SAPIENZA
UNIVERSITÀ DI ROMA

SLinear: Taming the Non-Stationarity in Long Sequence Time Series Forecasting

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Lorenzo De Santis

ID number 1849114

Thesis Advisor

Prof. Danilo Comminiello

Co-Advisor

Redemptor Jr Laceda Taloma

Academic Year 2022/2023

Thesis defended on 26 October 2023
in front of a Board of Examiners composed by:

Prof. Giuseppe Oriolo (chairman)
Prof. Laura Astolfi
Dr. Ala Arman
Prof. Silvia Bonomi
Prof. Danilo Comminiello
Prof. Francesca Cuomo
Prof. Giorgio Grisetti
Prof. Christian Napoli
Prof. Fabrizio Silvestri

SLinear: Taming the Non-Stationarity in Long Sequence Time Series Forecasting
Master's thesis. Sapienza – University of Rome

© 2023 Lorenzo De Santis. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: desantis.1849114@studenti.uniroma1.it

Abstract

Long Sequence Time Series Forecasting (LSTF) represents a fascinating challenge in the field of time series forecasting. With the advent of deep learning (DL), there has been a surge in efforts to develop increasingly sophisticated models to tackle LSTF. This surge has led to a proliferation of DL approaches. However, recent research has revealed an overestimation of the effectiveness of complex deep methods, such as Transformers-based models. Surprisingly, simpler linear-based models have shown superior performance on the LSTF task.

Effective preprocessing and normalization are crucial to ensure that the model remains robust to outliers and non-stationary series. Consequently, numerous studies have focused on normalization techniques in order to construct more powerful models. Unfortunately, much of this research focuses primarily on very deep learning architectures and may not take into account recent advances, such as LSTF-Linear [37].

These findings form the basis of our work, with a particular focus on addressing the non-stationarity inherent in real-world data and highlighting the central role of proper data normalization in improving the predictability of linear-based models. This effort culminates in the development of our innovative architecture, SLinear, which demonstrates a remarkable improvement in both Mean Squared Error (MSE) and Mean Absolute Error (MAE) on multivariate benchmarks.

In addition, our research is concerned with reconsidering the use of Transformer-based models in LSTF tasks, deviating from the conventional end-to-end training approach. Inspired by recent successes in representation learning for time series forecasting, in particular focusing on CoST work [33], a contrastive representation learning framework. Our goal is to identify a transformer-based encoder that enables the model to learn a more powerful latent representation of time series data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Context of the Research

Long Sequence Time Series Forecasting is a very interesting task, intuitively (at least from a relative perspective) we refer to LSTF as long horizon prediction task, it has various practical applications in different areas: economics, finance, energy, healthcare and so on. This kind of prediction problems has been deeply studied in statistics, but, with the developments of deep learning technologies, LSTF gains a great improvement. As a result, a lot of deep neural networks have been studied with the aim of designing better and better models that could achieve incredible prediction accuracy. We need to remember a particular class of models that in recent years, after the introduction of Vaswani et al.,2017[30] work, gained a lot of attention; transformer-based models have been developed and studied in the fields of computer vision and natural language. Due to the nature and characteristic of Transformer, its applications in time series forecasting task become natural. This was a great intuitions because, at least initially, it was outperforming the performance of the other predictor architecture, however some very recent work[37] highlined the lack and limitations of transformer-based models; by simply proposing a set of linear-based models, outperform all SOTA deep architecture in almost all the literature benchmarks. The use of normalization techniques is fundamental for DL models to deal with LSTF problems. in fact, if the data is not properly normalized, the performance of such models can degenerate very quickly. This simple statement inspires a lot of work in LSTF literature that explores and studies normalization techniques and frameworks that could improve the predictability performance, especially in real-word dataset.

1.2 Research Objectives and Questions

The main objective of our research is to develop novel approaches to improve the forecasting performance of linear models in Long Sequence Time Series Forecasting. This involves using unconventional normalization techniques in a way that enhances their effectiveness, especially when dealing with non-stationary real-world datasets. Our purpose is to augment these simple linear models by integrating tailored normalization modules and techniques, while exploring various alternatives.
While transformer-based models have been extensively studied in the field of time series forecasting, their application has largely been in the form of stand-alone architectures. This leaves untapped potential in various contexts, including improving the encode within representation learning frameworks. Consequently, we seek to explore this avenue and highlight the capabilities of transformer-based models in this alternative context.

In summary, our research attempts to answer key questions that have profound implications for the field of LSTF. Could the future of LSTF rely in model simplicity? What are the most effective normalization techniques in the context of LSTF? Is the performance of transformer-based models as remarkable as is commonly believed? These key questions will guide our research and contribute to a deeper understanding of forecasting methods.

1.3 Significance of the Study

This study is significant because it deals with an important subject, the prediction of the future, which has always fascinated researchers and not only. Being able to find a fresh and simple model, that is lighter to train, can learn more meaningful latent variables within a historical series, can lead to a better predictive model. If we think of the possible applications, they are unlimited, just think of how increasingly important it will be to be able to understand the energy needs of a city, so that infrastructures can be adapted to cope with an increasingly technologically advanced society (e.g. the management of charging stations for increasingly energy-intensive cars). This is just one example, the ability to predict a more distant future is crucial for all possible areas of application of the time series forecasting. This work is intended to help LSTF community researchers by giving them a good starting point and some direction for further work in these areas. All implementations are available in this repository [1].

[1] `https://github.com/whoami-Lory271/SLinear-Time-Series-Forecatsing`

1.4 Thesis Structure

In this work, we embark on a comprehensive exploration of the outlined objectives and questions. The thesis unfolds through the following structured chapters:

Chapter 2 This chapter provides a solid background by explaining essential concepts of time series analysis and deep learning. It provides the reader with the necessary knowledge to effectively understand our work.

Chapter 3 This provides a focused overview of the complexities involved in predicting long sequence time series. It outlines the challenges and implications involved, and reviews the existing literature to provide context.

Chapters 4 and 5 These chapters introduce our novel methods, provide a detailed exposition of the architectures and illustrate the results obtained.

Chapter 6 This part explores the potential applications of transformer-based models when used as encoders within representation learning frameworks, extending the scope of our investigation.

Chapter 7 We provide a careful analysis of the results and assess their practical implications in the field of long sequence time series forecasting. In addition, we outline potential directions of this work for future research and extensions.

Chapter 2

Background

In the field of data analysis, time series occupy a unique and invaluable place, providing a lens through which we can understand the intricate interplay of variables over time. In this chapter, we'll embark on a journey through this temporal landscape. We'll explore the methods and importance of time series analysis. We'll then look at the main applications and use cases, also introducing the problem of long-term time series forecasting.
We'll then briefly explore neural networks and deep learning models, analysing their basic principles, architectures and methodologies. We will also explain how these models learn and make predictions.

2.1 Time Series and Analytical Techniques

2.1.1 Time Series Definition

A *time series* in mathematics is delineated as a sequentially ordered sequence of data points, wherein these points are indexed in chronological order and exhibit uniform temporal spacing between each successive value. The representation of time series through visualizations such as *run charts*, scatter plots, and line graphs allows for a clear depiction of temporal patterns.

Figure 2.1. Run Chart Example

2.1.2 Time Series Use Cases

Time series analysis is widely used in a variety of fields where the sequential nature of data points provides invaluable insights into dynamic processes and phenomena.

Economic and Financial Time Series

One of the most common applications of time series analysis is in finance and economics. Forecasting stock prices, predicting economic indicators, and modelling market trends are central operations that rely heavily on understanding and modelling time-dependent data. Accurate predictions in these areas are essential for investment strategies, risk management and policy formulation.

Physical Time Series

In the physical sciences, time series analysis plays a crucial role, particularly in meteorology, geophysics and marine science. Monitoring several factors such as precipitation, temperature variations and other climatic factors is crucial for climate change research, natural disasters and sustainable resource management.
Certain analog sensors work by measuring variables continuously, as opposed to taking discrete measurements at specific time intervals. In laboratory environments, for example, it is critical that certain factors remain within defined thresholds. Continuous monitoring provides a visual overview of trends and variations. However, for more rigorous and detailed analysis, it is beneficial to convert this continuous curve into a discrete time series. This is achieved by sampling the data at equally spaced intervals, allowing more precise examination and analysis.

Marketing Time Series

The analysis of time series is an important problem in commerce. Observed variables might include sales figures in successive weeks or months, cash receipts, advertising costs and so on. Note the trend and seasonal variation typical of sales data. It is often important to forecast future sales in order to plan production. It may also be of interest to examine the relationship between sales and other time series, such as advertising costs.

