République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique d'Alger
Département du Génie Industriel

A thesis submitted in partial fulfillement of the requirements
for the industrial engineering degree, option:
Data Science And Artificial Intelligence

Modeling Extruder's Behavior as Multivariate Time Series for Deep Learning-Driven Forecasting

---

Internship Host School: **ICAM Nantes**

Presented By:
Mrs. Amira SOUILAH

Supervised by:
Mr. Hakim FOURAR LAIDI(ENP)
Mr. Jérome ROCHETEAU (ICAM)
Mrs. Hala GHAZI (ICAM)

Publicly defended on September 17, 2023

Board of Examiners

President	Iskander ZOUAGHI	MCA	(ENP)
Examiner.	Salah Eddine TACHI	MCA	(ENP)
Supervisor	Hakim FOURAR LAIDI	MCA	(ENP)

ENP 2023

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique d'Alger
Département du Génie Industriel

A thesis submitted in partial fulfillement of the requirements
for the industrial engineering degree, option:
Data Science And Artificial Intelligence

Modeling Extruder's Behavior as Multivariate Time Series for Deep Learning-Driven Forecasting

---

Internship Host School: **ICAM Nantes**

Supervised by:
Mr. Hakim FOURAR LAIDI(ENP)
Mr. Jérome ROCHETEAU (ICAM)
Mrs. Hala GHAZI (ICAM)

Presented By:
Mrs. Amira SOUILAH

Publicly defended on September 17, 2023

Board of Examiners

President	Iskander ZOUAGHI	MCA	(ENP)
Examiner	Salah Eddine TACHI	MCA	(ENP)
Supervisor	Hakim FOURAR LAIDI	MCA	(ENP)

ENP 2023

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique d'Alger
Département du Génie Industriel

Mémoire de projet de fin d'étude en vue de l'obtention du diplôme d'ingénieur d'Etat en génie industriel, option:
Data Science et Intelligence Artificielle

Modélisation du Comportement de l'Extrudeuse en Série Temporelle Multivariée pour la Prédiction grâce à l'Apprentissage Profond

---

Ecole d'acceuil: **ICAM Nantes**

Encadré par:
M. Hakim FOURAR LAIDI(ENP)
M. Jérome ROCHETEAU (ICAM)
Mme. Hala GHAZI (ICAM)

Presenté par:
Mme. Amira SOUILAH

Présenté et soutenu publiquement le 17 September, 2023

Composition du Jury

Président	Iskander ZOUAGHI	MCA	(ENP)
Examinateur	Salah Eddine TACHI	MCA	(ENP)
Promoteur	Hakim FOURAR LAIDI	MCA	(ENP)

ENP 2023

إلهي أنا الصغير الذي ربّيْته، فلك الحمد
و أنا الضعيف الذي قويته ، فلك الحمد
و أنا الفقير الذي أغنيته ، فلك الحمد
و أنا الجاهل الذي علمته ، فلك الحمد
فلك كل الحمد و الشكر يا الله ...

DEDICATION

I would like to dedicate this work to my family. Without their unconditional love, constant support, and encouragement, I could never have accomplished this memorable experience. I have always been someone who feared stepping out of their comfort zone and trying new things due to a fear of failure. However, thanks to their encouragement and positive influence, I was able to overcome this fear and embark on this adventure.

PAPA, I am infinitely grateful for everything you have done for me. You have been a pillar of strength and wisdom, always there to guide and support me.

MAMA, thank you for your generosity and love. I am deeply grateful for all the times you took care of me through your delicious dishes. Each bite was a true explosion of flavors, a testament to your culinary talent and unconditional love.

Kenza, thank you for sharing your valuable advice with me; you are a true treasure in my life.

Keltoum, thank you for helping me when I was on the verge of giving up; your presence gave me hope.

Fatma Zohra, thank you for your messages that brightened my darkest days.

Rokia, thank you for your unwavering support and comforting presence.

Zineb, thank you for taking the time to talk to me, listen to me, and share your own experiences, which made me feel less isolated and more confident in this unfamiliar environment.

Mohamed Ali, your presence in my life is trully a blessing, thank you for being the coolest brother and friend anyone could ever ask for.

Manar,my beautiful little sister, thank you for the way you affectionately call me "BIBO" every time you hear my voice. Your attitude brings so much joy and love into my life, and I cherish it dearly.

Lydia, my best friend, thank you for supporting me throughout these 7 years, always being there for me, no matter the circumstances. Your presence and unwavering support have been a source of strength and inspiration.

Wafaa, my cousin and childhood friend, thank you for your messages that bring a glimmer of hope and positivity in the difficult moments I have gone through.

To all of you, I want to say that "I love you GUYS". Thank you for encouraging and supporting my sometimes strange behavior. Your unwavering love and support have been crucial factors in the realization of this work.

Amira SOUILAH

ACKNOWLEDGEMENT

Words sometimes seem insufficient to translate the depth of my feelings and the gratitude I feel towards the people who have provided me with their invaluable support in the completion of this work.

First and foremost, I would like to express my sincere gratitude to you,**Mr. Iskander Zouaghi**, for believing in me and offering me an opportunity that I never thought possible. Your confidence in my abilities has deeply touched me and has been an invaluable source of inspiration.

I would also like to extend my sincere thanks to my supervisors: **Mr. Jérôme Rocheteau** and **Mrs. Hala Ghazi**, for their constant guidance and encouragement throughout my journey. Without your unwavering support, I could never have accomplished this work with such success. Your benevolent presence has been a true source of motivation for me.

I sincerely thank **Mr. Hakim FOURAR LAIDI** for accepting to mentor and support me throughout the internship. I want to express my utmost admiration and gratitude to him.

Lastly, I extend my profound gratitude to the **jury members** for dedicating their time and energy to examine and evaluate my work.

الملخص

مفهوم التوأم الرقمي (DT) يلعب دورًا حاسمًا في مشروع البحث (Recyplast-Demo) ، الذي يهدف إلى دراسة عملية الاستخراج للبلاستيك المعاد تدويره. يشير التوأم الرقمي إلى إنشاء تمثيل افتراضي وديناميي لنظام حقيقي، وفي هذه الحالة، آلة الاستخراج. يشمل العملية إدخال حبيبات البلاستيك إلى الآلة، التي تتقدم بعد ذلك بواسطة مسمار محرك داخل برميل مُسخن. تذوب الحبيبات وتتحول إلى بلاستيك مذاب أثناء مرورها من خلال قالب تشكيل، مكتسبة في النهاية شكلها النهائي.

حاليًا، يتم تعيين سرعة المحرك وإعدادات درجة الحرارة للمرآة بقيم ثابتة طوال عمليات الاستخراج. يُفترض أن خصائص حبيبات البلاستيك تبقى مستقرة، مما يؤدي إلى تحقيق نقاط التعيين المستقرة بمجرد تعديلها من قبل مشغلي الآلة. يراقب هؤلاء المشغلين بشكل رئيسي ضغط إخراج المواد ودرجة الحرارة وعزم المحرك لضمان أداء الاستخراج الأمثل.

ومع ذلك، يتسبب استخدام البلاستيك المعاد تدويره في تقديم تغييرات في خصائص المواد، مما يشكل تحديًا لافتراض الاستقرار والنقاط الثابتة. لذا يصبح من الضروري تكييف سرعة المحرك ودرجة حرارة المرآة بشكل ديناميي استنادًا إلى عوامل مثل ضغط إخراج المواد ودرجة الحرارة وعزم المحرك. هذا هو المكان الذي يصبح فيه التوأم الرقمي قيمة، حيث يوفر رؤى في الوقت الحقيقي حول سلوك الجهاز الاستخراج البلاستيكي. من خلال فهم سلوك الجهاز الاستخراج في الوقت الحقيقي، يمكننا الحفاظ على جودة المنتج مستقرة على الرغم من التغيرات في خصائص المواد الداخلة. بالإضافة إلى ذلك، يمكن للتوأم الرقمي تمكين تقييم دقيق لأداء الجهاز الاستخراج تحت ظروف وسياقات مختلفة.

في هذه الدراسة، أجرينا تحليلاً لسلوك الجهاز الاستخراج، معاملين القياسات على أنها بيانات سلاسل زمنية متعددة المتغيرات. هذا التحليل سمح لنا بفهم أعمق لسلوك الجهاز الاستخراج وتطوير نموذج ذكي لتوقع عملية الاستخراج. وبناءً على هذا النموذج، يمكن تطوير نظام توصية يمكنه تقديم رؤى قيمة واقتراحات في الأعمال المستقبلية.

الكلمات الرئيسية : التوائم الرقمية (DT) ، سلاسل زمنية متعددة المتغيرات (MTS) ، الذاكرة طويلة الأمد ـ قصيرة الأمد (LSTM) ، الشبكة العصبية العاملة مع الإعادة (RNN) ، نموذج الترميز والفك (E_D) ، متجه انحداري ذاتي (VAR) .

Résumé

Le concept de jumeau numérique (JN) joue un rôle crucial dans le projet de recherche Recyplast-Demo, visant à étudier le processus d'extrusion du plastique recyclé. Un jumeau numérique fait référence à la création d'une représentation virtuelle et dynamique d'un système réel, dans ce cas, la machine d'extrusion. Le processus implique l'alimentation en pelets de plastique dans la machine, qui sont ensuite propulsées en avant par une vis motorisée à l'intérieur d'un cylindre chauffé. Les pelets fondent et se transforment en plastique fondu en passant à travers une filière de formage, acquérant ainsi leur forme finale. Actuellement, la vitesse du moteur et les réglages de température du chauffage sont fixés à des valeurs constantes tout au long des opérations d'extrusion. On suppose que les propriétés des pelets de plastique restent stables, conduisant à des valeurs de consigne cohérentes une fois ajustées par les opérateurs de la machine. Ces opérateurs surveillent principalement la pression de sortie du matériau, la température et le couple moteur pour garantir des performances d'extrusion optimales. Cependant, l'utilisation de plastique recyclé introduit une variabilité dans les propriétés du matériau, remettant en question l'hypothèse de stabilité et de valeurs de consigne constantes. Il devient donc crucial d'adapter dynamiquement la vitesse du moteur et la température du chauffage en fonction de facteurs tels que la pression de sortie du matériau, la température et le couple moteur. C'est là que le jumeau numérique devient précieux, car il fournit des informations en temps réel sur le comportement de l'extrudeuse en plastique. En comprenant le comportement en temps réel de l'extrudeuse, nous pourrions maintenir une qualité de produit constante malgré les variations dans les propriétés du matériau d'entrée. De plus, le jumeau numérique permet une évaluation précise des performances de l'extrudeuse dans différentes conditions et contextes. Dans cette étude, nous avons mené une analyse du comportement de l'extrudeuse, traitant les mesures comme des données de séries temporelles multivariées. Cette analyse nous a permis d'approfondir notre compréhension du comportement de l'extrudeuse et de développer un modèle intelligent pour la prévision du processus d'extrusion. De plus, sur la base de ce modèle, un système de recommandation peut être développé pour fournir des informations précieuses et des suggestions dans les futurs travaux.

Mot-clés: Jumeaux Numériques, Séries Temporelles Multivariées, Cellule De Longue Mémoire à Court Terme, Réseau De Neuronnes Récurent, Modèle Encodeur-Décodeur, Modèle De Vecteur Autoregressif

Abstract

The concept of a digital twin (DT) plays a crucial role in the Recyplast-Demo research project, aiming to study the extrusion process of recycled plastic. A digital twin refers to the creation of a virtual and dynamic representation of a real system, in this case, the extrusion machine. The process involves feeding plastic pellets into the machine, which are then propelled forward by a motor-driven screw inside a heated barrel. The pellets melt and transform into molten plastic as they pass through a shaping die, ultimately acquiring their final form.

Currently, the motor speed and heater temperature settings are set to constant values throughout the extrusion operations. It is assumed that the properties of the plastic pellets remain stable, leading to consistent setpoints once adjusted by machine operators. Those operators mainly monitor material output pressure, temperature, and motor torque to ensure optimal extrusion performance.

However, the use of recycled plastic introduces variability in material properties, challenging the assumption of stability and constant setpoints. Therefore, it becomes crucial to dynamically adapt the motor speed and heater temperature based on factors such as material output pressure, temperature, and motor torque. This is where the digital twin becomes valuable, as it provides real-time insights into the behavior of the plastic extruder.

By understanding the extruder's real-time behavior, we can maintain consistent product quality despite variations in input material properties. Additionally, the digital twin enables accurate evaluation of the extruder's performance under different conditions and contexts.

In this study, we have conducted an analysis of the extruder's behavior, treating the measurements as multivariate time series data. This analysis allowed us to gain a deeper understanding of the extruder's behavior and develop an intelligent model for forecasting the extrusion process. Moreover, based on this model, a recommendation system can be developed to provide valuable insights and suggestions in the next works.

Keywords: Digital Twins, Multivariate Time Series, Long-Short Term Memory, Reccurent Neural Network, Encoder Decoder Model, Vector AutoRegression Model.

Contents

List of Figures

List of Tables

Abbreviations

Acronym	Signification
DT	Digital Twins
TS	Time Series
MTS	Multivariate Time Series
DL	Deep Learning
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
VAR	vector autoregression
ARIMA	AutoRegressive Integrated Moving Average
SARIMA	Seasonal ARIMA
ADF	The Augmented Dickey-Fuller
AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
HQIC	Hannan-Quinn Information Criterion
FPE	Focused Prediction Error
IRF	Impulse Response Function
MSE	Mean Square Error
RMSE	Root Mean Square Error
MAE	Mean Absolute Error
R^2	Coefficient of Determination
NN	Neural Network
Tanh	Tangent Hyperbolic function
$\sigma(x)$	Sigmoid Function
ML	Machine Learning
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
BPPTT	Backpropagation Through Time

Chapter 1

Introduction

A plastic extruder is a sophisticated machine that plays a vital role in manufacturing plastic products by melting and shaping plastic materials. Its efficiency and product quality heavily depend on various critical parameters, including temperature, pressure, motor speed, heaters and motor current [4]. Due to the dynamic nature of these parameters, which exhibit variations over time and mutual interdependencies because Changes in one parameter can affect others, for instance, altering the temperature can influence the viscosity of the plastic material, which in turn can affect pressure and motor current, comprehending and analyzing the behavior of a plastic extruder presents itself as a multifaceted multivariate time series (MTS) challenge.

The central focus of this study revolves around forecasting the forthcoming values of the essential parameters by leveraging historical data. By achieving accurate predictions (minimize the error between the predicted values and the real values), we can optimize the extrusion process, enhance product quality, and enable proactive measures for predictive maintenance.

Several methods are commonly used for forecasting multivariate time series (MTS), each with its strengths and limitations. Statistical time series models, such as AutoRegressive Integrated Moving Average (ARIMA) [5], Seasonal ARIMA (SARIMA) [6], and Vector autoregression (VAR) [7], have been widely employed for their ability to capture the temporal dependencies and interrelationships between variables. However, these methods may struggle to handle large-scale datasets with high dimensionality and complex nonlinear patterns. Additionally, they may not effectively capture long-term dependencies present in the data. On the other hand, advanced deep learning techniques [8], like Long Short-Term Memory (LSTM) [9] and Gated Recurrent Unit (GRU) networks [10], have shown promise in capturing long-term dependencies and complex temporal patterns, making them suitable for forecasting multivariate time series [11, 12]. Nevertheless, these methods often require a substantial amount of data for training, and they may be more susceptible to overfitting, especially in cases with limited historical data. Furthermore, the interpretability of deep learning models may pose challenges, hindering a clear understanding of the underlying relationships among the variables. While both classical and deep learning methods offer valuable insights, striking a balance between accuracy,

scalability, overfitting with the use of regularization techniques [13], interpretability [14] remains an ongoing challenge when forecasting multivariate time series.

