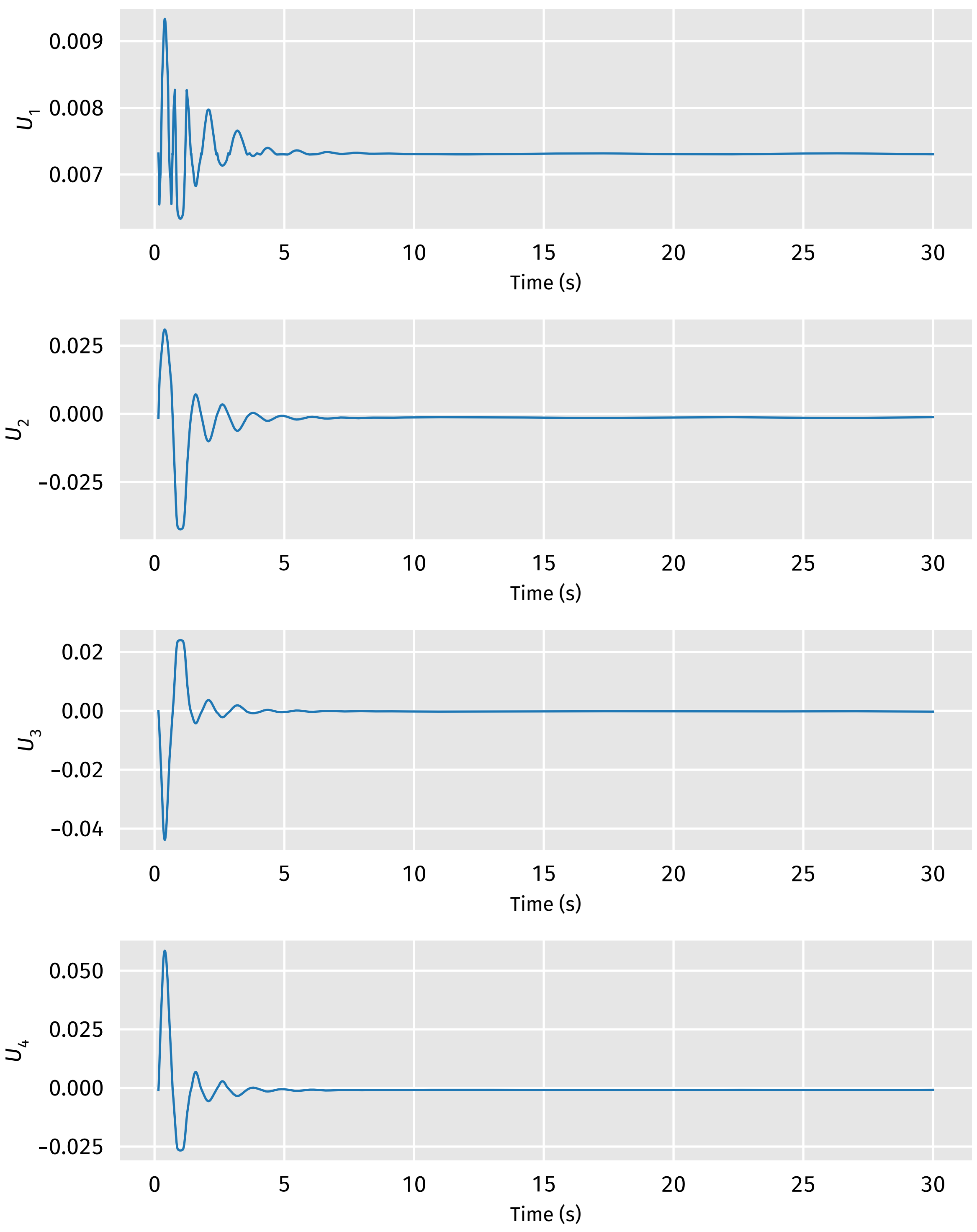
The recognized pattern will serve as the main basis for denormalization when the algorithm is applied. As can be seen in **??**, for U_2 and U_4, although the maximum and minimum local values do not match, the ANN could determine the pattern of the forces, following the raise and the fall of the real value. After stabilization, that is, when the UAV stops increasing in the $z(t)$ direction, both curves overlap. For U_3, it is noticed that after stabilization, the curves also overlap. However, before that, the decay pattern was recognized, but with some caveat. At the global minimum value of the real output, the ANN could not follow the pattern as it should; in fact, it went in the opposite direction. This can affect directly in the UAV trajectory and compromise the flight. For U_1, on the other hand, the recognition was poor, and for the UAV takeoff, the ANN could not determine the force to keep the flight going. After stabilization, although it improved compared to the beginning of the trajectory, the oscillations are high enough to compromise the flight.