Process Control Data

A critical challenge in process control is to detect changes in the performance of a manufacturing process. This is achieved by monitoring a variable that serves as an indicator of process quality. If these measurements deviate significantly from a pre-defined target value, this signals the need for immediate corrective action to

maintain optimum process control and maintain product quality standards. This practice is fundamental to ensuring consistent and reliable manufacturing operations.

Time Series in Smart Grid

Smart Grids represent an evolution in power distribution systems, enabling consumers to actively participate in the production, storage and distribution of energy. The management of smart grids presents a number of challenges, the most important of which are the dynamic changes and advances in energy consumption and production. It is imperative to effectively manage these changes in order to efficiently regulate energy distribution and ensure the resilience and sustainability of the grid infrastructure.

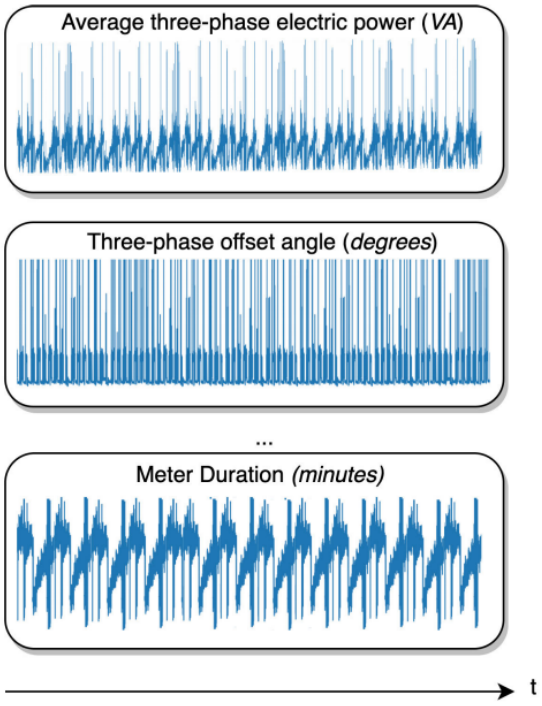


Figure 2.2. Multiple time series corresponding to real data observed in a power grid

2.1.3 Terminology

A time series is classified as *continuous* when observations are collected continuously over a period, whereas it is considered *discrete* when observations are taken at specific, typically evenly spaced, intervals.
In this study we'll focus mainly on the discrete case, which can occur in a number of ways. In a continuous time series, values can be read at equal intervals, the

resulting series is called a *sampled* series, the sampling interval must of course be carefully chosen. Another form of discrete series emerges when a variable lacks an instantaneous value, but its values can be aggregated or accumulated over uniform time intervals. Instances of this category include monthly exports and daily rainfall totals. Lastly, certain time series inherently exhibit a discrete nature, for example the dividends disbursed by a company to its shareholders in consecutive years.
An important distinction in time series analysis is the dependence of successive observations, in contrast to much of statistical theory which typically deals with random samples of independent observations. This temporal dependence requires consideration of the temporal order of the observations. In cases where observations are interdependent, it becomes possible to make predictions about future values based on past observations. A time series is said to be *deterministic* if its future values can be accurately predicted. However, most time series are *stochastic*, which means that the future is only partially determined by past values, making accurate predictions impossible. Instead, the concept of a probability distribution for future values, conditioned on knowledge of past values, becomes central to modelling such series.

2.1.4 Objectives of Time Series Analysis

The main aims of analysing time series can be classified as follows.

Description

When faced with a time series data set, the first step in analysis is usually to plot the observations against time, resulting in what is known as a *time plot*. This is usually followed by the calculation of simple descriptive statistics to gain insight into the main characteristics of the series.
For some series, variations are driven primarily by immediately identifiable patterns. In such cases a relatively simple model focusing on trend and seasonal variation may be sufficient to accurately characterise the variation in the series. However, for other series with more complex patterns, the use of more advanced techniques becomes crucial. The use of a graph is particularly useful not only in illustrating the trend and seasonality of the time series, but also in highlighting *outliers* that deviate significantly from the general pattern of the data. The treatment of outliers is extremely important as it allows us to better describe the time series. For example, they may indicate that the data is not normally distributed, or that a sensor is not working properly.

Explanation

When observations are collected over time for two or more variables, there is the potential to use variations in one time series to explain variations in another. This approach can provide a deeper understanding of the underlying mechanisms that generate a particular time series. While regression models can provide some insights in such cases, they are not inherently designed to deal with time series data with their inherent correlations and dependencies. Consequently, it becomes necessary to explore and consider alternative categories of models in order to effectively analyse and capture the complex dynamics present in time series datasets.

Forecasting

A key aspect of our study is the prediction of future values within a time series based on observed data points from the past. This prediction task is of significant importance in various applications and will be a central focus of our research.

Control

Time series data is collected with the primary objective of improving process quality by monitoring specific parameters. In manufacturing, for example, careful monitoring is used to maintain quality standards. A more sophisticated approach involves the construction of models that allow an optimal strategy to be continuously adopted based on the data available. This advanced method allows dynamic adjustments and fine-tuning of processes to achieve and maintain the highest levels of quality and efficiency.

2.1.5 Basic Descriptive Techniques

The range of statistical techniques available for analysing time series data extends from the simple to the highly complex. It is essential to start with descriptive methods as they play a crucial role in data cleaning and initial exploration. In addition, the consideration of context is paramount in time series analysis. Acquiring the appropriate background knowledge is essential to effectively deal with the specific problem at hand. In this section our attention will be focused on the investigation of prominent time series effects. This includes the examination of trends, seasonal patterns and the assessment of correlations between successive observations.

Types of Variation

A time series can be broken down into its main components, which represent the trend, seasonal variations and other cyclical changes. The remaining irregular

fluctuations can be classified as an error component. This method is particularly effective when the variation is dominated by trend and seasonality. Of course, decomposition into such components is not unambiguous, unless some assumptions are made, some modelling is required.
The different sources of variation can be classified as follows

Seasonal Variation Many time series show annual variation within their respective periods. This seasonal variation is relatively easy to understand and to estimate, especially when the focus of the analysis is on seasonality.

Other Cyclic Variation In addition to seasonal effects, certain time series show variations at fixed intervals due to different physical factors. A good example is the daily variation in temperature. In addition, some time series have fluctuations that do not have a fixed period but are still predictable to some extent. For example, economic data are often considered to be influenced by business cycles.

Trend This concept can be broadly described as a *long-term shift in average levels*. However, the challenge is to define precisely what is meant by long-term. For example, climate variables may show cyclical variations over a long period of time, such as 50 years. If a dataset of only 20 years were analysed, this prolonged oscillation could be mistaken for a trend. Conversely, with several hundred years of data, the true long-term cyclical variation would become visible.

Other Irregular Fluctuations The final component of a time series consists of random residuals rather than predetermined patterns or trends. These residuals represent the unexplained variability in the data, typically due to stochastic or unpredictable factors, but they can also come from a deterministic source.

Stationary Time Series

A time series can be said to be stationary if it shows no systematic change in its mean (i.e. no discernible trend) or variance, and if strictly periodic variations have been attenuated. In essence, this means that the statistical properties observed in one segment of the data reflect those of any other segment. A considerable part of probability theory is concerned with the analysis of stationary time series. Therefore, the transformation of a non-stationary time series into a stationary one is essential for the effective application of this theoretical framework.

Transformations

There are three main reasons for transforming data.

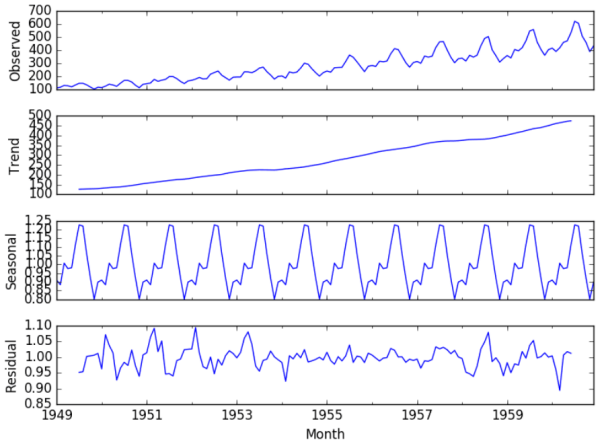


Figure 2.3. Observed data describe the total number of airline passengers over a period of time, the original time series is decomposed in trend, seasonal and residual component

Stabilize the Variance If a trend is identifiable in the time series and there is a corresponding increase in variance alongside the mean, it may be advantageous to consider data transformation. However, if the variance fluctuates over time without an accompanying trend, it is unlikely that the application of transformation will lead to significant improvements.