This work is structured into five chapters, introduction and a conclusion. Chapter 2 serves as the theoretical foundations, exploring the concepts of multivariate time series forecasting. Chapter 3 delves into related works, examining previous research in the field. Chapter 4 comprises the case study and data exploration. Chapter 5 presents the implementation section, providing a detailed account of the data preprocessing steps while chapter 6 represents results obtained. Finally, the concluding chapter draws insights from a comprehensive comparison, addressing the limitations and findings derived from the preceding chapters.

Chapter 2

Theoretical Foundations

2.1 Introduction

Forecasting multivariate time series is an essential practice in various industries, providing valuable insights for preventing problems and understanding the behavior of industrial machines. Statistical methods such as ARIMA [5] and VAR [7] have proven effective in forecasting multivariate time series. However, the emergence of deep learning models [8, 9] has revolutionized the field, providing unparalleled capabilities in capturing complex dependencies between variables. This chapter describes the basic concepts of multivariate time series forecasting, exploring the key techniques and methodologies employed to predict outcomes, leveraging the power of deep learning models. Additionally, it investigates the evaluation metrics and explores the implementation tools.

2.2 Time Series

2.2.1 Definition

A time series is a sequence of data points that occur in successive order over time. It shows all the data set variables that change over time [15]. Displays all dataset variables [15] that change over time. Examples of time series data include minute-by-minute data from sensors in industrial environments that record variables such as pressure (see figure 2.1).

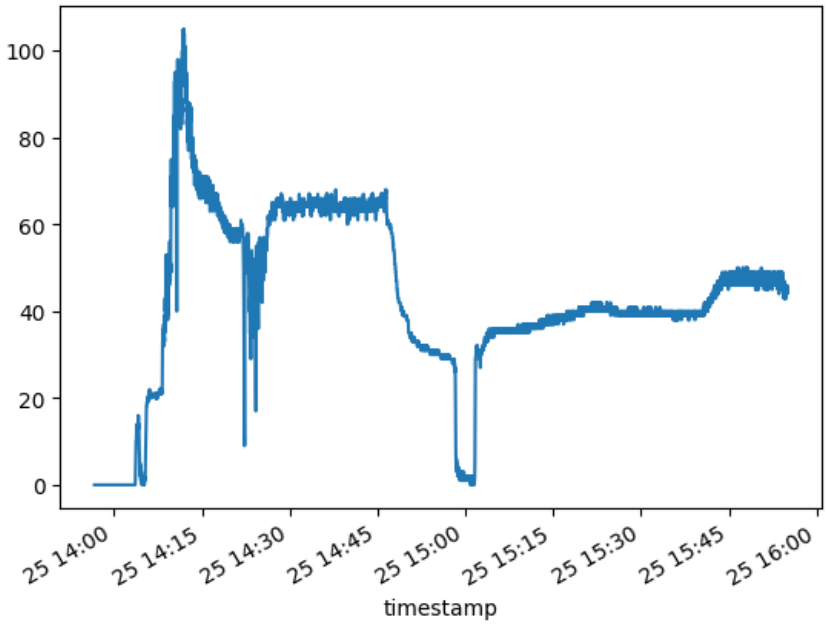


Figure 2.1: Pressure Time Series Data

2.2.2 Time Series Types

1. **Univariate Time Series**: Only one variable is varying over time (see figure 2.1).

2. **Multivariate Time Series**: Several variables are varying as time progresses. Examples of time series data include minute-level data from sensors in an industrial setting, recording variables like pressure and motor speed (see figure2.2).

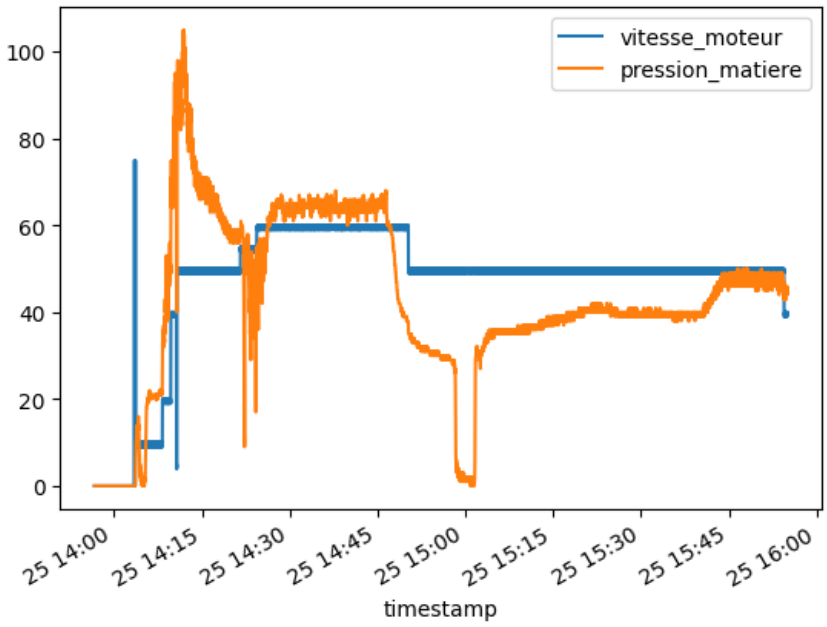
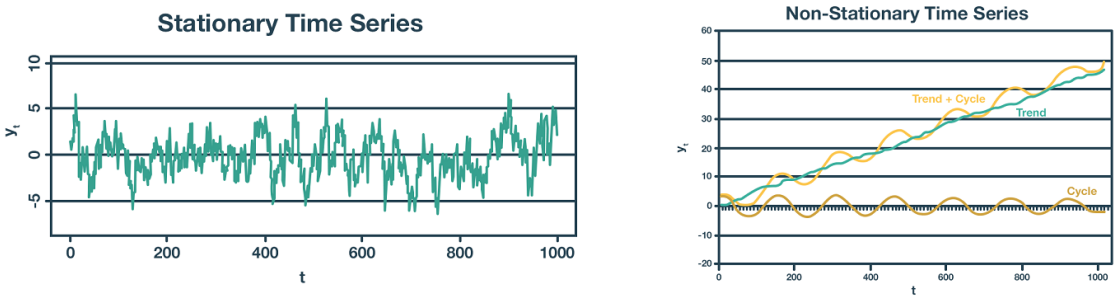


Figure 2.2: A multivariate time series that contains 2 variables.

2.2.3 Stationary Time Series

A stationary time series is a time series whose characteristics mean, variance, and autocorrelation structure do not depend on the observation time of the series. This is in contrast to non-stationary data, where the mean, variance, and covariance of data points change over time. That is, the data show trends, cycles, random walks, or some combination of the three. In general, when it comes to prediction, temporal data are unpredictable and cannot be modeled with [16]. (see figure 2.3)



(a) Stationary-Time-Series (b) Non-Stationary Time Series

Figure 2.3: Stationary And Non-Stationary Time-Series cited in [1]

2.2.4 The Augmented Dickey-Fuller (ADF) Test

There are various statistical techniques to check stationarity, such as: **Augmented Dickey-Fuller Test (ADF)**. The ADF test falls within the category of the unit root test that tests whether a time series is not stationary [17]. In other words the presence of unit root means the time series is non-stationary and defines the null hypothesis, and the alternative hypothesis defines time series as stationary.

$$\Delta y_t = \delta y_{t-1} + u_t \tag{2.1}$$

where:

- y_t represents the variable y at time t.
- Δy_t represents the change in the variable y at time t.
- δ represents the coefficient that represents the effect of the lagged value of y on the change in y at the current time period, t.
- u_t represents a random error.

2.2.5 Differencing To Achieve Stationarity

Differentiation involves calculating the difference between successive observations within a non-stationary time series in order to induce stationarity. As written in the following

formula:

$$y'_t = y_t - y_{t-1} \tag{2.2}$$

Where:

- y'_t : The differenced value of the time series at time.
- y_t : The original value of the time series at time t.
- y_{t-1} : The original value of the time series at the previous time step, which is t-1.

2.3 Multivariate Time Series Forecasting Models

Multivariate Time series forecasting uses historical data of multiple related variables or features gathered over a specific period of time, then a different techniques and algorithms are applied to create a predictive model that offers insights of one or more of these variables of what could happen in the future based on their past values as well as the past values of other associated variables.

Several models are available for the purpose of forecasting multivariate time series. In this section, we will concentrate on delving into two categories: deep learning techniques and the VAR (Vector AutoRegression) model from the realm of statistical methods.

2.3.1 Statistical Methods: Vector AutoRegression Model

1. **Defining VAR Model**: Vector AutoRegression model is a statistical model used to capture relationships between multiple quantities that change over time. Similar to autoregressive models, each variable has an equation that models its change over time. This equation contains the lagged (past) value of the variable, the lagged values of other variables in the model, and the error term [18]. VAR model can be defined as: (see the equation 2.3 cited in [19]).

$$y_t = v + \sum_{j=1}^{p} A_j y_{t-j} + u_t \tag{2.3}$$

   where:

   - y_t : denotes the value of the time series at time t.
   - v : constant term.
   - Aj : represents the coefficient for the lagged value y_{t-j} .
   - u_t : denotes the error term at time t.

2. **Lag Selection**: To determine the appropriate order that minimizes the potential of overfitting while maximizing the model's explanatory power, we rely on: **information criteria** such as the Akaike Information Criterion (AIC), Bayesian

Information Criterion (BIC), Focused Prediction Error (FPE) and the Hannan-Quinn Information Criterion (HQIC) [20].

2.3.2 Deep Learning

Various Deep Learning (DL) Models rooted in the Recurrent Neural Network (RNN) framework, have been suggested to enhance predictive accuracy and mitigate the inter-dependencies within multivariate time series data. In this section, we will lay down the theoretical underpinnings of these artificial neural networks. Simultaneously, we will provide an overview of the relevant literature, which will be expanded upon in Chapter 3.

Neural Network

A Neural Network (NN) is made up of vertically stacked components called Layers. Those layers are composed of small individual units called neurons that mimic the behaviour of a biological neuron neurons that receive signal from other neurons, make some processing, and produces an output [21]. There are three types of layers in a neural network:

1. **Input Layer** receives the raw input data and passes it on to the subsequent layers. Each neuron in the input layer corresponds to a feature or input variable.

2. **Hidden Layers** are intermediate layers that come between the input and output layers. They are responsible for learning and representing complex patterns in the data.

3. **Output Layer** is the final layer of the neural network. It produces the network's predictions or outputs based on the patterns learned in the hidden layers.

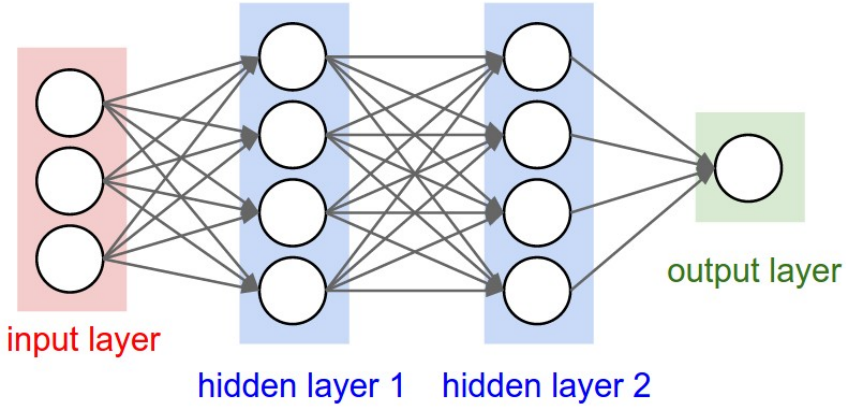


Figure 2.4: Schematic of a neural network

Training a Neural Network

Training an artificial neural network (ANN) involves a series of carefully orchestrated steps that enable the network to learn from data. It begins with seting up the initial condition

from which the ANN will start learning and that by initializing its weights randomly. Subsequently, bias terms (constant values) associated with each neuron are initialized as well. The bias is used to shift the result of activation function towards the positive or negative side [22]. The next step involves around deviding the data into smaller batches for efficient processing. Then, during a forward pass, the network computes outputs based on its current weights and biases, and this output is compared to the expected results, producing a measure of error, or loss. The next critical phase, weight adjustment, relies on backpropagation. Here, gradients are computed to fine-tune the model's weights and biases, ensuring they adapt to the data. This process iterates over multiple batches and epochs until the network converges or completes a preset number of iterations. The dynamic adjustment of weights and biases enables the network to learn and generalize, ultimately empowering it to make precise predictions on new, unseen examples [23].

1. **Backprobagation and Gradient Descent**

   (a) **Backpropagation**

   Backpropagation allows us to readjust our weights to reduce output error. The error is propagated backward during backpropagation from the output to the input layer. This error is then used to calculate the gradient of the cost function with respect to each weight [24] (see figure 2.5).

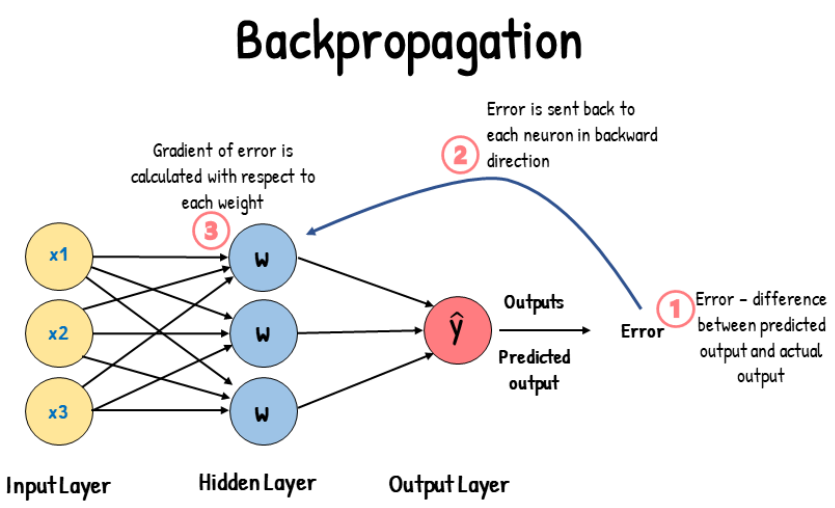


Figure 2.5: Backpropagation Neural Network mentioned in [2]

   (b) **Gradient Descent** is an optimization algorithm used to find the weights that minimize the cost function. We need a gradient and a learning rate to lower the cost function. Gradients help find the direction to reach the minimum point of the cost function. The learning rate helps determine how fast the minimum point is reached. After reaching the minimum point, gradient descent finds a weight equal to the minimum point. [24] (see figure 2.6).

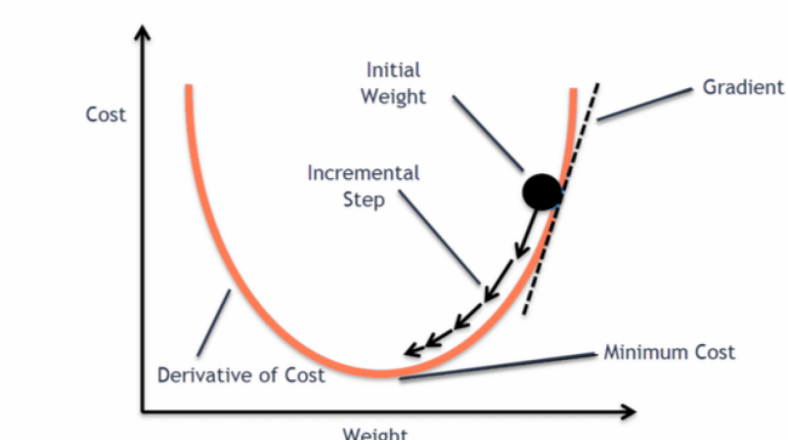


Figure 2.6: Gradient Descent cited in [2]

2. **Activation Function** of a node in an artificial neural network is a function that computes the node's output based on the inputs and the weights of each input [25]. Only two commonly used types of activation functions are presented (Figure 2.7).