As can be seen in **??**, the curves are practically identical to the normalized ones, but with different order of magnitude. This shows that ANN 2 had great performance in recognizing the norm patterns, especially considering the stabilization region. For U_2, U_3 and U_4 the curves are overlapping, and the order of magnitude is very low, indicating good accuracy. For U_1, as happened with ANN 1, it did not perform as well, and the difference is bigger from the real value compared to the other ones. Yet, after stabilization, the ANN was precise, but not exact. If this pattern continues for all predicted forces, it can be adjusted with a factor to correct this. Let α be this *correction factor*. If it is observed that the predicted values from the ANN consistently underestimate the original values by one unit, as occurred in U_1, then it can be assumed $\alpha = 1$.

The main problem of the ANN is the quantity of the samples. For a ANN, it is necessary a large amount of samples, at least hundreds of thousands of them. The data generation was limited by some factors: (i) computational resources, which were not sufficient to generate more than two thousand matrices with twenty thousand lines; (ii) the MATLAB version used, which could not generate files over 2 GB to export the files generated to the Python environment. These situations can be improved with powerful data processing and by using newer MATLAB versions.

This initial approach used an MLP, which is a simple ANN architecture. Although efficient for regression problems, the limited quantity of samples hindered its capability to determine the patterns more efficiently. Therefore, exploring alternative ANNs architectures

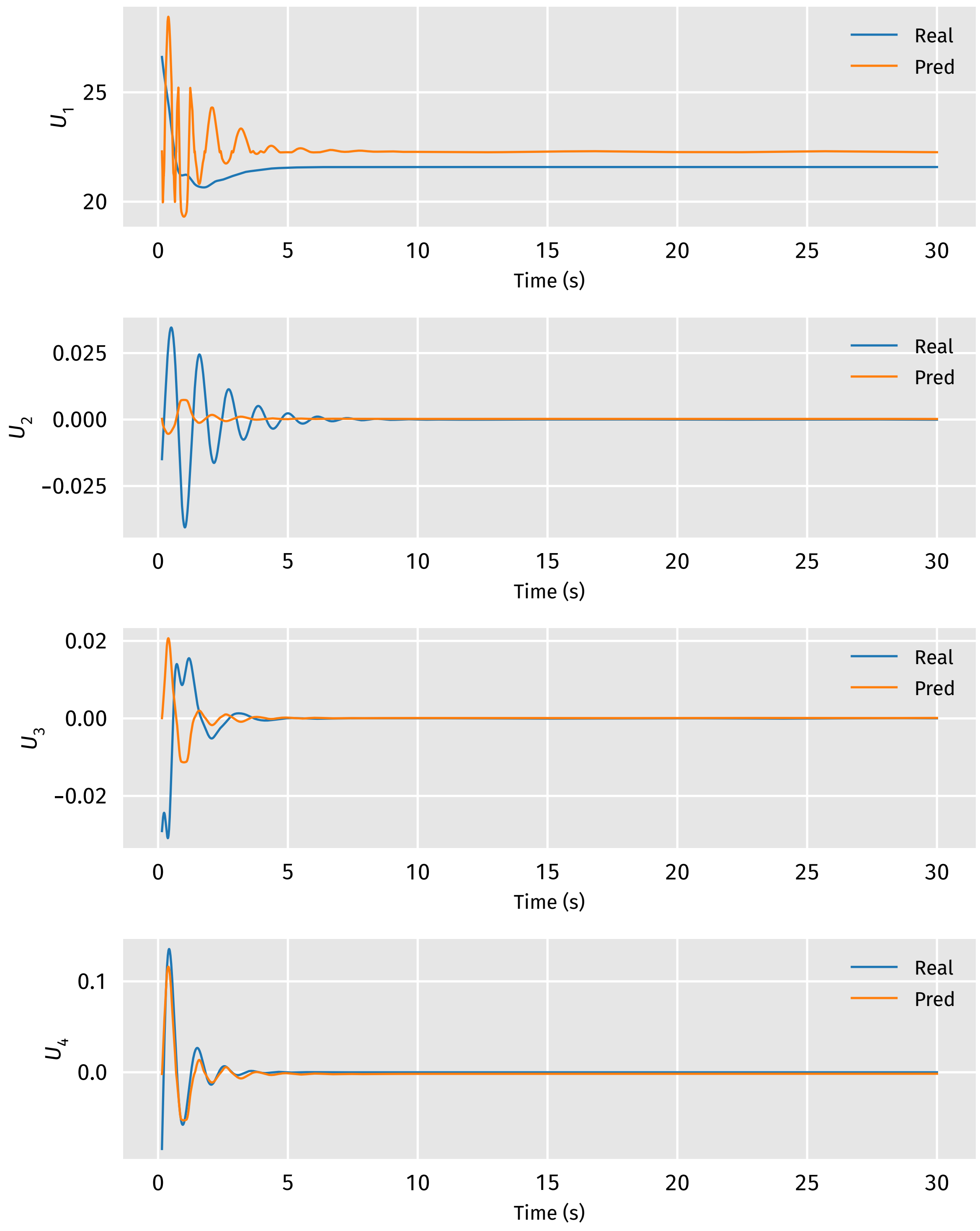
Figure 11 – Normalized Forces from ANN 1. Comparison is made from the results predicted from the neural network and the normalized preprocessing data, both for input and output data.