Make the Seasonal Effect Additive If a trend exists and the seasonal effect's size also increases with the mean, it may be beneficial to transform the data such that the seasonal effect remains constant. If the size of the seasonal effect is proportional to the mean, it is called multiplicative, whereas if the size of the seasonal effect is constant, it is called additive.

Make the Data Normally Distributed The probabilistic model used to forecast and generate a time series is frequently grounded on the presupposition that the data conforms to a normal distribution. In practice, achieving this can be challenging. For instance, if a time plot displays a trend characterized by spikes all in the same direction, eliminating this effect solely through transformation is difficult. In such cases, it may be beneficial to use an alternative error function.

It may not always be feasible to meet all the aforementioned requirements at the same time, and the transformed model may be more difficult to analyse and forecast, and may need to be transformed back, introducing bias. In general, it is advisable to minimize transformation unless transformed variables have a direct physical interpretation.

2.1.6 Time Series Analysis

The apparent correlation introduced by the sampling of adjacent points in time can severely limit the applicability of many conventional statistical methods, which traditionally rely on the assumption that these adjacent observations are independent and identically distributed. The systematic approach to answering the mathematical and statistical questions raised by these time correlations is commonly referred to as time series analysis.
Time series problems can arise in many fields: in economics, we can think of stock market quotations; an epidemiologist might be interested in the number of influenza cases observed over a period of time; in medicine, functional magnetic resonance imaging of brain wave time series patterns could be used to study the behaviour of the brain under certain conditions.
To go deeper into time series analysis, we need to mention two approaches to time series analysis: *Time Domain* approach and *Frequency Domain* approach.

Time Domain Approach

The rationale behind the time domain approach rests primarily on the assumption that the correlation between adjacent points in time is best explained by the interdependence of the current value with its preceding values. This approach focuses on formulating a predictive model for a future value within a time series, based on parametric functions that include both current and past values. Consequently, the first step is to construct linear regression models in which the current value of a time series is regressed against its own historical values as well as the previous values of other related series. This modelling approach then enables analysts to use the results of the time series approach as a forecasting tool, and it is for this reason that it is particularly popular.
One approach involves the development of a systematic class of models known as autoregressive integrated moving average (ARIMA) models, designed for effective modeling and forecasting. This model exhibits the capability to handle multiple input series through a multivariate ARIMA approach. A distinctive characteristic of these models is their multiplicative nature, implying that the observed data are presumed to arise from the multiplication of factors that incorporate differential or difference equation operators responsive to a white noise input. In this modern paradigm, time series analysis takes a more nuanced perspective. It involves decomposing the data into distinct components, each of which contributes in a distinct way to the overall pattern. Specifically, this approach implies that a given time series can be expressed as the cumulative effect of several well-defined components. First, the presence of a trend component is recognised. This element encapsulates the long-term directional

movement inherent in the data, reflecting its underlying trajectory over a longer period. A seasonal component is also considered. This component takes into account recurring patterns or fluctuations that occur within specific time intervals, such as daily, monthly or annual cycles. Finally, an error term is introduced to capture any stochasticity in the data, a proper treatment of this error term is essential for accurate modelling and forecasting. By adopting this approach, analysts can gain a more complete understanding of the underlying structures and patterns within the time series data.

Frequency Domain Approach

In contrast, the frequency domain approach assumes that the fundamental features of interest in time series analysis are periodic or systematic sinusoidal variations that are inherently present in most data. This approach aims to extract and analyse the underlying frequency components that contribute to the observed patterns in the data. In spectral analysis, the process of separating the different types of periodic variation within a time series involves the individual estimate of the variance associated with each specific periodicity of interest. The resulting distribution of variance across frequencies is referred to as the power spectrum. This analytical technique serves as a valuable tool for uncovering the dominant frequency components that contribute to the overall behaviour of the time series.

2.1.7 Forecasting

Forecasting the future values of an observed time series is a crucial task in many fields, including economics, energy production, sales, and stock control.
Suppose you have a time series $x_1, ..., x_n$. The task is to estimate future values, such as x_{N+h}, where h is known as the forecasting horizon. The predicted value of x_{N+h}, made at time N, for h steps in the future, is usually referred to as $\hat{x}_N(h)$.
A multitude of forecasting methods are obtainable and it's crucial to note that no single procedure is universally applicable. Instead, the analyst should opt for the most fitting method for the given set of circumstances. It should also be kept in mind that forecasting is a form of extrapolation that involves potential risks. Forecasts are statements dependent on specific assumptions about the future. Therefore, forecasts are not absolute and should always be subject to revision based on external information. For long-term forecasting, it may be useful to present a series of forecasts based on different sets of assumptions so that alternative scenarios can be explored.
We can classify these forecasting methods as follows.

Univariate

The forecasting of a variable will rely exclusively on current and past observations of that variable, so that $\hat{x}_N(h)$ will depend only on the values of $x_n, x_{N-1}, ...$, possibly augmented by a simple function of time. This means that, without taking into account any other economic factors, the future sales of a product are estimated entirely on the basis of past sales.

Multivariate

Forecasts for a given variable are dependent, at least in part, on the values of one or more additional series, referred to as predictor or explanatory variables. Sales forecasts, for instance, might rely on stocks and/or economic indicators.

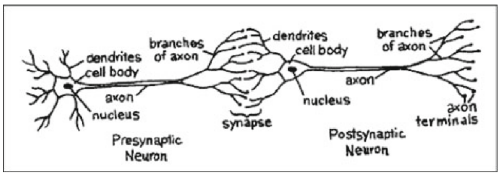
In practice, a forecasting procedure could potentially involve a combination of the aforementioned methods. Marketing forecasts, for instance, are commonly made by combining statistical predictions with subjective knowledge and insight from market professionals.

2.2 Neural Networks and Deep Learning

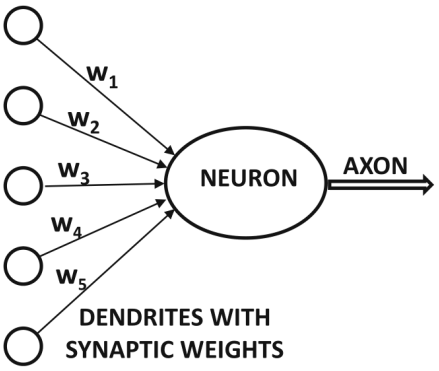
In recent years, the field of artificial intelligence has undergone a transformative revolution, propelled by the unprecedented advancements in neural networks and deep learning. This section explores the fundamental principles, architectures, and methodologies that underpin these powerful computational models. With the rise of deep learning, the integration of multiple layers in neural networks has unlocked a new dimension of complexity and capability, allowing the extraction of intricate features from raw data.

2.2.1 Introduction

Artificial neural networks are widely used machine learning methods. They are named for their structural analogy to biological neural networks. The human nervous system is composed of neurons, which communicate with each other via axons and dendrites in connecting regions. The regions linking these latter are known as synapses, as demonstrated in Figure 2.4a. This mechanism is simulated in an artificial neural network, which comprises computational units known as neurons. These units are connected to one another through weights, which simulate the synaptic connections' strength in the biological network. The artificial architecture is illustrated in Figure 2.4b.



(a) Biological Neural Networks



(b) Artificial Neural Networks

Figure 2.4. Biological and Artificial Neural Networks schema

An artificial neural network operates by computing functions of the input through the propagation of values from input neurons to output neurons, using weights as intermediary parameters. The learning process involves adjusting the network's weights, achieved by providing the architecture with input-output pairs of the function to be learned. For instance, in image recognition, the training data may consist of pixel values (input) along with corresponding output labels describing the image. These labels serve as a benchmark to evaluate the accuracy of the neural network's weights, based on how closely the predicted output aligns with the actual output. The iterative adjustment of the weights follows a mathematically justified approach, enabling the network to make increasingly accurate predictions.
With sufficient training, for instance, by exposing the neural network to numerous images of a dog, it eventually gains the capability to identify a dog in an image it has never encountered before. This capacity to provide accurate predictions for unseen inputs is referred to as *model generalization*, a critical attribute for a neural network model to effectively perform in real-world applications.
Looking at the most basic unit of computation in a neural network, we can recognize the inspiration from traditional machine learning algorithms such as *least-squares regression* and *logistic regression*. The true strength of a neural network lies in its ability to integrate these basic computational components and learn them collectively

with the aim of reducing prediction error. Of course, by combining simpler units, the expressiveness of the resulting model will increase, enabling it to learn more complex functions. Furthermore, sufficient training data is required to learn the weights of a large model, this means that deep learning models are increasingly more attractive when sufficient power and data are available.