   (a) **Sigmoid Function:** This function takes any real value as input and produces values in the range of 0 to 1. It is frequently employed in models that involve predicting probabilities as their output [26], [27]. Mathematically, it can be represented as:

   $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

   (b) **Tanh Function:** Also known as the Tangent Hyperbolic function, it is a mathematically shifted version of the sigmoid function. The key distinction is that the range of the tanh function is from -1 to 1. Both functions share the issue of vanishing gradients [26], [27]. Mathematically, it can be represented as:

   $$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

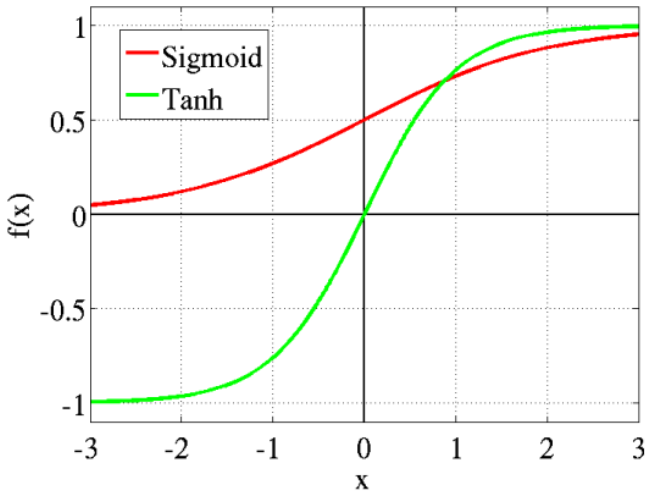


Figure 2.7: Activation Functions In Neural Network

3. **Loss Function**:
   In Machine Learning (ML), a loss function serves as a metric that mesures the degree of accuracy with which your ML model can predict the desired or expected outcome. The lower the values produced by the loss function, the higher the level of performance is achieved. The most commonly used loss function in ML is the **Mean Squared Error (MSE)** which excels at ensuring that the trained model doesn't produce outlier predictions with significant errors.

4. **Adam Optimization Algorithm** Adaptive moment estimation is an algorithm in the gradient descent optimization technique. This method is especially efficient when dealing with large problems containing many data or parameters. It uses less memory and is efficient. Intuitively, this is a combination of the "gradient descent with impulse" algorithm and the "RMSP" algorithm [28]. She was introduced in an article titled "ADAM: Methods of Stochastic Optimization"" by Diederik Kingma and Jimmy Ray Ba, in 2015 [29].
   Adam Optimizer can be expressed mathematically by the formula 2.6 below:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2 \tag{2.6}$$

   where:

   - w_t weights at time t
   - δL derivative of Loss Function.
   - δw_t derivative of weights at time t.
   - m_t aggregate of gradients at time t (initially, $m_t = 0$).
   - m_{t-1} aggregate of gradients at time t-1.
   - v_t sum of square of past gradients.
   - β_2 decay rate of average of gradients in the above two methods where: $\beta_2 = 0.999$.
   - β_1 decay rate of average of gradients in the above two methods where: $\beta_1 = 0.9$.
   - ϵ is a small positive constant introduced to avoid 'division by 0' error when v_t approaches 0 ($v_t \to 0$). In this context, ϵ is typically set to a small value, such as 10^{-8}.

2.3.3 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a special category of neural network that allows information to flow in both directions. A RNN has short-term memory that enables it to factor previous input when producing output. The short-term memory allows the network to store past information and, hence, revealing relationships between data points that are far apart. [30].

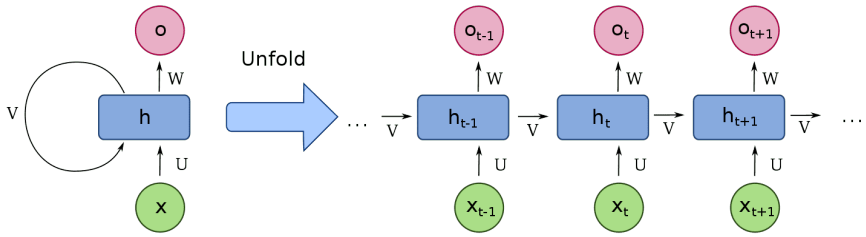


Figure 2.8: Open Source Structure of RNN

There are four types of RNN (figure 2.9):

1. **One to one** has one input and one output.
2. **One to many** has one input and multiple outputs.
3. **Many to one** has multiple inputs and a one output.
4. **Many to many** has many inputs and many outputs.

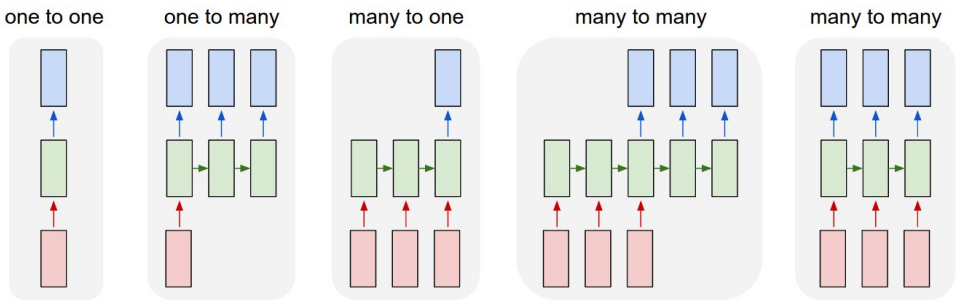


Figure 2.9: Types of RNNs mentioned in [3]

Problems of RNNs

RNNs work using Backpropagation Through Time (BPPTT). This will update the weights of the current and previous input. Weights are updated by transferring the error from the last time step to the first time step. As a result, the error for each time step is calculated. RNNs can not to handle long-term dependencies.

Two main problems occure at very long time steps: Vanishing Gradients and Exploding Gradients [30]. These issues can adversely affect RNN training and performance. When the gradient vanishes, the network can no longer learn from the past and loses the ability to capture long-term dependencies. This can lead to poor generalization and poor fitting. Explosive increases in the gradient make the network unstable and sensitive to small changes in the input. This can lead to numerical overflow, erratic behavior, and overfitting of [31].

Prevent or Mitigate the Vanishing and Exploding Gradient Problems

Various techniques can be used to prevent or mitigate the vanishing and exploding gradient problem in RNNs. Gradient clipping is a simple technique for setting a threshold

for the maximum or minimum value of a gradient. Gradients above or below this value will be clipped or rescaled. Weight initialization is another technique that involves choosing appropriate values for the network's initial weights and biases. A Long Short-Term Memory (LSTM) cell or Gated Recurrent Unit (GRU) cell is a special type of RNN cell with internal mechanisms that control information flow and gradients. It uses gates that learn to open and close based on inputs and outputs, preventing irrelevant information from accumulating and important information from fading out in cell states. Additionally, these gates can control how much gradients are affected by previous inputs and outputs, preventing them from disappearing or exploding [31] .

2.3.4 Long Short Term Memory (LSTM)

Long Short Term Memory networks (LSTMs) are a special kind of RNN, designed to avoid the long-term dependency problem. They were introduced by Sepp Hochreiter and Jurgen Schmidhuber in their article titled "LONG SHORT-TERM MEMORY" [32] published in 1997. The LSTM unit implements three gates: an input gate, a forget gate, and an output gate. The LSTM cell is presented in figure 2.10.

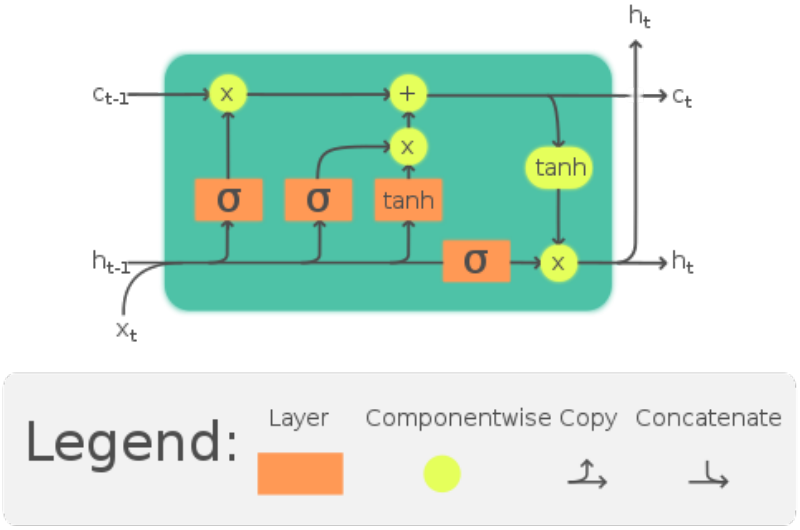


Figure 2.10: LSTM Cell by Guillaume Chevalier, CC BY-SA 4.0

The state updates satisfy the following operations:
The first step in the LSTM is to decide what information is discard from the cell state. This decision is made by a **sigmoid** layer called the Forget Gate Layer f_t. Digging into the previous hidden state of the LSTM cell h at time step $t-1$ and the input x at the time step t and outputs a number between 0 and 1 for each number in the cell state C_{t-1} [33, 34].

- 1 : represents "keep this completely ".
- 0 : represents "remove this completely".

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.7}$$

where:

- b_f: denotes the bias term associated with the forget gate.

Next, a decision of what new information we need to store in the cell state is taken. This consists of two parts, First, a **Sigmoid** layer called the input gate layer i_t decides which value to update. The **tanh** level then creates a vector $C‘_t$ of new candidate values that can be added to the state. In the next step, these two parts are combined to create an update with status [33, 34].

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.8}$$

$$C'_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.9}$$

Then the old cell state C_{t-1} is updated to the new cell state C_t. The old state is multiplied by f_t, forgetting what you decided to forget before. Then $i_t C'_t$ is added as [33, 34].

$$C_t = f_t C_{t-1} + i_t C'_t \tag{2.10}$$

The next step is to determine the output. This output will be taken from our cell state but in a more sophisticated format. Initially, we pass it to a **sigmoid** layer, which decides which aspects of the cell state to free. Next, we apply **tanh** transformation (limiting values to the range -1 to 1) to the subject the cell state and multiply it by the result of the sigmoid gate. This process ensures generating output only based on the selected components [33, 34].

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = tanh(W_h[h_{t-1}, x_t] + b_h) \tag{2.12}$$

2.3.5 Encoder Decoder Seq2Seq Model

The encoder-decoder architecture is a deep learning architecture consists of two main components: an encoder and a decoder. The encoder takes in an input sequence and creates a fixed-length vector representation of it, often called a hidden representation. This representation intended to capture the key information from the input sequence in a compressed form. The decoder then takes the latent representation and produces an output sequence based on it. The most basic components used to build the encoder-decoder architectures are neural networks. Depending on encoder decoder architecture [35], Different types of neural networks, including RNNs, can be used.

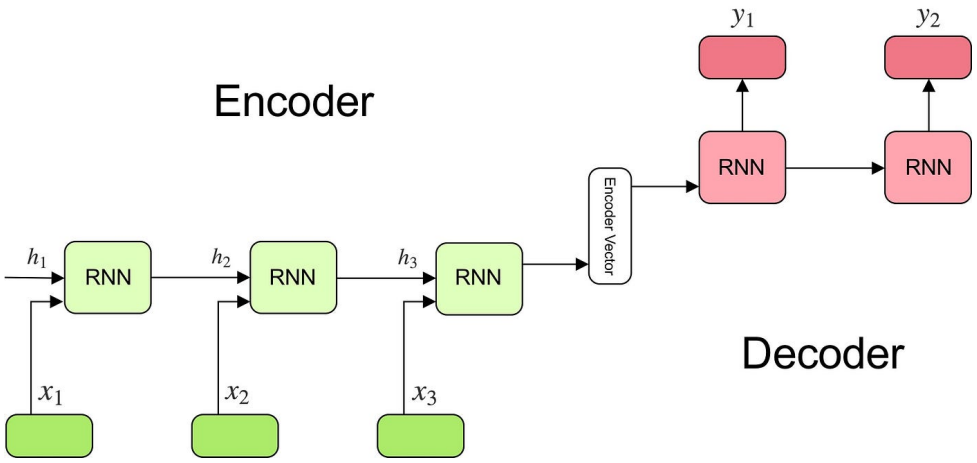


Figure 2.11: An Encoder-Decoder Simple Seq2Seq model

2.3.6 Attention Mechanism

The attention mechanism is a method of unequally weighting the contributions of different input features to enhance the target learning. [36]. In the typical attention mechanism [37, 38] within a RNN, The set of inputs is expressed by $H = \{h_1, h_2, \ldots, h_{t-1}\}$ which is determined based on previous states and the context vector v_t is extracted from the previous states. v_t is the weighted summation of each column h_i in H, capturing information related to the current time step. v_t is further combined with the present state h_t to make the prediction. Suppose a scoring function: $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ that computes the relevance between its input vectors. Formally, we have the following formula to compute the context vector v_t [39]:

$$\alpha_i = \frac{\exp(f(h_i, h_t))}{\sum_{j=1}^{t-1} \exp(f(h_j, h_t)} \tag{2.13}$$

$$v_t = \sum_{i=1}^{t-1} \alpha_i h_i \tag{2.14}$$

The purpose of this attention algorithm is to empower the model to direct its attention toward particular segments of the input sequence (h_i) while producing an output (v_t).

2.4 Evaluation Metrics

In this section, we explore the concept of evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Coefficient of Determination (R^2), and Root Mean Squared Error (RMSE). We are exploring these metrics because we are currently working on predictive models to address a regression problem. These metrics are instrumental in assessing the performance of the models presented in chapter 6.

2.4.1 Mean Squared Error (MSE)

MSE calculates the average of the squared differences between the predicted and actual values, offering an evaluation of how well the predictions align with the true values, on average. As shown in the equation:

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.15}$$

where:

- n is the number of data points in your dataset,
- y_{i} denotes the actual (observed) value of the target variable at data point i,
- $\hat{y}_{\text{i}}$ represents the predicted value of the target variable at data point i.
- $(y_i - \hat{y}_i)^2$ calculates the squared difference between the actual and predicted values for each data point.

It penalizes larger errors significantly, which can be useful when we want to heavily penalize large prediction errors.

2.4.2 Root Mean Squared Error (RMSE)

As indicated by its name, Root Mean Squared Error (RMSE) is a straightforward concept, being the square root of the Mean Squared Error.

$$\textbf{RMSE} = \sqrt{MSE} \tag{2.16}$$

2.4.3 Mean Absolute Error (MAE)

As indicated by its name, Mean Absolute Error(MAE) represents the average of the absolute values of the errors, providing valuable insights into the magnitude of deviations between predicted and actual values. As shown in the equation:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.17}$$

2.4.4 Coefficient of Determination (R^2)

R^2 is a statistical measure utilized to evaluate how much of the variability in the dependent variable can be attributed to the independent variables in a regression model.

The coefficient of determination is a value between 0 and 1. An R^2 value of 1 means that the model perfectly fits the data, while an R^2 value of 0 means that the model does

not explain any of the variance in the dependent variable.

$$R^2 = 1 - \frac{SSR}{SST} \tag{2.18}$$

where:

- SSR represents the sum of squared errors.

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.19}$$

- SST denotes the total sum of squares.

$$SST = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{2.20}$$

2.5 Implementation Tools

In this section, we introduce the key tools and frameworks utilized in our project's implementation. These tools are essential for various tasks, such as data analysis and deep neural network training. Our focus centers on three primary tools: Python, TensorFlow, and Keras, in addition to the cloud-based platform, Google Colab.

2.5.1 Python

Python is a high-level programming language known for its simplicity and readability. It was created by Guido van Rossum and was first released on February 20, 1991. It has gained immense popularity in various domains, including data analysis, machine learning, and web development. It offers a vast ecosystem of libraries and frameworks that provide powerful tools for different tasks, such as NumPy for numerical computing, pandas for data manipulation, and scikit-learn for machine learning [40].