Source: prepared by the author.

and approaches that can use less data and get more results is an alternative. Since the methodology used is in the AI field, then machine learning techniques can also be a good approach, due to the data nature that is structured and well-defined matrices.

Figure 12 – Denormalized Forces from ANN 1 and ANN 2 Combination. Comparison is made from the results predicted from the combination of both neural network and the raw data got from the white box parametric model.



Source: prepared by the author.

One alternative strategy is to split the trajectory and use different ANN for each part or to use different ANN to each force separately. This can reduce bias from the different trajectories or forces. The takeoff pattern is very different of when the UAV reaches the desired altitude. Therefore, splitting one ANN for takeoff and another for the already defined trajectory may be a good approach.

Finally, this is the initial attempt utilizing ANN. Therefore, it started with the simplest available method, achieving satisfactory results within this scenario. The objective is to begin with a straightforward approach and refine the methodology as necessary. Using pre-trained models for regression is also a feasible alternative, involving adjustments to a few layers and tweaking to this specific problem. As the sample size increases and the ANN becomes more complex, the predictions can become more accurate.

5 CONCLUSION

The use of AI is widespread in various areas, reaching people at different levels. Its application in numerous fields makes it useful to solve problems with the vast amount of data collected nowadays. Using AI to achieve goals is a good strategy. Controlling UAVs is vital because it can reach difficult areas with ease. Keeping them on the desired trajectory is also crucial to guarantee flight stability and ensure that goals are not interrupted by external disturbances. A modern and effective strategy involves the use of ANNs to determine UAV forces and keep them on the right trajectory. As observed, a simple ANN with small samples trained with limited resources was able to predict force patterns and satisfactory results. Therefore, the use of AI with ANN techniques is a good strategy to ensure flight maintenance, and improving the ANN with more samples in data generation can enhance its capability to determine UAV control forces with greater precision.

5.1 Next Steps

Although the developed ANNs yielded satisfactory results, here are some suggestions for future work:

- use more samples to train the ANN;
- try different architectures for each ANN;
- split different segments of the trajectory among different ANNs;
- utilize the forces predicted by the ANNs in the white box script and verify if the flight is executed appropriately;
- use different out-of-scope deep learning techniques and try traditional machine learning techniques for well-structured data.