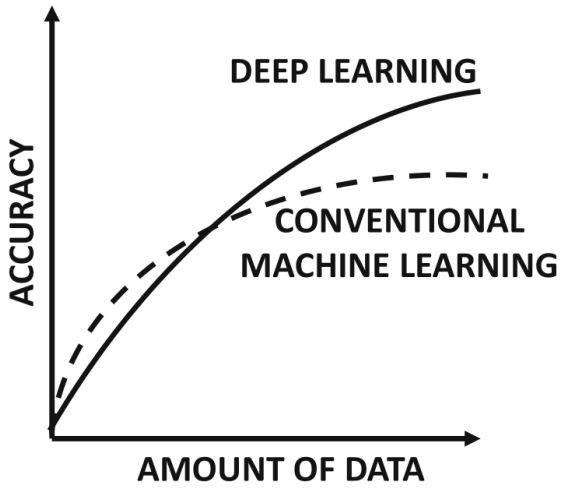


Figure 2.5. An illustrative comparison of the accuracy of a typical machine learning algorithm with that of a large neural network

2.2.2 Single Computational Layer: The Perceptron

The *perceptron* is the primary unit of neural networks, comprising a single input and output node. A diagram showcasing this fundamental structure is provided below, in Figure 2.6. We denote each training instance as $(\bar{X}, y)$ within the context of our assumptions, with $X = [x_1, ..., x_d]$ representing a group of d variables. The binary class variable's observed value is denoted as $y \in -1, 1$. The main goal is to create a flexible model that can predict values beyond those explicitly given in training data.

The input layer consists of d nodes, each representing a distinct feature $\bar{X} = [x_1, ..., x_d]$. These features are then transmitted to a single output node through edges weighted by $\bar{W} = [w_1, ..., w_d]$. All computations are concentrated at the output node. To be more precise, a linear function $\bar{W} \cdot \bar{X}$ is applied to produce a real number, which is then fed into a sign function, resulting in the predicted value $\hat{y}$.

The resulting formula is the following:

$$\hat{y} = sign\{\sum_{j=1}^{d} w_j x_j\} \tag{2.1}$$

The sign function maps a real value to either 1 or -1. Therefore, the resulting error in the prediction is given by $E(\bar{X}) = y - \hat{y}$.
The perceptron can be conceptualised as a fundamental computational unit, providing the foundation to assemble numerous components for constructing a more resilient model. As we have seen, the sign function serves as an activation function, mapping an aggregated value to a class label. This aspect of the perceptron highlights the importance of activation functions in the model.
Furthermore, we can investigate the version of the perceptron that incorporates bias. This involves incorporating an invariant component, called bias, into the prediction process. This approach proves particularly advantageous when handling highly asymmetric binary class distributions. By incorporating the bias, we effectively consider this invariant component of the prediction. Mathematically, bias can be represented as the weight associated with a dedicated *bias neuron*. This neuron transmits a value of 1 to the output node, enabling the seamless incorporation of biases into our model.
The perceptron's main aim is to reduce prediction errors, often by applying methods such as least squares over all training instances within the data set D, the minimization function is the following:

$$\min_{\bar{W}} L = \sum_{(\bar{X} \in D)} (y - \hat{y})^2 = \sum_{(\bar{X} \in D)} (y - sign\{\bar{W} \cdot \bar{X}\})^2 \tag{2.2}$$

The function we aim to minimise is known as the *loss function*. It is important to note that least-squares regression is suitable for target variables with continuous values. Therefore, the related loss function is also continuous.
However, a challenge arises when using the least square form of the objective function. The sign function used is non-differentiable and holds a constant value for a substantial part of the domain. This characteristic results in a loss surface that has a staircase structure, which is suboptimal for gradient descent. To deal with this, we can construct a smooth approximation of the gradient. This alteration enables more efficient optimisation during the training process.

$$\nabla L_{\text{smooth}} = \sum_{(\bar{X} \in D)} (y - \hat{y})\bar{X} \tag{2.3}$$

The neural network training algorithm usually follows a sequence of steps: The input data ($\bar{X}$) is fed sequentially, either individually or in small batches, into the network that generates the prediction $\hat{y}$. The weights $\bar{W}$ are adjusted according to

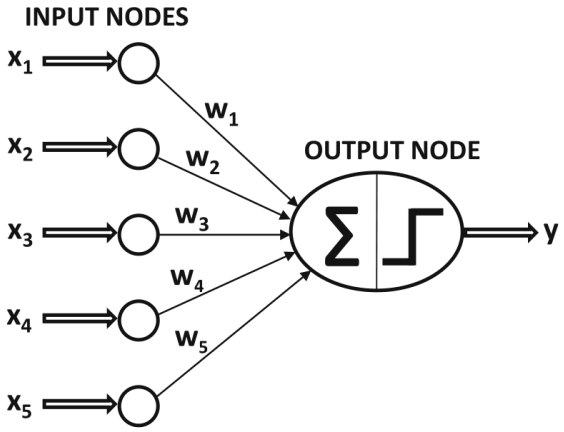


Figure 2.6. Perceptron architecture without bias

the specific error value $E(\bar{X})$.

$$\bar{W} \leftarrow \bar{W} + \alpha(y - \hat{y})\bar{X} \tag{2.4}$$

The parameter α governs the learning rate of the neural network. The perceptron algorithm functions by iteratively cycling through all the training samples in a random fashion, adjusting the weights until convergence is achieved. Each complete cycle through the training data is commonly known as an epoch. This specific perceptron algorithm is often termed the *stochastic gradient-descent* method, owing to its characteristic of randomly selecting training points for weight adjustments. The perceptron operates primarily as a linear model and performs well when processing data that is linearly separable. However, its effectiveness decreases when presented with datasets similar to the one shown in Figure 2.7. This limitation highlights the need for more complex neural networks.

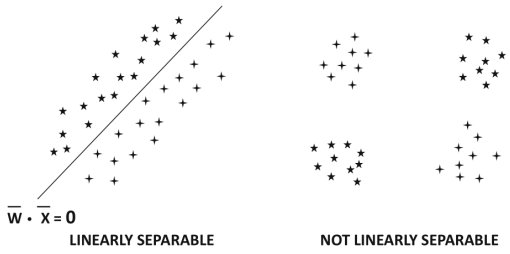


Figure 2.7. Example of linearly and non linearly separable data sets

2.2.3 Multilayer Neural Networks

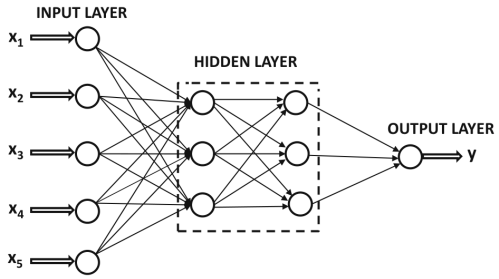
Up to this point, we have examined the operation of the perceptron, which is comprised of a single input and output layer. In this architecture, all computations

are transparent and discernible to the user, as only the output layer is actively involved in the computation.

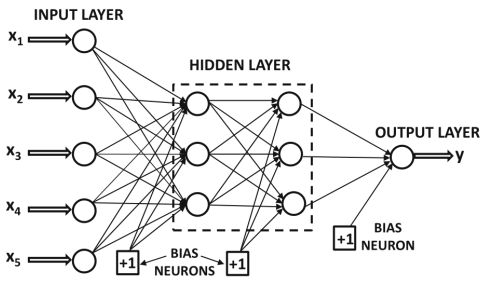
Building on this basic architecture, we encounter *multilayer neural networks*, which include multiple computational layers, including intermediate layers, called *hidden layers* due to their hidden computations. This architecture, which is referred to as a *feed-forward* neural network, demonstrates a sequential flow of information from one layer to the next.

In our exploration of the perceptron algorithm, we have discussed the use of the perceptron criterion. It is important to acknowledge that alternative options exist. In cases of discrete predictions, it is common to use *softmax* outputs in combination with the *cross-entropy* loss function. Conversely, objective measures are typically utilized when making real-valued predictions, utilizing a linear output and squared loss. This principle is applicable even to singular layers.

When dealing with multiplayer networks, the decision to incorporate bias neurons is a crucial factor to consider. As illustrated in the Figure 2.8, such kind of network can be used with or without bias neurons. The dimensionality of a layer refers to



(a) No bias network



(b) With bias network

Figure 2.8. Examples of Multilayer architecture

the number of units in that layer, furthermore, it is typical to represent all units within a single layer as a single vector.

2.2.4 Practical Issues in Neural Networks Training

There are a number of challenges associated with practical issues during the training of neural networks, these are the most important.