2.5.2 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem for building and deploying machine learning models, especially deep neural networks. TensorFlow allows users to define and train complex mathematical models by creating computational graphs. These graphs represent the flow of numerical operations and can be executed on various devices, such as CPUs, GPUs, and TPUs. TensorFlow offers a wide range of tools and APIs for different levels of abstraction, allowing flexibility and scalability in model development [41].

2.5.3 Keras

Keras is a high-level neural network API that runs on top of TensorFlow (and other deep learning frameworks like Theano and Microsoft Cognitive Toolkit). Keras aims to provide a user-friendly and intuitive interface for building neural networks. It simplifies the model creation and training process by abstracting away many low-level implementation details. Keras offers a modular and flexible architecture for defining models, allowing you to easily stack layers, connect inputs and outputs, and customize model behavior. It supports a wide range of network architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers [42].

2.5.4 Google Colab

Google Colab (short for Collaboratory) is an online cloud-based platform provided by Google for running and collaborating on Python code. It offers a Jupyter Notebook-like environment where users can create and execute Python code in their web browser. Google Colab provides several advantages, such as free access to powerful hardware accelerators like GPUs and TPUs, pre-installed libraries like TensorFlow, and seamless integration with other Google services. It allows users to easily share notebooks, collaborate with others in real time, and run code without the need for local setup or hardware configuration.

2.6 Conclusion

This Chapter serves as the theoretical foundation for multivariate time series forecasting, exploring recurrent neural networks like LSTM and Encoder Decoder Seq2Seq models, along with the attention mechanism. We introduce essential evaluation metrics, including MAE, MSE, R^2 and RMSE to assess model performance. And we explore the key implementation tools that enable us to build and deploy the forecasting models, including: Python, TensorFlow, Keras and Google Colab.

Chapter 3

Related Work

Numerous old and recent studies [5, 6, 9, 10, 43, 44, 45], are proposed as a possible solution to the problem of multivariate time series forecasting. Below is a selection of those works:

3.1 Statistical Methods

Saputro et al (2011), Diani et al (2013), Adenomon (2013) and Das (2013) used VAR models to analyze the relationships between variables in the field of meteorology [46] while Lemya Taha in [47] employed Vector Autoregressive Model to analysis the relationship between two financial time series as well as forecasting.

3.2 Neural Networks

Yoga Estu Nugraha Nugraha, Ishak Ariawan and Willdan Aprizal Arifin in their article [48] employed the LSTM neural networks to forecast future weather conditions and They proceeded to assess the model's effectiveness by employing the RMSE evaluation metric. In [49] Qing Li, Jinghua Tan, Jun WangJun Wang and HsinChun Chen introduces a tensor-based event-driven LSTM model that preserves the interrelations between different data types, allowing for the capture of their combined effects on Stock Prediction Using Online News.

3.3 Hybrid Models

Mogarala Tejoyadav, Rashmiranjan Nayak and Umesh Chandra Pati [50] suggested a hybrid **VAR-LSTM** model to predict the 04 water pollutants along with their water quality index. They utilized a VAR model to capture the interrelationships among the various water pollutants using multivariate time series analysis. Subsequently, the estimated values from the VAR model were fed into an LSTM model to explore the temporal patterns and characteristics of the water quality data. This integration of models resulted in more precise predictions of water quality compared to the outcomes achieved by using each model individually.

In [43], the authors introduced a novel approach for multivariate workload prediction using a combination of**the Vector AutoRegressive (VAR) model** and **Stacked LSTM neural network**. The VAR model is employed to analyze multivariate time series data and forecast their future values. Subsequently. the residues obtained from the VAR model are computed and utilized as inputs for the subsequent stacked LSTM model. This stacked LSTM model effectively forecasts future resource values while taking into account the information captured in the VAR model's residuals. However, the results obtained from the proposed approach indicated lower values of Mean Square Error (MSE) compared to the ARIMA-LSTM (CPU prediction), RNN-GRU and AR-MLP models.

Harya Widiputra, Adele Mailangkay and Elliana Gautama [51] developed a composite **CNN-LSTM** model for Financial Time-Series Prediction. By leveraging the feature extraction capabilities of CNNs and the temporal modeling abilities of LSTMs, the combined model effectively captures complex patterns and dependencies in the multivariate time series data, leading to improved performance in forecasting tasks.

XIANYUN WEN and WEIBANG LI [44] introduced a framework called **AT-LSTM** for forecasting financial time series that leverages the power of attention mechanisms which selects the most relevant input features and effectively improves the accuracy of prediction to a certain extent, and has skillfully captures longer time dependencies on three real datasets.

In [45], the authors introduced a new model for financial time series prediction, which combines **Long Short-Term Memory (LSTM)** with **Multi-Layer Perceptron (MLP)** to map the original features into a latent space and a **feed-forward attention mechanism** which assigns attention weights to emphasize important feature information

3.4 Limitations in Existing Research

Although there has been notable advancement in the domain of multivariate time series forecasting through previous research, there remain various gaps and constraints that require further attention and resolution.

One of the main limitations in current research concerns the application of deep learning techniques, which are characterized by high data requirements. These methods are poorly suited for data-limited scenarios due to their insatiable appetite for large-scale training data. Moreover, their inherent complexity often requires computationally intensive training, which can reduce their practicality in real-time applications and resource-constrained situations.

A second notable limitation concerns the complex process of hyperparameter tuning. Achieving optimal performance in a deep learning model requires careful tuning of hyperparameters, a process that can be time consuming. Moreover, preprocessing of multivariate time series data can be complex, requiring careful handling of data transformation and scaling. Moreover, choosing the right architecture for a particular prediction task remains a significant task, further complicating the use of deep learning techniques in practice.

Chapter 4

Dataset

4.1 Introduction

This chapter primarily revolves around two key elements: the case study conducted and the dataset utilized for the implementation of the study.

4.2 Study Case

The plastic extrusion process involves melting plastic materials and shaping them continuously using an extruder machine. Plastic pellets are fed into a hopper and driven forward by a rotating screw powered by a motor. Inside the heated barrel, the plastic undergoes melting, transforming into molten plastic. This molten plastic is then extruded through a shaping die, resulting in the desired final shape (see Figure 4.1).

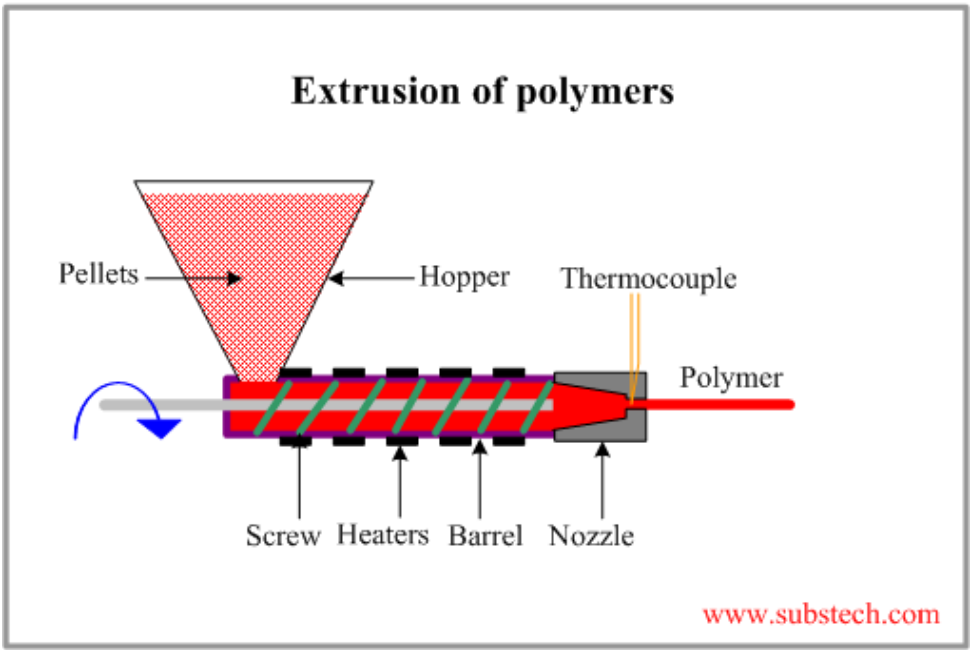


Figure 4.1: Extrusion process

Currently, the plastic extrusion process relies on fixed motor speed and heater temperature setpoints, assuming consistent material properties from plastic pellets. These

static setpoints, determined by machine operators, remain unchanged during the extrusion process. Monitoring output material pressure, temperature, and motor torque serves as the primary means of ensuring smooth operation.

However, integrating recycled plastic introduces variability in material properties, challenging the validity of the assumed constant setpoints. To address this challenge and incorporate the process into the production line, the research project "Recyplast Demo" has undertaken the task.

It is imperative to dynamically adjust the motor speed and heater temperature based on real-time output material pressure in Bars, material temperatures in Celsius degrees, motor speed, motor torque and 17 heaters. To achieve this, leveraging historical data captured through an acquisition system to create a forecasting model for the plastic extruder's behavior becomes crucial. This forecasting model will accurately predict how the extruder will perform under various conditions and contexts, allowing it to maintain consistent output quality despite the varying input properties of recycled plastic.

4.3 Data Exploration

The dataset consists of 41,293 observations from three datasets collected by ICAM as part of the Recyplast-Demo project, with the first recorded on November 25th, the second on March 27st, and the third on April 24th. It contains valuable information on the variations of 23 features, including material pressures (called Pression matière), temperatures (called Température matière), motor torquee (called Moteur Courant), motor speed (called Vitesse Moteur), and 17 heaters(called (zone1,zone2,..,zone17)) recorded over time. These data points offer crucial insights into how these parameters change throughout the given period.

To begin with, we first included the essential python libraries (such as pandas, NumPy, matplotlib, seaborn) and then loaded the three datasets. During this step, we set the timestamp feature as the index and converted its type into datetime format for better handling of time-related data. Finally, we concatenated the three datasets into a single dataset to facilitate further analysis and manipulation.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df1 = pd.read_csv('/content/Extrusion_2022-11-25-20230609091145-synchronized-.csv',
    delimiter= '\t',index_col='timestamp',parse_dates=True)
df2 = pd.read_csv('/content/Extrusion_2023-03-27-20230609091553-synchronized-.csv',
    delimiter= '\t',index_col='timestamp',parse_dates=True)
df3 = pd.read_csv('/content/Extrusion_2023-04-24-20230609092228-synchronized-.csv',
    delimiter= '\t',index_col='timestamp',parse_dates=True)
concatenated_df = pd.concat([df1, df2, df3], axis=0)
```

The next step is to inspect for any missing values.Fortunately, there were no null values present, eliminating the need for imputation.

```
df.isnull().sum()
```

The next step is examining duplicate entries. These duplicates correspond to identical rows or observations in the dataset and can occur accidentally during the acquisition and collection of process measurements. To ensure data cleanliness and avoid redundancy, we promptly removed to retain only unique observations. As a result of this operation, the dimension of the dataset consisted of 41293 entries was reduced to 17693 entries.

```
df.duplicated()
df.drop_duplicates(inplace=True)
```

To gain a deeper insight into the connections between different features in our dataset, we utilized a powerful visualization technique called heatmap. This heatmap is based on the correlation matrix, which calculates the strength and direction of correlations between each pair of features.

```
sns.heatmap(df.corr())
```

The figure presents the information derived from the visualization 4.2 Upon examining

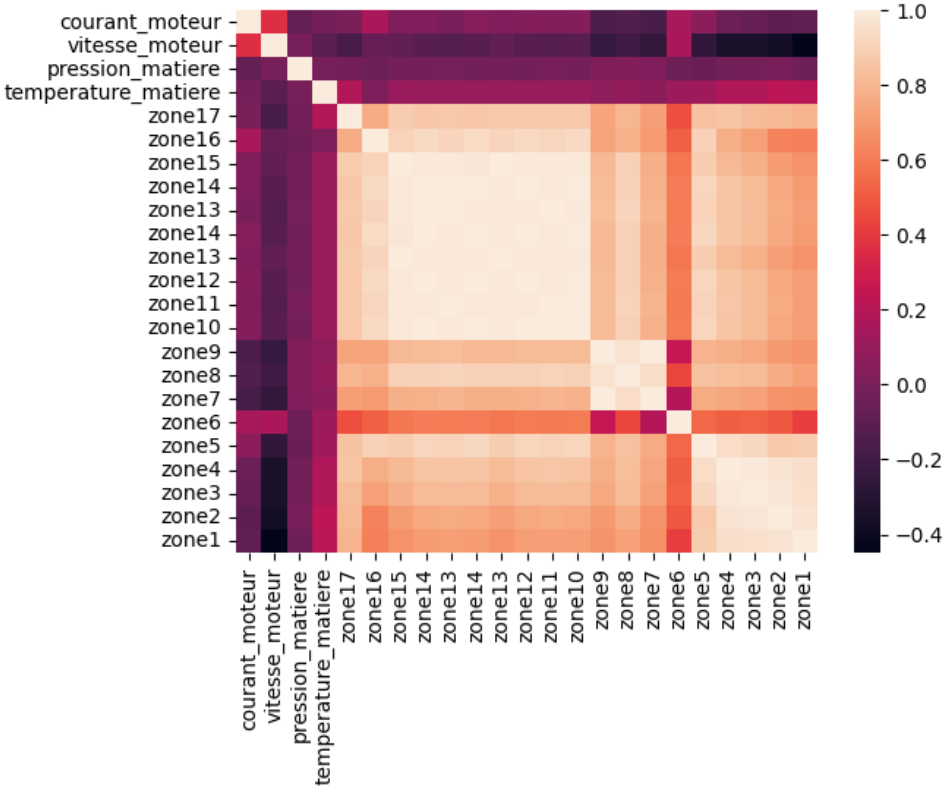


Figure 4.2: Corralation Matrix

the corrrelation matrix, we make several key observations:

1. The variables "zone 1," "zone 2," "zone 3," "zone 4," and "zone 5" exhibit high correlation coefficients among themselves, suggesting a strong similarity or close relationship. These variables were grouped due to their high correlation.

```
df['zone1_5'] = df[['zone1', 'zone2', 'zone3','zone4','zone5']].mean(axis=1)
```

2. The variables "zone 7," "zone 8," and "zone 9" also demonstrated high correlation coefficients among themselves, indicating a close relationship or similarity. They were grouped accordingly.

```
df['zone7_9'] = df[['zone7', 'zone8', 'zone9']].mean(axis=1)
```

3. The variables "zone 10," "zone 11," "zone 12," "zone 13," "zone 14," "zone 15," "zone 16," and "zone 17" displayed high correlation coefficients among themselves, justifying their grouping.