The Problem of Overfitting

The problem of *overfitting* is a result of the observation that the performance of a model on a specific training dataset does not necessarily guarantee an equally good performance on an unseen test dataset. This inconsistency in performance between the training and testing phases is likely to be more pronounced when dealing with more complex models and limited datasets. Models with a significant number of parameters, commonly known as high-capacity models, require a larger dataset to ensure robust generalization to unseen test data.
Neural networks are considered universal approximators of any function, but their efficiency can be compromised by limited data availability. The recent increase in data availability has exposed the benefits of neural networks over conventional machine learning algorithms. However, to avoid overfitting, neural network design must be approached with care. We will outline the most important design methodologies used to address this kind of issue.

Regularization The risk of overfitting is a prominent concern when dealing with large numbers of parameters. A common strategy is to apply constraints that limit a model's use of non-zero parameters. The overfitting can also be mitigated by reducing the absolute values of the parameters. However, directly imposing constraints on parameter values can be a challenging task. A more nuanced approach involves augmenting the loss function with a penalty term, denoted by $\lambda||\bar{W}||^2$.
Regularization is particularly important in scenarios where data availability is limited. It provides a means to leverage complex models, even in situations where data volume is lower. This, in turn, supersedes the necessity of resorting to simpler models, affirming the effectiveness of regularization in achieving robust and accurate modelling results.

Parameter Sharing An optimal method for constructing a neural network involves incorporating domain-specific knowledge into the model architecture. For instance, in natural language processing, words in a sentence frequently have sequential correlations, as do adjacent pixels in an image. Designing an architecture that captures these connections may require parameter sharing, resulting in a model with fewer parameters.

Early Stopping *Early stopping* can be considered a type of regularization technique, where the process of gradient descent is terminated after a limited number of iterations. One approach to implementing this strategy is to allocate a portion of the training data as a validation data set, and evaluate the model error on this set at the end of each epoch. The training phase is finished once the validation error starts to increase.
From a theoretical standpoint, this technique reduces the dimensionality of the parameter space. Therefore, early stopping acts as a regularizer, preventing overfitting and facilitating a more generalizable model.

Treading off Breadth for Depth When a neural network has more layers, which signifies an increased depth, there is a tendency to decrease the number of units per layer. Paradoxically, this reduction in unit count enhances the model's effectiveness. Increasing the number of layers acts as a form of regularization as it forces each layer to conform to a specific structure prescribed by the preceding layers. This architectural limitation leads to a network with more layers having considerably fewer parameters, even with the total parameters accounted for across the higher number of layers.
While deep neural networks demonstrate increased resilience to overfitting, they face a unique set of challenges during the training process. Specifically, the derivatives of the loss with respect to the weights display significant differences in magnitudes. This inconsistency creates difficulties in determining the correct step size, leading to problems with *vanishing* and *exploding* gradients. Furthermore, deep neural networks require a significant amount of time to achieve convergence. The complex structure and substantial parameter space can prolong the convergence process, necessitating careful consideration and potentially greater computational resources.

Dropout *Dropout* is a recently developed algorithm used for training neural networks. It relies on the stochastic process of dropping out neurons during the training phase. This random dropout is used to mitigate the co-adaptation of feature detectors, thereby increasing the robustness and generalisation capabilities of the model. This innovative method of training incorporates a random element, which diversifies the learning process and enhances the model's adaptability to a variety of input data.

The Vanishing and Exploding Gradient

While deeper neural networks offer increased capacity for learning complex features, they present a challenge in the form of the backpropagation problem, which is primarily due to the instability of the updating process. In particular, updates in

earlier layers can either become extremely small, leading to the problem of *vanishing gradients*, or, conversely, become extremely large, leading to the problem of *exploding gradients*.
To understand the underlying causes of these phenomena, we can examine the local derivatives along a given path, which result from the product of the weight and the derivative of the activation function. This analysis leads to two possible scenarios:

- If the expected value is less than one, the product of these derivatives will decrease exponentially with path length.
- Conversely, if the expected value is greater than one, it can trigger an explosion of the gradient.

This problem is characteristic of deep neural networks and is a significant challenge in achieving stable training processes. A number of solutions have been proposed to address this issue, including the implementation of different activation functions (e.g. sigmoid or ReLU), the adoption of adaptive learning rates, and the use of recent techniques such as batch normalization.

Difficulties in Convergence

Fast convergence is difficult to achieve with deep neural networks, because the resistance of the network to smooth out the gradient increases with depth. Some tricks have been proposed, such as gating networks and residual connections.

Local and Spurious Optima

The inherent high non-linearity of deep neural networks can lead to a challenge known as getting stuck in local minima during the training process. To mitigate this problem, a strategic approach known as pre-training is used.
The basic concept is to train individual layers of a shallow subnetwork one at a time to determine the optimal initialization point for that particular layer. By carefully selecting the starting point of the network, the researchers aim to guide it towards a more favourable optima. This process of unsupervised pre-training yields an initialization point that is closer to the optimal solution when applied to test data, as opposed to focusing only on training data.
This approach introduces a sophisticated concept of optimization that goes beyond the conventional paradigm that focuses primarily on the shape of a loss function within the training data. Interestingly, the challenge of local minima, while not entirely absent, is of less relevance in the context of such a highly non-linear function. More often, it is the non-linearity itself that poses challenges during the training

process, leading to problems such as failure to converge, rather than getting stuck in local minima.

Computational Challenges

Another problem with neural networks is the amount of time it takes to train a network; it is not uncommon for a neural network to take several weeks to train. Recent technological advances in computing power, such as advances in hardware components like GPUs, have speeded up the typical process of training a neural network. Some frameworks, such as Torch, are particularly convenient because they have GPU support built directly into the platform.

2.2.5 Setup and Initialization Issues in Training

The successful deployment of a neural network depends on a careful setup process that involves a number of critical considerations. Among these, *hyperparameter* tuning, *feature preprocessing*, and *model initialization* are the basis of a well-structured and effective neural network. Each of these elements plays a central role in shaping the network's performance, influencing its ability to learn complex patterns, and ultimately determining its effectiveness in tackling the task at hand. In this section, we examine the details of these critical components and provide insights and strategies for their optimal implementation.

Tuning Hyperparameters

In neural networks, the term *hyperparameters* refers to a set of parameters that govern the architecture and behaviour of the model. These include critical factors such as learning and regularization rates, as distinct from the parameters that represent the weights of the connections within the neural network. This distinction highlights the existence of two distinct categories of parameters, each requiring a tailored initialization approach.
Before backpropagating to optimize the primary parameters, it is essential to set the hyperparameters. This can be achieved either through manual configuration or through a dedicated tuning phase. To effectively adjust the hyperparameters, a separate dataset is used, which is different from the one used for the gradient descent process. This validation dataset serves as a critical tool to evaluate the performance of the model under different hyperparameter configurations, thereby mitigating the risk of overfitting to the training data.
A commonly used strategy for hyperparameter tuning is the grid search method. This approach involves selecting a predefined set of values for the hyperparameters and systematically testing their performance. However, it's worth noting that in

its simplest implementation, the computational time grows exponentially with the number of parameters, making it potentially impractical for complex models. To overcome this challenge, an alternative approach is to first perform a coarse grid search. Once a promising interval is identified, subsequent iterations can use finer grids to find a better value. This alternative may still be expensive, so a random sampling of the hyperparameters should also be considered. However, this method is too slow to be used practically in a large neural network and is still the subject of research.

Feature Preprocessing

Neural networks can also use the same pre-processing of machine learning algorithms, such as:

Additive preprocessing and mean-centering It may be useful to mean-centre the data to remove certain types of bias effects by subtracting a column-wise mean vector from each data point. A second type of preprocessing is used when we want all non-negative values for the features, in this case the absolute value of the most negative entry of a feature is added to the corresponding feature value of each data point.

Feature Normalization A typical normalization is to divide each feature value by its standard deviation, we can combine this technique with mean centering, in this case the data is said to have been standardized. The idea is that each characteristic is assumed to be drawn from a standard normal distribution. Another type of feature normalization can be used when the data needs to be scaled in a range (0,1). Let min_j and max_j be the minimum and maximum values of the jth attribute, then each feature value x_{ij} for the jth dimension and the ith point is scaled by min-max normalization as follows

$$x_{ij} \leftarrow \frac{x_{ij} - min_j}{max_j - min_j} \tag{2.5}$$

Feature normalisation often provides better performance.

Initialization

The training of neural networks is susceptible to a phenomenon known as the stability problem, where the activation of each layer becomes progressively weaker or stronger. This effect is particularly pronounced in deep neural networks and increases exponentially with the depth of the network. It is essential to address this issue, especially in the context of deep learning.
An effective strategy to mitigate this effect is to stabilize the gradients by initializing

the network from a well-chosen starting point. For example, one approach is to sample the weights from a Gaussian distribution with a mean of zero and a small standard deviation. However, it's important to note that this method is not sensitive to the number of inputs. For example, if we have two neurons - one with only two inputs and the other with 100 inputs - the output of the former will be more influenced by the average weight due to the additive effect. To compensate, a Gaussian distribution can be selected with a standard deviation of $\frac{1}{\sqrt{r}}$, where r is the number of inputs. In addition, the bias weights are initialised to zero.
More sophisticated forms of initialisation techniques have been developed, such as Xavier initialization or Glorot initialization. Regardless of the specific method used, it is important to always include a source of asymmetry. This ensures that the neurons produce distinct features and prevents the redundancy of identical features produced by the neurons.