```
df['zone10_17'] =df[['zone10','zone11','zone12','zone13','zone14','zone15','zone16'
    ,'zone17']].mean(axis=1)
```

After grouping the variables based on their high correlation coefficients and subsequently dropping duplicates, the resulting dataset is now reduced in size and contains a subset of unique observations. As illustrated in the figure 4.3.

timestamp	temperature_matiere	pression_matiere	vitesse_moteur	courant_moteur	zone7_9	zone1_5	zone6	zone10_17
2022-11-25 13:56:40.036	148.0	0.0	0.0	0.0	181.000000	170.0	184.0	188.7
2022-11-25 13:58:48.065	148.0	0.0	0.0	0.0	181.000000	170.0	184.0	188.8
2022-11-25 14:03:30.060	148.0	0.0	75.0	276.0	181.000000	170.0	184.0	188.8
2022-11-25 14:03:31.060	148.0	0.0	75.0	323.0	181.000000	170.0	184.0	188.8
2022-11-25 14:03:32.060	148.0	0.0	0.0	354.0	181.000000	170.0	184.0	188.8
...	...	...	...	...	...	...	...	...
2023-04-24 17:21:54.939	196.0	78.0	69.0	418.0	215.666667	215.2	219.0	213.7
2023-04-24 17:21:55.939	196.0	80.0	69.0	405.0	215.666667	215.2	219.0	213.7
2023-04-24 17:21:56.938	196.0	80.0	70.0	452.0	215.666667	215.2	219.0	213.7
2023-04-24 17:21:57.939	196.0	79.0	70.0	383.0	215.666667	215.2	219.0	213.7
2023-04-24 17:21:58.939	196.0	81.0	69.0	391.0	215.666667	215.2	219.0	213.7

17693 rows × 8 columns

Figure 4.3: Reduced Dataset

In the following table, we present statistical data for key parameters related to the extrusion process. This data provides valuable insights into the characteristics and variability of these parameters. As illustrated in the figure 4.4.

	temperature_matiere	pression_matiere	vitesse_moteur	courant_moteur	zone7_9	zone1_5	zone6	zone10_17
count	17516.000000	17516.000000	17516.000000	17516.000000	17516.000000	17516.000000	17516.000000	17516.000000
mean	174.247831	56.043275	30.583467	470.602763	186.933375	176.267447	184.274892	187.162520
std	46.964699	1106.722137	21.941360	103.285372	12.107486	16.787242	16.457677	9.122471
min	141.000000	0.000000	0.000000	0.000000	170.666667	146.200000	128.000000	169.000000
25%	155.000000	18.000000	9.000000	467.000000	180.333333	170.200000	179.000000	183.300000
50%	165.000000	38.000000	20.000000	489.000000	185.333333	176.000000	184.000000	188.800000
75%	171.000000	55.000000	50.000000	508.000000	187.000000	182.000000	189.000000	189.400000
max	420.000000	65535.000000	75.000000	1269.000000	218.333333	219.400000	223.000000	214.400000

Figure 4.4: Statistical Summary of Extrusion Process Parameters

We divide the dataset into the train part and the test part

```
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(df, test_size=0.20, shuffle=False)
```

For the normalization of the data we will use Min Max Scaler to ensure the values are scaled within the range of 0 to 1, such as: for every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. The Min Max Scaler equation is given by:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{4.1}$$

where:

- X is the original data,
- X_{min} is the minimum value of the data.
- X_{max} is the maximum value of the data.
- X_{scaled} is the scaled data between 0 and 1.

The code snippet for the Min Max Scaler in Python is as follows:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(cc)
```

In order to address the multivariate time series data as a supervised learning problem, we adopt the sliding window method. The sliding window approach is utilized in the following part of the code:

```
# Define the window size for the LSTM model
window_size = 3
# Create the input sequences and labels
X = []
y = []
for i in range(window_size, len(scaled_data)):
    X.append(scaled_data[i-window_size:i])
    y.append(scaled_data[i])
X = np.array(X)
y = np.array(y)
```

We conducted an extensive exploration of various sliding window sizes, ranging from 3 to 25. However, in our final configuration, window_size is set to 3, and a for loop is used to create input sequences (X) and corresponding labels (y) for training the LSTM model. Each element of X is a sequence of three consecutive data points, and the corresponding element in y is the next data point after the sequence. As the loop progresses, the window slides through the data, creating overlapping sequences, which allows the model to learn patterns and relationships in the time-series data. This transformation enables the model to learn from historical patterns and predict future values based on previous observations.

Chapter 5

Modeling and Implementation

5.1 Introduction

The Experiments chapter focuses on the implementation of the solution and provides specific steps taken to execute it.

5.2 VAR model

In this section, we will employ the VAR model to forecast our multivariate time series (MTS) data. First, we focus on evaluating the stationarity in the MTS data. If non-stationarity is detected, a transformation method is employed to achieve stationarity. Following this, our objective will be to establish an effective model, which involves determining the optimal number of lag terms to include in our analysis.

5.2.1 Checking Stationarity

To initiate the training of our VAR model, our first focus should be on testing the stationarity of our multivariate time series. This allows us to determine whether our variables exhibit stable behaviors over time. To conduct this test, we employ the Augmented Dickey-Fuller Test, which compares the original time series with a lagged version of itself to ascertain the presence of non-stationary behavior. This concept has been elaborated upon in chapter 2.

```
# we Select the variables of interest
variables_of_interest = ['temperature_matiere', 'pression_matiere', 'vitesse_moteur',
                         'courant_moteur', 'zone7_9', 'zone1_5', 'zone6', 'zone10_17']
final_data = concatenated_df[variables_of_interest]

# we Create an empty DataFrame to store ADF test results
adf_results = pd.DataFrame(columns=['Variable', 'ADF Statistic', 'p-value', 'Is
    Stationary'])

# we Perform ADF test for each variable in the dataset
for column in final_data.columns:
    result = adfuller(final_data[column])
    is_stationary = result[1] <= 0.05
    adf_results = adf_results.append({
```

```
        'Variable': column,
        'ADF Statistic': result[0],
        'p-value': result[1],
        'Is Stationary': is_stationary
    }, ignore_index=True)
```

The test furnishes a test statistic and an associated p-value. Should the p-value be lower than a predetermined threshold (0.05), we reject the null hypothesis indicating non-stationarity.
The result of this test indicates that 4 series are stationary (temperature_matiere, pression_matiere, vitesse_moteur, courant_moteur), while the other 4 series are non-stationary (zone1_5, zone6, zone7_9, zone10_17) as shown in figure 5.1. Therefore, a Differencing transformation is necessary.

```
final_data.diff().dropna()
```

```
adf_results = adf_results.append({
```

	Variable	ADF Statistic	p-value	Is Stationary
0	temperature_matiere	-7.049159	5.584611e-10	True
1	pression_matiere	-18.789606	2.023565e-30	True
2	vitesse_moteur	-4.961215	2.646893e-05	True
3	courant_moteur	-6.971835	8.625098e-10	True
4	zone7_9	-0.200653	9.384437e-01	False
5	zone1_5	-0.547636	8.823808e-01	False
6	zone6	-1.725612	4.179669e-01	False
7	zone10_17	0.749511	9.907759e-01	False

Figure 5.1: Adfuller Test Results

5.2.2 Lag Order Determination

To determine the optimal lag order, we perform a sort of test where we identify when the information criterion reaches its minimum value. We plot the the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), Focused Prediction Error (FPE) and the Hannan-Quinn Information Criterion (HQIC) values againt the lag orders as illustrated in visualisation 5.2 below:

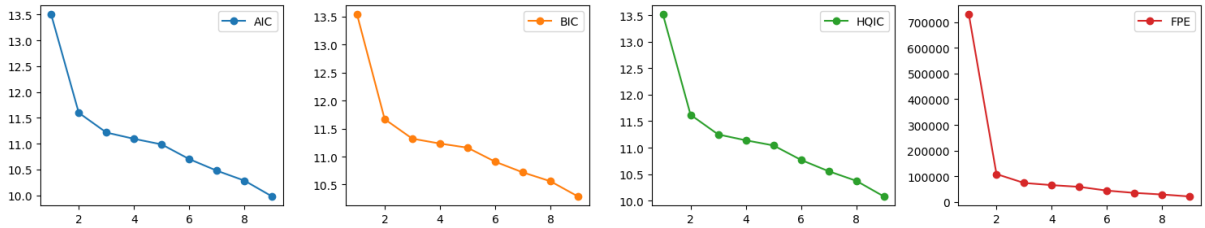


Figure 5.2: Lag Order Determination

In this case, a lag of five (5) is chosen because it represents the point at which the information criterion (AIC, BIC) indicates the best balance between model complexity

and predictive accuracy. It signifies that a lag of 5 captures the relevant historical information in the data without overcomplicating the model, making it an optimal choice for forecasting purposes.

5.2.3 Model Fitting

The next step revolves around training the VAR model, using the training differencing data, wherein the employment of the lagged values of the variables as input to generate the appropriate predictions.

```
p = 5
model = VAR(train_diff)
var_model = model.fit(p)
var_model.summary()
```

The Forecasting results will be provided in chapter 6.

5.3 Long Short Term Memory (LSTM) Model

We specify the model architecture as demonstrated in the figure 5.3 with the following considerations:

- We create a sequential model, a linear stack of layers that simplifies the construction of deep neural networks.

```
    model1 = Sequential()
```

- In our pursuit of optimizing the LSTM model, we conducted extensive training experiments with varying numbers of epochs, including 25, 35, 40 and 100 epochs. The results of our intensive experiments showed a clear trend. The model's performance peaked surprisingly early, after only 25 epochs. The model incorporates an LSTM layer with 25 units and 'tanh' activation. 'tanh' activation is preferred in LSTM networks due to its effectiveness in handling vanishing gradient problems compared to sigmoid and other activation functions.

- The input shape is set to (n_steps, 8), where 'n_steps' represents the number of time steps in each sample, and '8' corresponds to the number of features in the multivariate time series data. This allows the LSTM to process the input data correctly, considering both temporal structure and feature dimensions.

```
    model1.add(LSTM(25, activation='tanh',input_shape=(n_steps,8)))
```

- To address overfitting, we introduce a Dropout layer with a 0.10 dropout rate. Dropout randomly sets a portion of input units to zero during training, promoting robust representations and reducing reliance on specific features.

```
    model1.add(Dropout(0.10))
```

- After the LSTM layer, we add a Dense layer with 8 units, matching the output size. This final layer enables the model to produce predictions for the multivariate time series data.

```
model1.add(Dense(8))
```

- We compile the model using the Adam optimizer, a popular choice for training neural networks. Adam combines the benefits of AdaGrad and RMSProp, adjusting learning rates for each parameter, leading to faster convergence and improved performance compared to standard stochastic gradient descent.

- The loss function used during training is Mean Squared Error (MSE), measuring the difference between predicted values and actual targets. Minimizing MSE encourages accurate predictions and effective learning from the training data, making it suitable for regression tasks like time series forecasting.

```
model1.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),loss='mse')
```

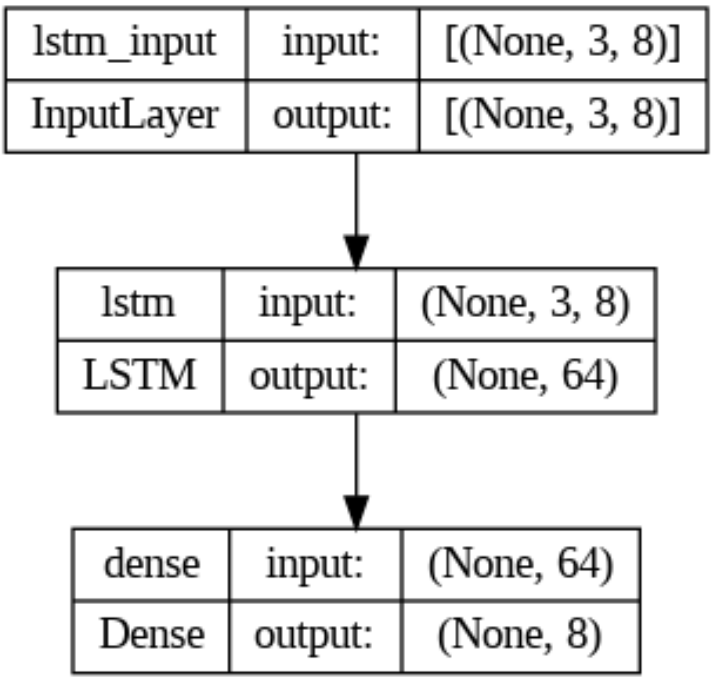


Figure 5.3: LSTM model

Finally, using the trained LSTM model, we perform multivariate series forecasting on the test data. The predictions will be showcased in the "Results" section through visualizations that demonstrate the forecasted values alongside the actual test data. Additionally, we will provide a table containing Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2) metrics for each feature, assessing the model's performance on various aspects.

5.4 Encoder Decoder Model

The initial phase involves importing essential modules from TensorFlow's tensorflow.keras package, including tools like regularizers to prevent overfitting, the Model class for creating the model, and various layer types.

```
from tensorflow.keras.layers import Input, LSTM, concatenate, RepeatVector,
    TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
```

Following this, we proceed by creating the input layer for the encoder. The input layer will handle past data instances, where the specific case involves sequences with 20 time steps and 8 distinct features.

```
n_past = 7
n_features = 8
encoder_inputs = Input(shape=(n_past, n_features))
```

The encoder consists of two LSTM layers: the first contains 32 units and the second with 16 units. These layers process the input sequences while maintaining their internal states. The states of the second layer are concatenated with the states of the first layer, forming the context vector that retain the most relevant information. This context vector is then repeated to match the number of time steps in the future predictions.

```
# Define the inputs
encoder_inputs = Input(shape=(n_past, n_features))
# Encoder LSTM layers
encoder_l1 = LSTM(32, return_sequences=True, return_state=True,activation = 'tanh')
encoder_outputs1 = encoder_l1(encoder_inputs)
encoder_states1 = encoder_outputs1[1:]
encoder_l2 = LSTM(16, return_state=True)
encoder_outputs2 = encoder_l2(encoder_outputs1[0])
encoder_states2 = encoder_outputs2[1:]
# Concatenate the encoder states
encoder_states = concatenate([encoder_states1[0], encoder_states2[0]])
```

For the decoder, a RepeatVector layer duplicates the context vector for each time step in the future prediction sequence. Two LSTM layers, with 32 and 16 units respectively, process this repeated context vector. Worth noting is that these decoder LSTM layers are fortified with L2 regularization, designed to curb overfitting, with a regularization intensity of 0.05. The outcome of the second LSTM layer in the decoder undergoes a two-step transformation: first through a TimeDistributed layer and then via a fully connected (Dense) layer. This process ultimately gives rise to the final projected sequences.

```
# Decoder inputs
decoder_inputs = RepeatVector(n_future)(encoder_states)
# Decoder LSTM layers with L2 regularization
decoder_l1 = LSTM(32, return_sequences=True, kernel_regularizer=regularizers.l2(0.05))(
    decoder_inputs, initial_state=encoder_states1)
decoder_l2 = LSTM(16, return_sequences=True, kernel_regularizer=regularizers.l2(0.05))(
    decoder_l1, initial_state=encoder_states2)
decoder_outputs2 = TimeDistributed(Dense(n_features))(decoder_l2)
model = Model(encoder_inputs, decoder_outputs2)
model.summary()
```

Subsequently, The model is constructed using the Model class, and a model's summary is printed, displaying the shapes and number of parameters for each layer.(see figure 5.4)

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 7, 8)]       0           []

 lstm_8 (LSTM)                  [(None, 7, 32),      5248        ['input_3[0][0]']
                                 (None, 32),
                                 (None, 32)]

 lstm_9 (LSTM)                  [(None, 16),         3136        ['lstm_8[0][0]']
                                 (None, 16),
                                 (None, 16)]

 concatenate_2 (Concatenate)    (None, 48)           0           ['lstm_8[0][1]',
                                                                  'lstm_9[0][1]']

 repeat_vector_2 (RepeatVector)  (None, 4, 48)       0           ['concatenate_2[0][0]']

 lstm_10 (LSTM)                 (None, 4, 32)        10368       ['repeat_vector_2[0][0]',
                                                                  'lstm_8[0][1]',
                                                                  'lstm_8[0][2]']

 lstm_11 (LSTM)                 (None, 4, 16)        3136        ['lstm_10[0][0]',
                                                                  'lstm_9[0][1]',
                                                                  'lstm_9[0][2]']

 time_distributed_2 (TimeDistri  (None, 4, 8)        136         ['lstm_11[0][0]']
 buted)

==================================================================================================
Total params: 22,024
Trainable params: 22,024
Non-trainable params: 0
```

Figure 5.4: a summary of the model architecture

We opt for the Adam optimizer and we set up its parameters the learning rate is established at 0.008, the first-moment decay rate (beta_1) at 0.9, the second-moment decay rate (beta_2) at 0.999, and the AMSGrad variant is engaged. The model is compiled with the Mean Squared Error (MSE) as the chosen loss function. The training process spans across 25 epochs, where each epoch involves processing batches of 32 samples.

```
opt = tf.keras.optimizers.Adam(learning_rate=0.0008, beta_1=0.9, beta_2=0.999, amsgrad=
    True)
model.compile(loss='mse', optimizer=opt)
history=model.fit(X_train,y_train,epochs=25,validation_data=(X_test,y_test),batch_size
    =32,verbose=0)
```

In chapter 6, we will present the learning curve depicting the behavior of the loss function over the training epochs. Additionally, we will include a table showcasing the performance evaluation metrics.

5.5 Long Short Term Memory (LSTM) Model And The Attention Mechanism

The model architecture is outlined as illustrated in Figure 5.5 We set up the necessary libraries and modules to build LSTM model with the attention mechanism.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Model
from keras.layers import LSTM, Dense, Dropout, Input, Permute, Multiply, Activation,
    Lambda, Flatten
```

```
from keras import backend as K
```

In the next step, we split the data into training and testing sets while defining the number of time steps to consider in each sample, which is n_steps = 7. This means that the LSTM model will be trained to take 7 consecutive time steps as input to predict the output for the next time step. After that we prepare the training data for both training and evaluation of the LSTM model.

```
train_size = int(len(scaled_data) * 0.80)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]
n_steps = 7
#Prepare training data
X_train, y_train = [], []
for i in range(n_steps, len(train_data)):
X_train.append(train_data[i - n_steps:i, 2:])  # Select the last 6 columns for X_train
y_train.append(train_data[i, 2:])  # Select the last 6 columns for y_train
X_train, y_train = np.array(X_train), np.array(y_train)
#Prepare testing data
X_test, y_test = [], []
for i in range(n_steps, len(test_data)):
X_test.append(test_data[i - n_steps:i, 2:])  # Select the last 6 columns for X_test
y_test.append(test_data[i, 2:])  # Select the last 6 columns for y_test
X_test, y_test = np.array(X_test), np.array(y_test)
```

After that, we define the attention mechanism for the LSTM model that calculates the attention weights using a dense layer with softmax activation, reshapes and expands dimensions to align with input tensor, and then applies element-wise multiplication to scale the input features according to their attention weights.

```
# Define the attention mechanism
def attention(inputs):
    attention_weights = Dense(n_steps, activation='softmax')(inputs)
    attention_weights = Permute((2, 1))(attention_weights)
    attention_weights = Lambda(lambda x: K.expand_dims(x, axis=3))(attention_weights)
    attention_weights = Lambda(lambda x: K.repeat_elements(x, 18, axis=3))(
        attention_weights)
    return Multiply()([inputs, attention_weights])
```

Finally, we construct the LSTM model with an attention mechanism with 18 units, employing the (tanh) activation function. The attention mechanism is then invoked, allowing the model to focus on relevant information within the sequences. To avoid overfitting, a dropout layer is used with a dropout rate of 0.07 follows to regularize the network. The output is then flattened, and a dense layer with 6 units is employed to produce the final predictions, matching the number of six columns in the input data. The model is then compiled with the Adam optimizer and the mean squared error loss function. The model is trained using the training data for 25 epochs with a batch size of 32 and 20% validation split.

```
# Build the LSTM model with attention mechanism
inputs = Input(shape=(n_steps, 6))  # Adjusted to use 6 columns
lstm_out = LSTM(18, activation='tanh', return_sequences=True)(inputs)
attention_out = attention(lstm_out)
dropout_out = Dropout(0.07)(attention_out)
flatten_out = Flatten()(dropout_out)
dense_out = Dense(6)(flatten_out)  # Adjusted to use 6 columns
```

```
model3 = Model(inputs=inputs, outputs=dense_out)
model3.compile(optimizer='adam', loss='mse')
# Train the model
history3 = model3.fit(X_train, y_train, epochs=25, batch_size=32, validation_split=0.20,
    verbose=1)
```

Finally the predictions are done and the results are visualized in the results chapter.

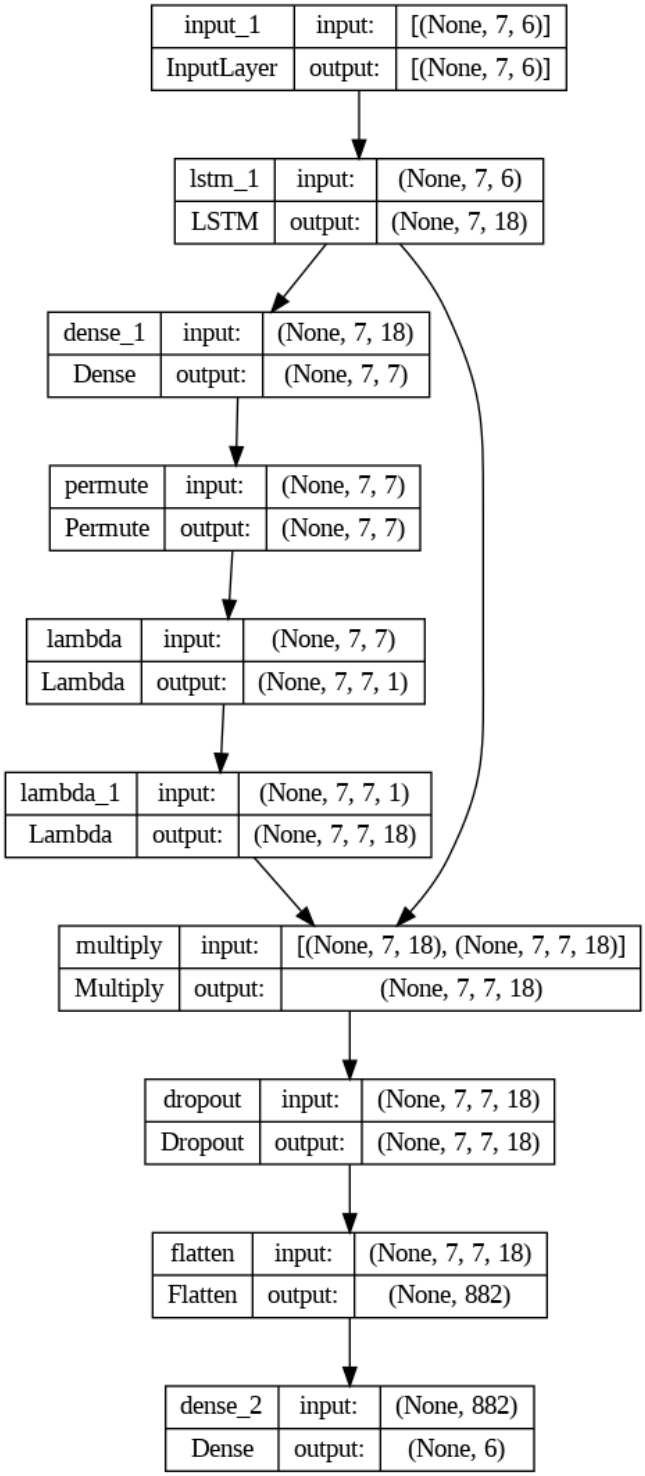


Figure 5.5: LSTM model+Attention Mechanism

Chapter 6

Results

6.1 Introduction

In this chapter, we present the outcomes obtained from employing various models to forecast our multivariate time series. The models under examination include LSTM (Long Short Term Memory), LSTM with the attention mechanism, Encoder-Decoder Model, and VAR (Vector Autoregression). Each model's performance and predictive capabilities are thoroughly analyzed to gain insights into their effectiveness in capturing the dynamics of the multivariate time series data.

6.2 VAR Model

The figure 6.1 presented below delve into the insights provided by the VAR summary. With a total of eight equations, the Akaike Information Criterion (AIC) yielded a value of 9.67 and examine the correlation matrix of residuals that calculates the difference between the actual values and its predictions.

For example, the positive correlation between "zone1_5" and "zone7_9" suggests that when one of these zones has higher-than-expected residuals, the other zone tends to have higher residuals as well while the positive correlation between "zone7_9" and "vitesse_moteur" and between "zone10_17" and "temperature_matiere" are closer to 0 and that indicate a weaker and a negligible relationship.

To understand the impact of "temperature_matiere" and "pression_matiere" on the remaining variables, we employ the Impulse Response Function (IRF) as shown in figure 6.2 and figure 6.3

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Tue, 15, Aug, 2023
Time:                     19:20:45
--------------------------------------------------------------------
No. of Equations:         8.00000    BIC:                    9.99166
Nobs:                     14144.0    HQIC:                   9.78346
Log likelihood:          -228426.    FPE:                    15988.6
AIC:                      9.67963    Det(Omega_mle):         15343.5
--------------------------------------------------------------------
```

(a) Model Summary