2.2.6 Gradient-Descent Strategies

Up to this point we have focused on *steepest descent* as a method for learning parameters. This approach uses the gradient of the loss function to guide updates. However, it's important to recognise that steepest-descent is not without its limitations. In particular, it only represents the optimal path when infinitesimal steps are considered. In some cases, it can change to an ascending direction after a small update of the parameters.
This results in the need for numerous course corrections, highlighting a potential inefficiency. Recognising the limitations of steepest descent, we will explore alternative and more sophisticated learning strategies.

Learning Rate Decay

Opting for a constant learning rate may prove to be suboptimal. A lower learning rate may lead to slower convergence, potentially resulting in the algorithm oscillating around a point for an undefined period of time. Conversely, a higher learning rate may cause the algorithm to diverge. Finding the right balance is critical.
One effective approach is to implement a learning rate decay mechanism that allows the learning rate to be adjusted over time. This dynamic adaptation can lead to the desired convergence to an optimal solution. Among the various decay functions available, two of the most common are exponential decay and inverse decay.

Momentum-Based Learning

Momentum-based techniques aim to mitigate the zigzagging behaviour often seen in gradient learning. They achieve this by incorporating an averaged direction based

on recent steps, with the aim of providing a smoother trajectory.
Mathematically, when performing gradient descent with respect to the parameter vector $\overline{W}$, the typical gradient descent updates of the loss L are as follows

$$\overline{V} \leftarrow -\alpha \frac{\partial L}{\partial \overline{W}}; \quad \overline{W} \leftarrow \overline{W} + \overline{V} \tag{2.6}$$

Here, α represents the learning rate.
In momentum-based techniques, the vector $\overline{V}$ is introduced to smooth the updates, incorporating a momentum parameter $\beta \in (0, 1)$ for this purpose. The resulting formula is the following

$$\overline{V} \leftarrow \beta \overline{W} - \alpha \frac{\partial L}{\partial \overline{W}}; \quad \overline{W} \leftarrow \overline{W} + \overline{V} \tag{2.7}$$

The use of momentum-based descent accelerates the learning process, as it tends to move towards a path that is often closer to the optimal solution, while minimizing unnecessary sideways oscillations, resulting in faster learning.

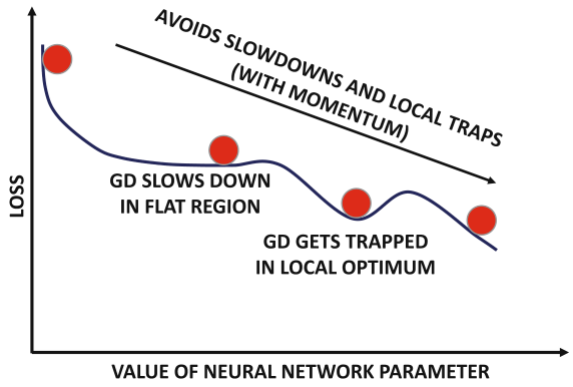


Figure 2.9. Effect of momentum in navigating complex loss surfaces.

Parameter-specific Learning Rates

Consistency in gradient direction can also be achieved by using different learning rates for different parameters. This approach is motivated by the observation that parameters with large partial derivatives tend to oscillate, while those with smaller partial derivatives generally maintain a consistent direction.
One of the early techniques in this regard was the delta-bar-delta method. This method involves monitoring the sign of the partial derivatives. If the sign remains constant, it indicates that the direction is correct and the partial derivatives are increasing in that particular direction. Conversely, if the sign fluctuates, it indicates that the partial derivatives are decreasing.
It's important to note that this method is most effective with standard gradient

descent. When applied to stochastic gradient descent, the potential for error increases significantly. In order to overcome this limitation, several alternative methods have been proposed, which are also effective when used in conjunction with the mini-batch approach, like Adam [13].

Chapter 3

Long Sequence Time Series Forecasting in Deep Learning Approaches

Time Series Forecasting (TSF) can be applied to a wide range of real-world problems where there is a temporal component to be predicted. Applications range from weather, energy consumption, financial indices, retail sales, medical monitoring, traffic forecasting, influenza-like illnesses and many more. The TSF is a challenging task, the designed model must take into account several issues such as trend and seasonal variation, correlation between close observed values and many other time series characteristics that we have seen in Chapter 2. Deep learning has become a great approach for time series analysis tasks, also in TSF has achieved far superior performance compared to the classical machine learning and statistically based model that only care about the linear relationship in the data. In this chapter, we begin a comprehensive exploration of the intricacies surrounding the problem of time series forecasting. In addition, we delve into an in-depth analysis of the most effective deep learning models tailored for this domain.

3.1 Introduction to Time Series Forecasting and Long Sequence Forecasting

3.1.1 Time Series Forecasting Problem

Before the introduction of neural networks, the most commonly used approaches in TSF were the traditional statistical methods (e.g. ARIMA). However, they often failed when applied directly without taking into account the theoretical requirements of stationarity of time series (2.1.5) and the necessary pre-processing. Another pitfall

is that these methods require a large amount of historical data with an explicable structure. In the absence of these requirements, they often fail to effectively extract the latent features and patterns. Furthermore, each method is well suited to a specific time series, so we cannot use it as a universal framework in TSF; the massive computational requirements can become prohibitive.
Conversely, deep learning techniques allow the construction of a global model to predict multiple related time series, but that's not the only advantage. Their superior performance is due to several characteristics: the deep structure allows them to learn well non-linear relationships, they can automatically extract temporal patterns without any theoretical assumptions about the distribution of the data, this greatly reduces the work of pre-processing. The ability to generalize also allows neural networks to exploit cross-series information coming from the different variables in a multivariate time series or coming from several related time series.

3.1.2 Long Sequence Time Series Forecasting Problem

As technology advances and computing power becomes more accessible, there is an unprecedented opportunity to use large amounts of historical data to make long-term predictions. This specialised task, known as *Long Sequence Time Series Forecasting* (LSTF), plays a crucial role in enabling individuals and organizations to make informed decisions for the future. However, traditional forecasting methods often fall short when both the historical window and the forecast horizon are significantly extended.
In response to this challenge, recent years have seen the emergence of various methodologies tailored to the intricacies of LSTF. In particular, the introduction and widespread success of Tansformer[30] in various deep learning domains has brought LSTF into the spotlight. However, the computational cost associated with training and deploying Transformer models is a significant barrier, making them impractical for many real-world LSTF applications.
In response, a multitude of non transformer-based approaches have emerged to provide more practical solutions to LSTF challenges.

3.2 Formal Problem Definition

In this section we recall the definitions of time series, its characteristics and finally we classify TSF tasks.

3.2.1 Time Series

A time series is a sequential arrangement of data points, each corresponding to a specific point in time. It serves as a representation of the evolving state of a particular phenomenon over a given period of time. Formally, a time series matrix, represented as $\hat{X}_{1:N,t-n:t}$, with N dimensions and n samples, it can be described as follows:

$$\hat{X}_{1:N,t-n:t} = \{x_{1,t-n:t}, ..., x_{N,t-n:t}\} \tag{3.1}$$

where $x_{i,t-n:t}, i \in \{1, ..., N\}$ is represented as a vector $x_{i,t-n:t} = \{x_{i,t-n}, x_{i,t-n+1}, ..., x_{i,t}\}$ and $x_{i,t-n+j}, j \in \{0, ..., n\}$ denotes a time series point at time $t - n + j$ of a time series. Thus 3.1 denotes a composition of multiple time series.
Time series data share several common characteristics:

Trend This refers to the long-term changes observed in the time series data. It may take the form of a gradual and persistent shift in values over an extended period, either in an upward or downward direction.

Periodicity It indicates that the time series is subject to regular variations over time. The data follow a recurring pattern of increases and decreases.

Seasonality A time series with seasonality experiences changes in response to natural seasons or cycles. This results in a regular pattern of increases and decreases in the data that correspond to specific periods. It's important to distinguish between seasonality and periodicity; while seasonality follows a known frequency, periodicity involves changes with an unknown frequency.

Randomness In a time series it indicates the presence of irregular, unpredictable fluctuations. Despite these irregular shifts, the entire time series obeys statistical principles and follows certain underlying laws.

In addition, time series can be classified according to other characteristics of TSF tasks.

3.2.2 Time Series Forecasting

There are various analysis tasks associated with time series data, but our primary focus is on time series forecasting. In TSF, we are dealing with one or more features that are ordered in time, and the objective is to predict future values based on the historical data. The temporal order is crucial; the model must exploit the sequential nature of the data to effectively capture upcoming trends.

Let's consider the ith time series represented by $x_{i,t-n:t} = \{x_{i,t-n}, x_{i,t-n+1}, ..., x_{i,t}\}$ with n samples. Using the historical data, we want to predict its future value at time step $t+1$. The result of the prediction will be

$$\hat{X} = f(X_{i,t-n:t}) \quad x_{i,t} \in \mathbb{R} \tag{3.2}$$

Where $f(x)$ is the prediction model.
Depending on the characteristics of the time series and the nature of the problem, we can classify TSF tasks as follows:

Univariate, Covariate and Multivariate Forecasting

Depending on the number of features for the time series prediction and the prediction objective, we can classify the prediction task into.

Univariate *Univariate* forecasting refers to cases where both the historical and future values of the time series are univariate. The dimension of the forecast window is called the forecast horizon. This task can be expressed mathematically as

$$\hat{X}_{t+1} = f(X_{t-n:t}) \tag{3.3}$$

Where:

- t is the current time step
- n is the input step size, meaning that the historical data of n time steps will be used to predict the feature values at time step $t+1$

Covariate In order to predict the future values of a single variable, it may be necessary to model the associations of the trait between multiple variables. These associated variables are known as *covariates*, and the corresponding task is known as a covariate prediction task. While we still have a single feature prediction value, similar to the univariate case, the input now consists of multiple input features. In the case of single level prediction, the mathematical expression is:

$$\hat{X}_{1,t+1} = f(X_{i,t-n:t}) \tag{3.4}$$

Where:

- i denotes the number of input features
- n is the size of the time steps
- t represents the current time step, with $t > 1$

Multivariate In the *multivariate* prediction task, predicting the feature values of multiple variables involves modelling the associations between these variables. Consequently, both the inputs and the outputs are multivariate. This task can be expressed formally as follows:

$$\hat{X}_{j,t+1} = f(X_{i,t-n:t}) \tag{3.5}$$

Where:

- i denotes the number of input features
- j is the number of outputs features, $j \leq i$
- n is the size of the time steps
- t represents the current time step, with $t > 1$

Single-Step Forecasting and Multi-Step Forecasting

Based on the number of future steps to be predicted, we can categorize the forecasting task into two main types: single-step and multi-step forecasting. In the context of multivariate forecasting tasks, we can express both as follows

Single-Step Forecasting

$$\hat{X}_{j,t+1} = f(X_{i,t-n:t}) \tag{3.6}$$

Multi-Step Forecasting

$$\hat{X}_{j,t+1:t+h+1} = f(X_{i,t-n:t}) \tag{3.7}$$

Where:

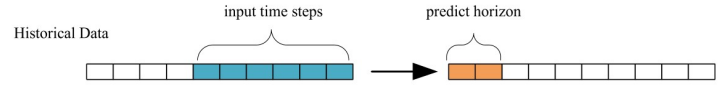
- i is the number of input features
- j is the number of output features, $j \leq i$.
- n is the time step size
- t is the current time step, where $t > 1$.
- h is the predicted horizon

In addition, multi-step forecasting can also be achieved using recursive forecasting. In this method, the predicted value is calculated at time step $t + 1$, combined with historical data, and then fed back into the model to calculate the forecast for time

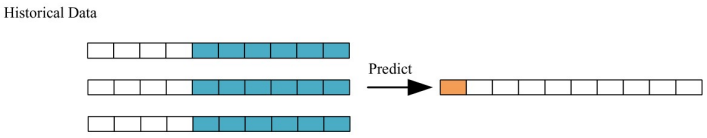
step $t+2$. This iterative process can be extended to forecast time steps further than $t+n$ in a similar way to the multi-step forecasting approach. More mathematical:

$$\begin{aligned} \hat{X}_{i,t+1} &= f(X_{i,t-n:t}) \\ \hat{X}_{i,t+2} &= f(X_{i,t-n+1:t+1}) \\ &\vdots \\ \hat{X}_{i,t+h+1} &= f(X_{i,t-n+h:t+h}) \end{aligned} \tag{3.8}$$

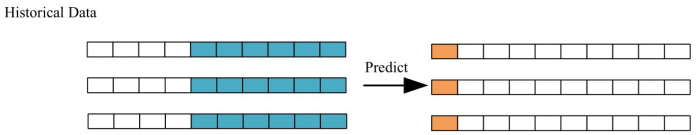
However, this method leads to an accumulation error problem, resulting in low forecast accuracy.



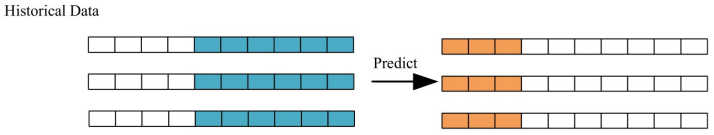
(a) Univariate forecasting



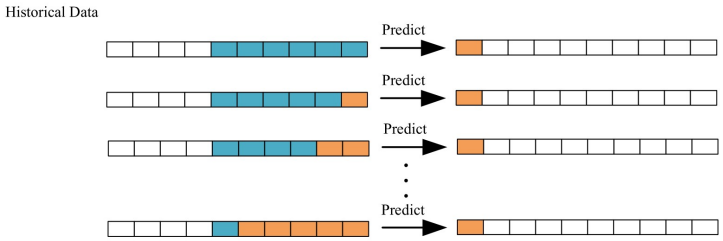
(b) Covariate forecasting



(c) Multivariate forecasting



(d) Multi-Steps forecasting



(e) Recursive forecasting

Figure 3.1. Time Series forecasting tasks

3.2.3 Long Sequence Time-Series Forecasting

The typical definition of *Long Sequence Time Series Forecasting* (LSTF) as forecasting into the distant future can be quite ambiguous. To provide clarity, we need to examine this concept from both a *relative* and an *absolute* perspective:

Relative perspective In this view, the task of forecasting over a long time horizon is typically categorised as LSTF. However, what constitutes long or short is relative and based on comparative results. Typically, the forecast accuracy of the model tends to decrease as the forecast horizon increases.

Absolute Forecast The absolute view is more application specific. Here, the definition of the LSTF is derived from the characteristics of the particular domain. In most cases, it is closely related to the cyclical nature of the data, which is a critical factor in forecasting tasks. In essence, LSTF can be considered when the forecast horizon is close to or exceeds the maximum cycle of the data relevant to the forecasting task. For example, in meteorology, forecasting weather conditions over a month or longer is categorised as LSTF, as the maximum cycle of data in these tasks is typically one month or one quarter. Conversely, in electrical load forecasting, an LSTF scenario might involve predicting load patterns several years ahead, up to a decade or two, where the maximum cycle is approximately one year.
Based on this examination, the definition of an LSTF problem depends on the comparison between the forecasting horizon and the cycles inherent in the data.

3.3 Literature Review

The application of deep learning models to time series forecasting has marked a significant paradigm shift in predictive analytics. As the demand for accurate and reliable forecasting over extended time horizons continues to grow, the study of Long Sequence Time Series Forecasting has attracted considerable attention from the research community. In this section, we provide a comprehensive review of the existing literature on deep learning models specifically tailored to the LSTF problem. We delve into a wide range of studies, starting from RNN-based models, CNN-based models, Transformers-based models, Representation Learning frameworks and Composite models.

3.3.1 Recurrent Neural Network Based Model

Recurrent Neural Networks (RNN) are a type of ANN commonly used to detect patterns in a sequence of data. The difference to other models (such as Multi-Layer

Perceptron) is how the information flows and how it is passed through the network, the RNN has cycles and sends information back to itself. This allows it to take into account past inputs rather than just the current one, we can see the basic flow in the figure. Its structure is well suited to processing temporal information, hence time series. RNN has been widely used for processing sequential data, but the vanilla model suffers from the vanishing and exploding gradient problem when dealing with long sequences.
Long Short Term Memory (LSTM) has been developed to counteract this problem using gating units and memory mechanisms.
LSMT requires a lot of computational resources to be trained, so researchers proposed an improved method called Gated Recurrent Unit (GRU) model, which simplifies a lot the previous model thanks to an update gate.