```
Correlation matrix of residuals
                      temperature_matiere  pression_matiere  vitesse_moteur  courant_moteur   zone7_9   zone1_5     zone6  zone10_17
temperature_matiere              1.000000          0.003463       -0.014253        0.008145  0.006413 -0.001683 -0.004166  -0.007416
pression_matiere                 0.003463          1.000000       -0.008749       -0.006394  0.038207 -0.072980 -0.102002  -0.041436
vitesse_moteur                  -0.014253         -0.008749        1.000000        0.058863 -0.005166  0.007284  0.019570   0.013437
courant_moteur                   0.008145         -0.006394        0.058863        1.000000  0.005302  0.012404  0.023317   0.011973
zone7_9                          0.006413          0.038207       -0.005166        0.005302  1.000000  0.745560  0.107341   0.680763
zone1_5                         -0.001683         -0.072980        0.007284        0.012404  0.745560  1.000000  0.423151   0.887747
zone6                           -0.004166         -0.102002        0.019570        0.023317  0.107341  0.423151  1.000000   0.471514
zone10_17                       -0.007416         -0.041436        0.013437        0.011973  0.680763  0.887747  0.471514   1.000000
```

(b) Correlation Matrix Of Residuals

Figure 6.1: VAR Summary

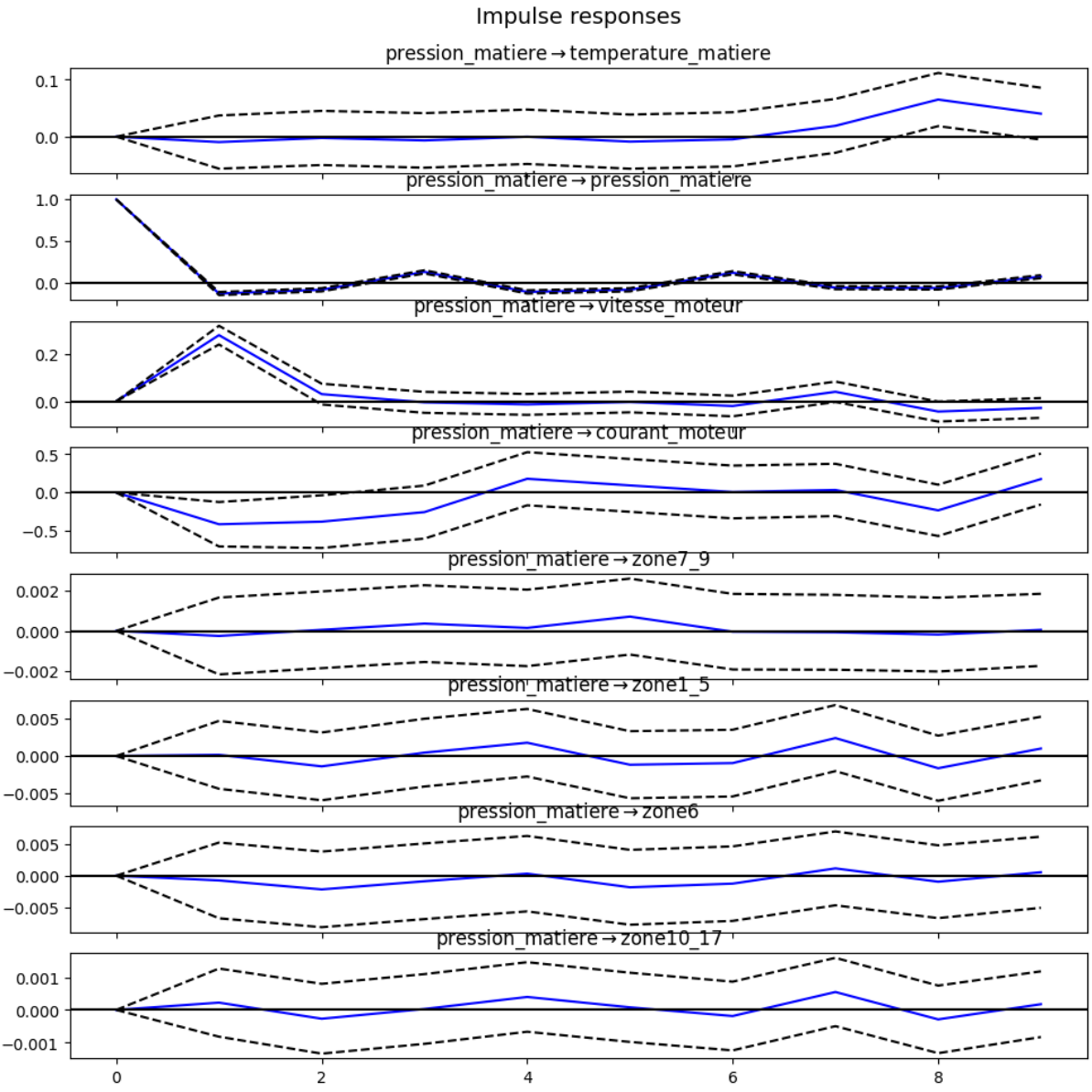


Figure 6.2: Impulse Response Function :"pression_matiere"

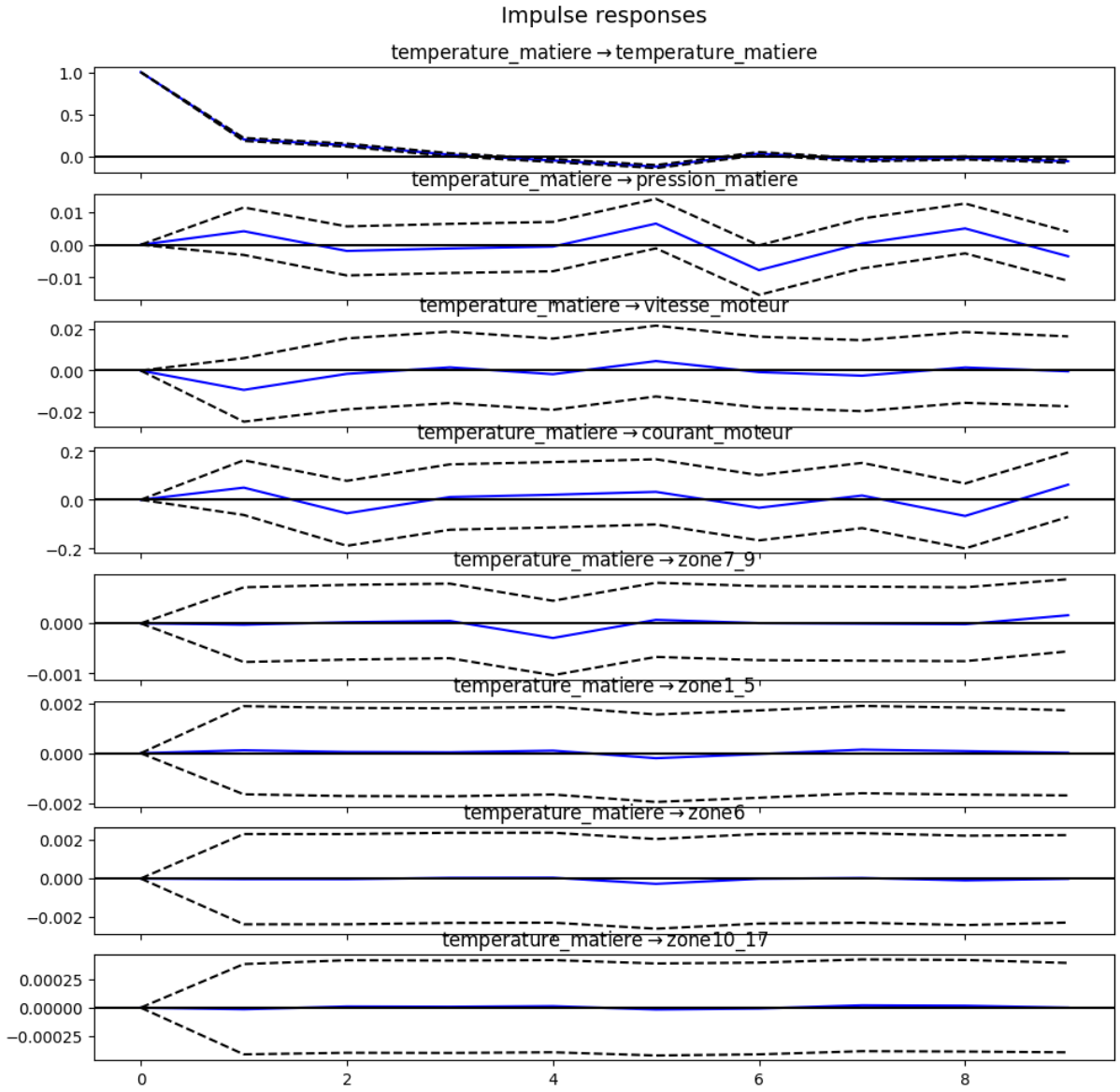


Figure 6.3: Impulse Response Function :"temperature_matiere"

The graph illustrates that "zone7_9," "zone10_17," "zone1_5," and "zone6" exhibit varying response patterns to a "pression_matiere" shock over time. However, a pronounced positive correlation exists between the "pression_matiere" impulse and these responses. For instance, an upsurge of one standard deviation in "pression_matiere" due to the shock leads to a substantial 0.3 standard deviation increase in "vitesse_moteur" during the initial first time periods.In the second time period, the standard deviation decreases . Subsequently, the shock's effect stabilizes during the third time period, followed by a gradual reduction in standard deviation during the final time period.
Finally, we provide a table 6.1 that displays a comprehensive overview of the model's performance on different features along with the corresponding evaluation metrics including R^2 (Coefficient Of Determination), MAE (Mean Absolute Error), MSE (Mean Squared Error), and RMSE (Root Mean Squared Error).

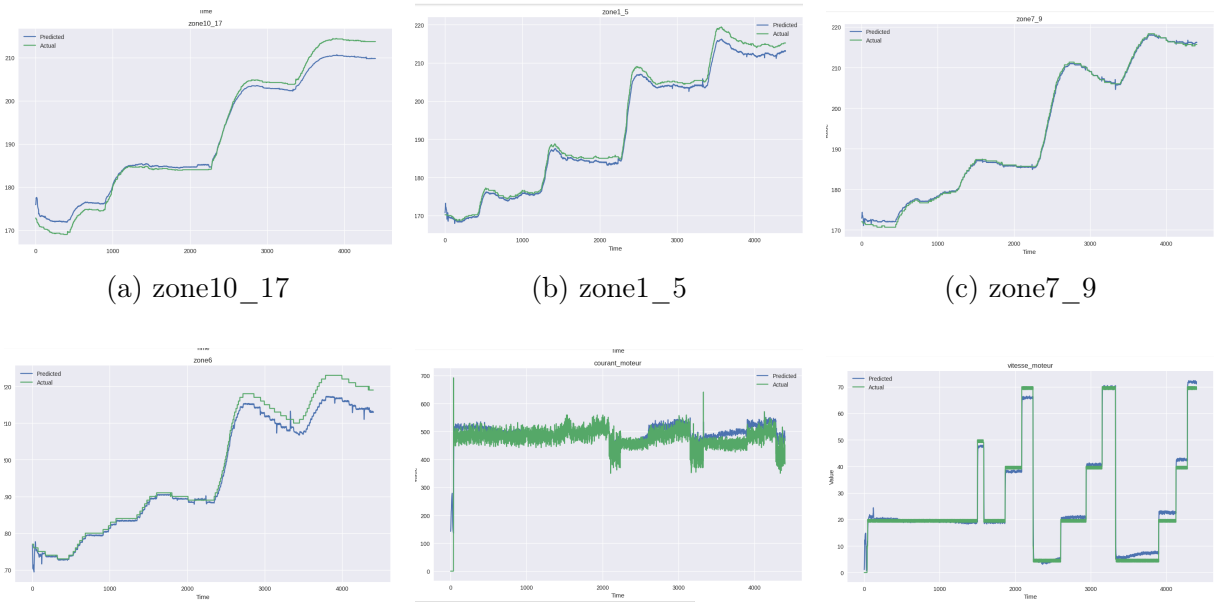
Features & Evaluation metric	R^2	MAE	MSE	RMSE
Courant_moteur	-0.06510	38.73712	8947.79314	94.59277
Vitesse_moteur	-0.39681	13.18129	449.11865	21.19242
Zone1_5	-0.1418533	4.91342	60.73536	7.79329
Zone_6	-0.399640	2.3767	14.78256	3.84481
Zone7_9	-0.3380	2.82434	19.38271	4.40258
Zone10_17	-0.218762	1.45308	6.39842	2.52951

Table 6.1: VAR model results

The model's results for "courant_moteur" and "vitesse_moteur" demonstrate poor performance. The negative R^2 and high values of MAE, MSE, and RMSE suggest that the VAR model fails to accurately predict both "courant_moteur" and "vitesse_moteur", which is crucial for process control in extrusion. It also shows that The VAR model shows subpar performance across all extrusion zones ("Zone1_5," "Zone_6," "Zone7_9," "Zone10_17"). Negative R^2 values and high MAE, MSE, and RMSE values indicate that the model is not suited for predicting characteristics in these zones, which is essential for ensuring the quality of the extruded product.
In summary, the results of the VAR model are unsatisfactory in the context of recycling plastic extrusion.

6.3 Long Short Term Memory (LSTM) Model

The figure's visualization 6.4 enables a comprehensive assessment of how effectively the LSTM model aligns with the actual data.



(a) zone10_17 (b) zone1_5 (c) zone7_9

(d) zone6 (e) courant_moteur (f) vitesse_moteur

Figure 6.4: Plots of the LSTM forecasting model

The following table 6.2 displays a comprehensive overview of the model's performance on different features along with the corresponding evaluation metrics including R^2 (Coefficient Of Determination), MAE (Mean Absolute Error), MSE (Mean Squared Error), and RMSE (Root Mean Squared Error).

Features & Evaluation metric	R^2	MAE	MSE	RMSE
Courant_moteur	0.4524	27.8556	1549.6172	39.3651
Vitesse_moteur	0.9790	1.5001	7.3863	2.7177
Zone1_5	0.9906	1.3738	2.5931	1.6103
Zone_6	0.9685	2.2123	9.4120	3.0679
Zone7_9	0.9988	0.4386	0.4386	0.5666
Zone10_17	0.9809	1.7204	4.4705	2.1143

Table 6.2: LSTM model results

The results of the LSTM model indicate that this model exhibits varying performance for different process characteristics. Here is a context-specific interpretation:

- The LSTM model shows moderate performance in predicting "courant_moteur". Although the coefficient of determination (R^2) is not very high, this model manages to capture some relationships in the data.
- The model's results for "vitesse_moteur" are promising, with an R^2 close to 1, indicating a strong correlation between predictions and actual data. This suggests that the LSTM model is capable of accurately predicting motor speed, which is essential to control the extrusion process.
- The LSTM model displays varying performance for different extrusion zones. Some zones, such as "Zone1_5" and "Zone7_9," show good performance with R^2 values close to 1, while others, like "Zone_6," exhibit less satisfactory performance.

These findings highlight the potential of the LSTM model to contribute to process optimization and waste reduction through precise predictions of process characteristics, even though its performance varies across different aspects of the extrusion process.

6.4 Encoder Decoder Model

In Table 6.3, the results of the Encoder Decoder model are presented, showcasing the performance evaluation metrics for various features that offer a comprehensive view of the model's accuracy, capturing aspects such as variance explained, absolute and squared prediction errors, as well as overall prediction precision.
The Encoder-Decoder model's performance varies across different extrusion zones. Zone1_5 and Zone7_9 exhibit reasonably good predictions with high R^2 values, indicating a strong correlation. However, Zone_6 shows poor performance with a negative R^2, indicating a lack of correlation, while Zone10_17 has relatively good R^2 but slightly higher errors. These variations suggest that the model's success depends on the specific extrusion zone being considered.
The Encoder-Decoder model performs remarkably well in predicting Vitesse_moteur. The high R^2 value of 0.87534 indicates a strong correlation between predicted and actual values. Additionally, the low MAE, MSE, and RMSE values suggest that the model provides accurate predictions for Vitesse_moteur, which is essential for controlling the extrusion process.

Features & Evaluation metric	R^2	MAE	MSE	RMSE
Courant_moteur	0.37651	0.20247	0.06781	0.26042
Vitesse_moteur	0.87534	0.08940	0.02023	0.14225
Zone1_5	0.59293	0.14796	0.06037	0.24572
Zone_6	-0.52548	0.05843	0.01447	0.12030
Zone7_9	0.85756	0.05870	0.00515	0.07182
Zone10_17	0.83036	0.07944	0.01831	0.13534

Table 6.3: Encoder Decoder model results

The learning cure shown in figure 6.5 illustrates the behaviour of MSE loss function over the training epochs. The plot showcases a gradual decrease in the model's loss, indicating a convergence towards a a point of stability The plot showcases of both the training dataset and the validation dataset a gradual decrease in the model's loss, indicating a convergence towards stability. We can notice that the loss is consistently lower on the training dataset compared to the validation dataset and that's explain the gap existed between the train and validation loss learning curves.

In summary, the Encoder-Decoder model's results indicate that it excels in predicting Vitesse_moteur, which is crucial for the extrusion process. However, its performance varies for different extrusion zones, with some zones showing good predictions and others requiring improvement

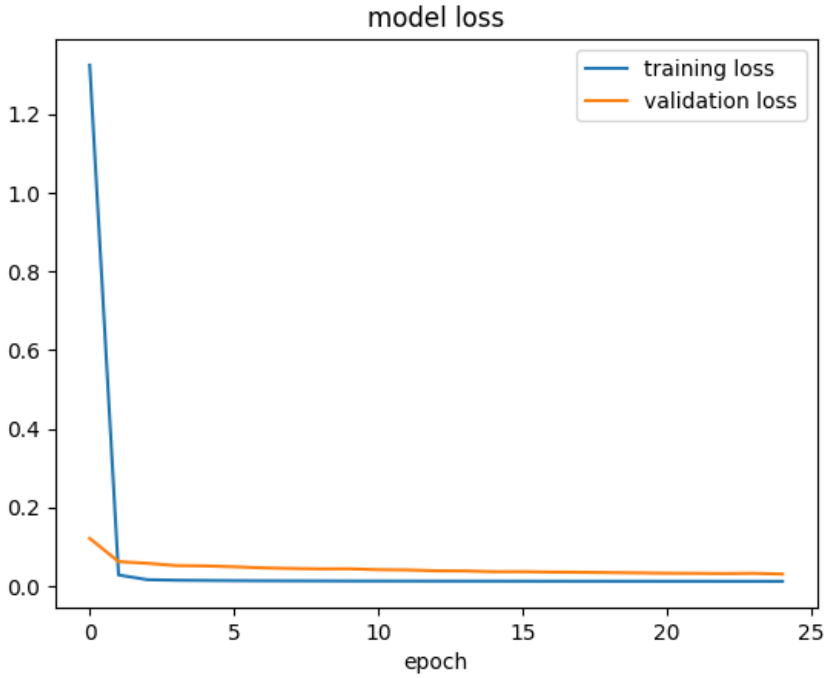


Figure 6.5: Behavior Of Loss Function Over The Training Epochs

6.5 Long Short Term Memory (LSTM) Model With The Attention Mechanism

The mean squared error (MSE) loss curve shown in figure 6.6 shows a smooth slope, indicating that the model improved during training. The decreasing loss values means that the accuracy of the prediction is high and the deviation between the predicted value and the actual target value is effectively minimized, which is a positive outcome. Additionally, the learning curve computed from the holdout validation dataset provides valuable insight into the model's generalization performance. A gently sloping validation learning curve means that the model not only learns from the training data, but also captures important patterns and trends present in the validation data.

The table 6.4 below provides an overview of the model's performance on different features along with corresponding metrics such as R^2, MAE, MSE and RMSE.

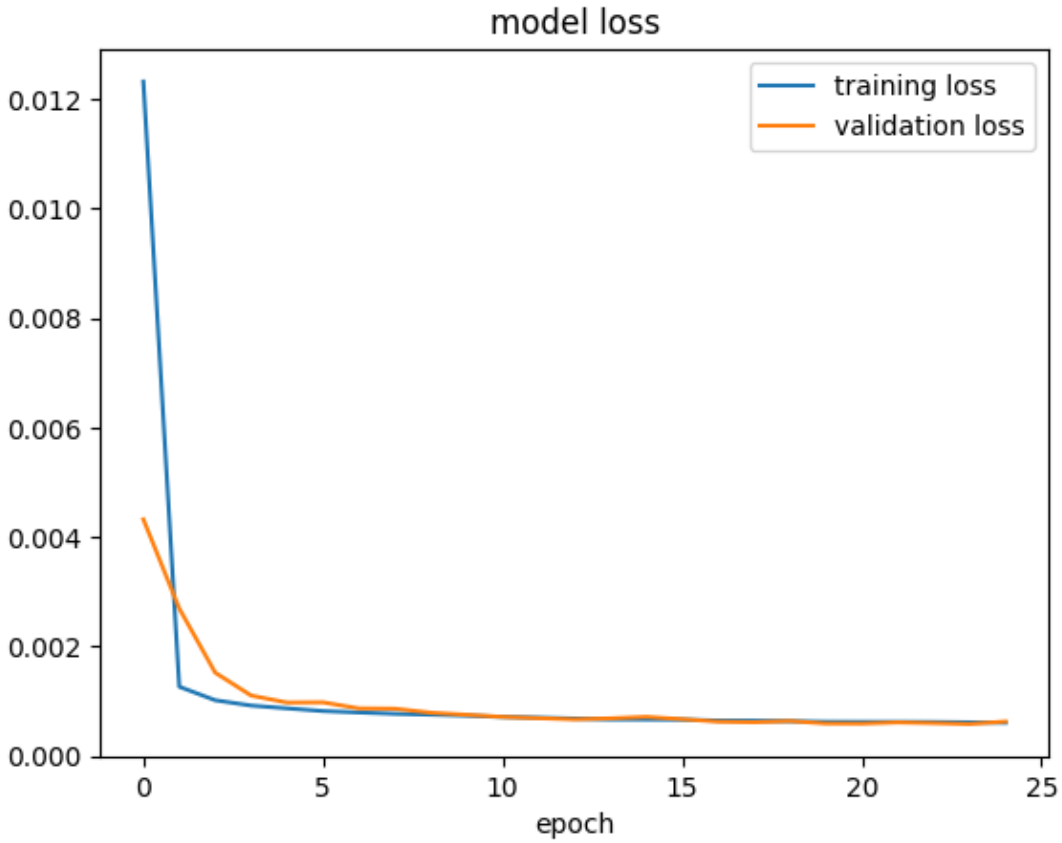


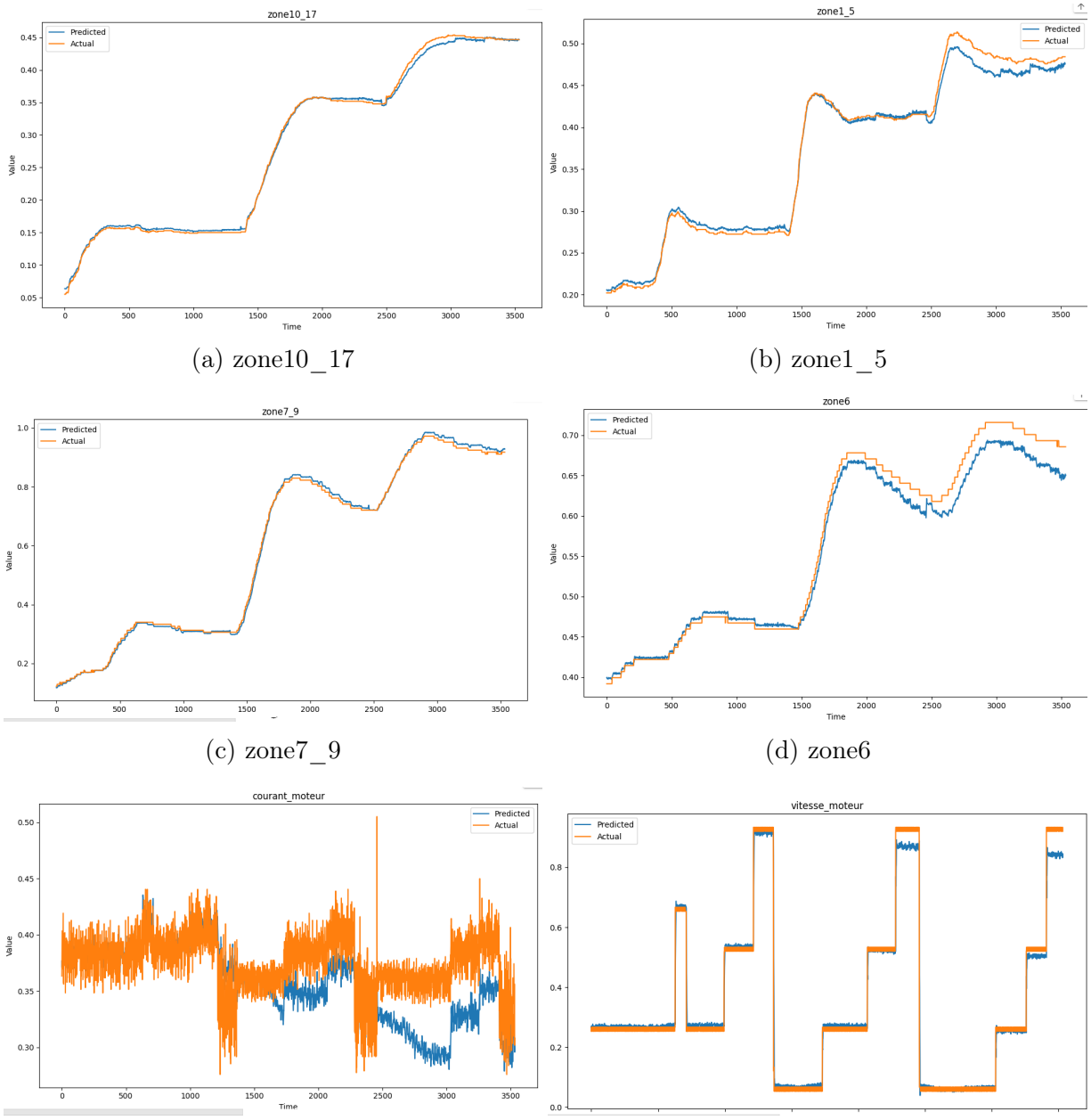
Figure 6.6: Learning Curve

Features & Evaluation metric	R^2	MAE	MSE	RMSE
Courant_moteur	-1.169948	0.029198	0.001347	0.03671
Vitesse_moteur	0.98029	0.018149	0.00148	0.03854
Zone1_5	0.99151	0.0071	8.55×10^5	0.00925
Zone_6	0.97700	0.01362	0.00029	0.01707
Zone7_9	0.99882	0.00839	9.93×10^5	0.00996
Zone10_17	0.99856	0.00396	2.32×10^5	0.00482

Table 6.4: LSTM model with the attenstion mechanism results

The LSTM model performs exceptionally well in predicting Vitesse_moteur and excels in predicting all extrusion zones. The high R^2 value indicates a strong correlation between predicted and actual values. Additionally, the MAE, MSE, and RMSE values are low, suggesting highly accurate predictions.

Finaly, we provide visualizations in 6.7 for a comprehensive assessment of how well the predictions of the LSTM model with the attention mechanism matches the real data where The forecast results (blue line) demonstrate a strong alignment with the original time series (orange line), indicating a good fit between the predicted values and the actual data.



(a) zone10_17

(b) zone1_5

(c) zone7_9

(d) zone6

(e) courant_moteur

(f) vitesse_moteur

Figure 6.7: Plots of the LSTM with the attention mechanism forecasting model

In summary, The LSTM model with an attention mechanism performs exceptionally well for Vitesse_moteur and all extrusion zones. However, it struggles to predict Courant_moteur accurately, as indicated by the negative R^2 value and higher errors.

6.6 Discussion

We compare the performance of the four models (VAR Model, LSTM, Encoder Decoder Model and LSTM with the Attention Mechanism) over the concatenated datasets (Extruder Parameters) over time. We use Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) and Coeffecient Of Ddetermination (R^2) to assess the forecasting performance, which can be obtained as we already explain in the chapter 2.
The table 6.6 and the table 6.5 provide the summary statistics on models performances:

Features	**LSTM with Attention**			**Encoder-Decoder Model**		
	RMSE	MAE	R^2	RMSE	MAE	R^2
$Courant\_moteur$	0.03671	0.02919	-1.16994	0.26042	0.06781	0.37651
$Vitesse\_moteur$	0.98029	50.0476	0.98029	0.14225	0.08940	0.87534
$Zone1\_5$	0.00925	0.0071	0.99151	0.24572	0.14796	0.59293
$Zone\_6$	0.01707	0.01362	0.97700	0.12030	0.05843	-0.52548
$Zone7\_9$	0.00996	0.00839	0.99882	0.07182	0.05870	0.85756
$Zone10\_17$	0.00482	0.00396	0.99856	0.13534	0.07944	0.83036

Table 6.5: Performance Comparison (LSTM with Attention vs. Encoder-Decoder Model)

Features	**LSTM**			**VAR Model**		
	RMSE	MAE	R^2	RMSE	MAE	R^2
$Courant\_moteur$	39.3651	27.8556	0.4524	94.59277	38.73712	-0.06510
$Vitesse\_moteur$	1.6103	1.5001	0.9790	21.19242	13.18129	-0.39681
$Zone1\_5$	1.6103	1.3738	0.9906	7.79329	4.91342	-0.14185
$Zone\_6$	3.0679	2.2123	0.9685	3.84481	2.3767	-0.39964
$Zone7\_9$	0.5666	0.4386	0.9988	4.40258	2.82434	-0.3380
$Zone10\_17$	2.1143	1.7204	0.9809	2.52951	1.45308	-0.21876

Table 6.6: Performance Comparison (LSTM vs. VAR Model)

We can note this results:

1. The "LSTM with Attention" consistently achieves the lowest RMSE and MAE values among all models, indicating superior performance in terms of minimizing prediction errors.

2. The "Encoder-Decoder Model" with LSTM layers shows competitive RMSE and MAE values, suggesting that the LSTM layers in this model contribute to improved predictive accuracy.

3. The "VAR Model" generally exhibits higher RMSE and MAE values compared to the LSTM-based models indicating larger discrepancies between predictions and actual values.

4. Both LSTM-based models (with Attention and without) tend to yield higher R^2 values compared to the "Encoder-Decoder Model" with LSTM layers and the "VAR Model." This suggests that the LSTM models better explain the variance in the data and provide better-fitting models while the "VAR Model" has a poorer fit to the data in terms of explaining variance.

Overall, both LSTM with the attention mechanism and The encoder decoder with two LSTM layers model exhibit notably superior performance compared to LSTM and the VAR model, considering R^2, RMSE and MAE. It's worth noting that the distinct performance hierarchy of the four prediction models ($LSTM\_attention\_mechanism \succ EncoderDecoder \succ LSTM \succ VAR$) proves valuable in establishing a connection between model performance and interpretability.

In short, it is clear that extending the LSTM model with an attention mechanism, emerges as the most appropriate predictive model for the extruder's parameters, especially for motor speed and different heat zones. Its outstanding accuracy, competitive performance, and overall superiority in predictive capabilities make it the best choice for this specific application.

Chapter 7

Conclusion

In Chapter 2, we presented a comprehensive definition of multivariate time series and explored how statistical methods and deep learning approaches have been widely utilized in forecasting them over the past decade. Subsequently, in Chapter 3, we reviewed existing research related to forecasting multivariate time series, focusing on our specific interest in predicting the behavior of parameters in an extruder machine.

In pursuit of this objective, we explored three different solutions and conducted a comparative analysis of the results obtained. Among these solutions, we specifically applied the Long Short-Term Memory (LSTM) model enhanced with an attention mechanism to address our study case. This approach was implemented using a dataset from the recycled plastic extrusion process, which constituted a multivariate time series.

The outcomes of our experimentation with the LSTM model combined with the attention mechanism were exceedingly gratifying, showcasing substantial enhancements in the accuracy and precision of our forecasts. Consequently, this particular model emerged as the most favorable and optimal choice for effectively tackling the forecasting task at hand.

Looking to the future, we envision further advancements in our research. The inclusion of sensors to monitor the viscosity of the final plastic material would be a valuable enhancement to the extrusion process offering a valuable means of quality control and process optimization. Moreover, with the availability of more data from the recycled plastic extrusion process, we plan to develop a new neural network that combines LSTM with other elements to capture dependencies over time. However, we are keen to explore the integration of reinforcement learning methods and the application of Markov chain approaches. This enhanced model will enable us to forecast the attitude of the extruder machine even more effectively, paving the way for continued progress in this domain.

Bibliography

[1] Stationarity in time series analysis explained using python. https://www.quantinsti.com/.

[2] Ajitesh Kumar. Gradient descent explained simply with examples. `https://vitalflux.com/gradient-descent-explained-simply-with-examples/`.

[3] Jimeng Sun Cao Xiao. Recurrent neural networks (rnn). *SpringerLink*.

[4] Attila GerGely Attila Gyárfás. The design of a small-scale plastic extrudermachine. 2019.

[5] Baydaa Ismael Ayat Ahmed Hamel. Time series forecasting using arima model. 2022.

[6] Abdu Mohammed Seid Bayile Damtie Abebaw Bizuneh AlemuU, Jaya Prakash RajuU. Comparative study of seasonal autoregressive integrated moving average and holt-winters modeling for forecasting monthly ground-level ozone. 2023.

[7] Jagjeevan Kanoujiya Rahul Singh Gautam. Multivariate inflation forecasting: A case of vector auto regressive (var) model. 2022.

[8] Abdelmounaim Abdali Ibtissam Amalou, Naoual Mouhni. Multivariate time series prediction by rnn architectures for energy consumption forecasting. 2022.

[9] Xiaoxuan Liu Zijun Fu, Yongming Wu. A tensor-based deep lstm forecasting model capturing the intrinsic connection in multivariate time series. 2022.

[10] Lotfi Najdi Hassan Bousnguar, Amal Battou. Gated recurrent units (gru) for time series forecasting in higher education. 2023.

[11] Kittisa Kerdprasop Pasapitch Chujai, Nittaya Kerdprasop. Time series analysis of household electric consumption with arima and arma models. 2013.

[12] Miguel García Torres Francisco A. Goméz Vela José Luis Vázquez Noguera Federico Divina, ORCID. A comparative study of time series forecasting methods for short term electric energy consumption prediction in smart buildings. 2019.

[13] Duc Do Micheal Hintlian Mahdy Shirdel, Reza Asadi. Deep learning with kernel flow regularization for time series forecasting. 2021.

[14] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. 2021.

[15] Time series analysis and forecasting using auto time series. https://www.section.io/engineering-education/time-series-analysis-and-forecasting-using-auto-time-series/.

[16] Denis Kwiatkowski, Peter C B Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1-3):159–178, 1992.

[17] S. Prabhakaran. Augmented dickey fuller test (adf test) â must read guide," machinelearningplus.com,. november 2, 2019.

[18] var. https://en.wikipedia.org/wiki/Vector_autoregression.

[19] Vector auto-regressive (var) models for multivariate time series forecasting. https://medium.com,.

[20] Venus Khim-Sen Liew. Which lag selection criteria should we employ? *Economics Bulletin 3(33):1-9*, 2014.

[21] Himanshi Singh. Deep learning 101: Beginners guide to neural network, 2021.

[22] Neural networks bias and weights. https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da,.

[23] Jean-Louis Queguiner. What does training neural networks mean? https://blog.ovhcloud.com/what-does-training-neural-networks-mean/.

[24] Mbali Kalirane. Gradient descent vs. backpropagation: What's the difference? https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/.

[25] Knut Hinkelmann. Neural networks, 2018.

[26] Activation functions in neural networks. https://www.geeksforgeeks.org/activation-functions-neural-networks/.

[27] Activation functions in neural networks [12 types use cases]. https://www.v7labs.com/blog/neural-networks-activation-functionsh3.

[28] prakharr0y. Intuition of adam optimizer.

[29] Jimmy Lei Ba Diederik P. Kingma. Adam: A method for stochastic optimization. 2015.

[30] Derrick Mwiti. Getting started with recurrent neural network (rnns). https://towardsdatascience.com/getting-started-with-recurrent-neural-network-rnns-ad1791206412.

[31] How do you deal with the vanishing and exploding gradient problems in rnns? https://www.linkedin.com/advice/3/how-do-you-deal-vanishing-exploding-gradient#:~:text=The%20vanishing%20and%20exploding%20gradient%20problems%20occur%20when%20the%20gradients,information%20from%20previous%20time%20steps.

[32] Jurgen Schmidhube Sepp Hochreiter. Long short-term memory. 1997.

[33] Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/#fn1.

[34] Héctor Corrada Bravo imothy Wood. Improving long-horizon forecasts with expectation-biased lstm networks. 2018.

[35] Demystifying encoder decoder architecture neural network. https://vitalflux.com/encoder-decoder-architecture-neural-network/#What%E2%80%99s_Encoder_Decoder_Architecture_How_does_it_work.

[36] Attention mechanisms in deep learning — not so special. https://medium.com,.

[37] Yoshua Bengi KyungHyun Cho. Neural machine translation by jointly learning to align and translate. 2015.

[38] Christopher D. Manning Minh-Thang Luong, Hieu Pham. Effective approaches to attention-based neural machine translation. 2015.

[39] Hung-yi Lee Shun-Yao Shih, Fan-Keng Sun. Temporal pattern attention for multivariate time series forecasting. 2019.

[40] python. https://www.python.org/.

[41] tensorflow. https://www.tensorflow.org.

[42] keras. https://keras.io/api/.

[43] Soukaina Ouhame and Youssef Hadi. Multivariate workload prediction using vector autoregressive and stacked lstm models. March 28–29,2019.

[44] XIANYUN WEN and WEIBANG LI. Time series prediction based on lstm-attention-lstm model. *IEEE Access*, January 2023.

[45] YAQUN HUANG YUNTONG LIU, CHUNNA ZHAO. A combined model for multivariate time series forecasting based on mlp-feedforward attention-lstm. *IEEE Access*, January 2022.

[46] Lemya Taha. Forecasting time series using vector autoregressive model. 2021.

[47] S Sudibyakto-Aris Poniman Sri Hartini, Muhammad Pramono Hadi. Application of vector auto regression model for rainfall-river discharge analysis. 2015.

[48] Willdan Aprizal Arifin Yoga Estu Nugraha Nugraha, Ishak Ariawan. Weather forecast from time series data using lstm algorithm. 2023.

[49] Jun WangJun Wang HsinChun Chen Qing Li, Jinghua Tan. A multimodal event-driven lstm model for stock prediction using online news. 2020.

[50] Umesh Chandra Pati Mogarala Tejoyadav, Rashmiranjan Nayak. Multivariate water quality forecasting of river ganga using var-lstm based hybrid model. 2022.

[51] Elliana Gautama Harya Widiputra, Adele Mailangkay. Multivariate cnn-lstm model for multiple parallel financial time-series prediction. *Complexity*, October 2021.