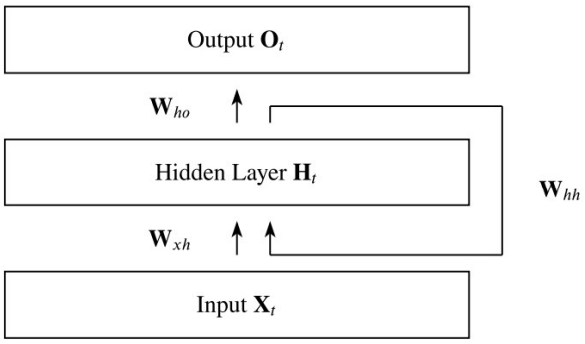


Figure 3.2. Vanilla RNN schema, the hidden layer H_t can be a composition of multiple hidden layers

RNN-Variant

Given the outstanding ability of Long Short-Term Memory (LSTM) models to capture long-term dependencies, several Recurrent Neural Networks (RNN) incorporating LSTM architectures have been introduced. In particular, a novel model [12] combining RNN and LSTM has emerged and demonstrated its ability to predict the potential yield of photovoltaic (PV) solar power generation. This innovative approach adeptly assimilates temporal variations from various sources, including solar radiation and prevailing weather conditions.
In addition, a novel multivariate time series forecasting methodology based on Bidirectional LSTM (BiLSTM) [18] was presented. This model is uniquely trained on a number of features, enhancing its predictive capabilities. It is imperative to highlight the remarkable success of ES-LSTM [28], the outstanding champion of the M4 [20] competitions. By combining Exponential Smoothing (ES) with LSTM, this

model achieves a harmonious blend of efficient learning to capture the fundamental components of each sequence. The intrinsic strength of LSTM lies in its ability to recognise non-linear trends and promote cross-learning. ES-LSTM is a prime example of this, operating as a hierarchical framework that integrates local and global parameters to facilitate cross-learning.
A novel solution has been proposed to address the challenge of training RNNs on long sequences. This method captures temporal dependencies across different scales and layers. The Dilated RNN [5] variant, consisting of multiple layers with hierarchical and recursive expansion, effectively mitigates the complexities encountered by RNN in the domain of Long Sequence Time Series Forecasting (LSTF).

RNN with Attention

In 2014, the introduction of the attention mechanism marked a significant advance in the field of deep learning. In recent years, this mechanism has been increasingly integrated with Recurrent Neural Networks (RNNs) to enhance the model's ability to recognise relevance for future predictions. This combined approach has led to significant performance improvements in Long Sequence Time Series Forecasting (LSTF).
A novel architecture [35] has emerged that relies on a combination of RNN, Long Short-Term Memory (LSTM) and the attention mechanism. First, the LSTM Spatial Attention (LSTM-SA) component is used to extract spatial correlations. The attention mechanism is then used to further refine the relevance. Finally, the LSTM-SA component is reapplied to extract event correlations from both dependent and independent variables.
Effective prediction of multivariate time series in LSTF scenarios is challenging because not all variables may contribute meaningfully to the prediction process. Here, the attention mechanism plays a critical role in identifying and filtering out extraneous independent variables, thereby improving prediction accuracy. Another interesting method that incorporates the attention mechanism is the DA-RNN [25]. This two-stage RNN framework first extracts correlations and hidden features from the data. The temporal attention mechanism is then used to selectively focus on specific hidden states, which improves the model's predictive capabilities.

RNN with Seq2Seq and Attention

The introduction of the Sequence-to-Sequence (Seq2Seq) paradigm within Recurrent Neural Networks (RNNs) has significantly improved their effectiveness in handling long sequences, particularly in multi-step prediction tasks. A significant illustration of this progress can be found in a paper [9] that blended an encoder-

decoder architecture with LSTM units. In this model, a bidirectional LSTM decoder is employed that effectively propagates future information both forward and backward through the sequence. In particular, the model considers different historical periods at each time step to generate different attention vectors. This approach to learning temporal attention over multiple historical periods has been shown to significantly improve prediction accuracy.
Another innovative model in this area is the Multi-Horizon Quantile Recurrent Forecaster (MQRNN) [32]. It adopts an encoder-decoder structure, where the decoder part of the RNN is transformed into multiple fully connected layers. This architectural adjustment addresses multi-horizon forecasting tasks, enabling more accurate forecasting over different time frames.
Furthermore, Seq2Seq and attention mechanisms can be seamlessly integrated, as exemplified in the MTSMFF model [8]. In this configuration, a Bidirectional LSTM with temporal attention acts as an encoder, adeptly learning long-term correlations and capturing multivariate temporal data. The final decoder module is then composed of stacked vanilla LSTMs.

3.3.2 CNN-based model

While Convolutional Neural Networks (CNNs) excel at processing long sequences in computer vision tasks, they primarily capture short-term local dependencies. This characteristic makes them less effective for Long Sequence Time Forecasting (LSTF), where capturing long-term global dependencies is critical. The introduction of dilated causal convolution [36], as shown in the Figure 3.3, has addressed this limitation. Several variants have emerged from this innovation.

CNN Variant

The Temporal Convolutional Network (TCN) [2] is suited to modelling and analysing sequential data. Thanks to its causal and dilated convolution, it effectively prevents information leakage from the future and handles long sequence inputs. TCN demonstrates that a simple convolutional network can perform comparably to RNN-based models, with the added benefits of parallelism and feature compression, making it even more suitable than vanilla RNNs for certain applications.

CNN with Attention

Traditional combinations of LSTM and other models with attention mechanisms have shown limitations in multivariate long-term prediction tasks. In response, a novel network called DSANet [11] has been introduced to address this challenge. This model uses two parallel convolutions to capture global and local temporal

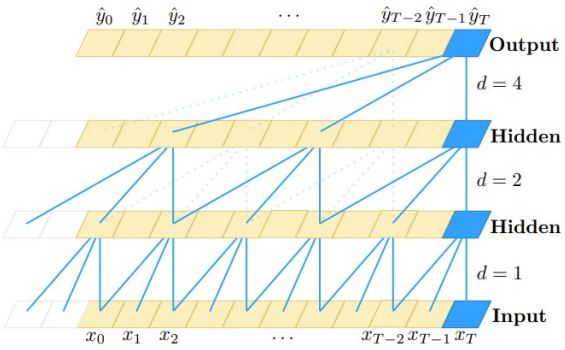


Figure 3.3. The dilated causal convolution in a TCN

patterns, while still employing a self-attention mechanism to learn connections between multiple time series.

CNN with Seq2Seq

Efforts have also been made to exploit the strengths of both CNNs and sequence-to-sequence (Seq2Seq) models to improve prediction performance. SCINet [16] is a notable example, as it combines rich features at different granularities, thus improving the capture of both local and long-term temporal dependencies.

3.3.3 Transformer-based model

Transformers have attracted considerable attention for their exceptional performance in various deep learning tasks, including natural language processing (NLP), computer vision (CV), and speech processing. Their ability to model long-range dependencies and interactions within sequential data has led to the development of numerous Transformer variants, which have been successfully applied to various time series tasks, including LSTF. In this section, we will explore some of the key insights, benefits and potential limitations of these models.

Vanilla Transformer

The Vanilla Transformer[30] is structured with both an encoder and a decoder, each consisting of several identical blocks. The encoder block contains a multi-head self-attention module coupled to a position-wise feed-forward network. The decoder block, on the other hand, contains cross-attention models that sit between the multi-head self-attention module and the position-wise feed-forward network.
Unlike RNNs, the Vanilla Transformer does not rely on recurrence. Instead, it uses positional coding embedded in input embeddings to effectively model sequence information.

Time Series Forecasting Transformer

To adapt the Vanilla Transformer architecture for time series tasks, some tuning is required:

Timestamp encoding The inclusion of timestamp information is essential for a thorough understanding of temporal data. This includes the introduction of calendar timestamps (such as seconds, minutes, hours) and special timestamps (such as holidays and events), which Vanilla Transformer lacks. Both Informer [38] and Autoformer [34] address this by introducing a learnable embedding layer for encoder timestamps, which serves as additional positional encoding.

Attention module This is essentially a fully connected layer that dynamically computes weights based on the pairwise similarity of input patterns. However, this approach imposes a limit on the maximum path length due to the memory complexity of the Self-Attention module, which is bounded at $\mathcal{O}(N^2)$ (where N is the length of the input time series).
Many efficient transformers have been proposed to cope with this problem, we can reduce the modification in two concepts: introducing a sparsity bias in the attention mechanism, as in Pyraformer [17] and LogTrans [15], and exploring the low-rank property of the self-attention mechanism, as in Informer and FEDformer [39]. In both cases, the aim is to remove noise in order to achieve low order computational complexity.
Specifically: LogTrans implements a convolutional self-attention mechanism that uses causal convolution to generate queries and keys. It also uses a logsparse mask in the self-attention process, effectively reducing the computational complexity to $N \log(N)$.
Instead of explicit sparse bias, Informer chooses to select dominant queries based on query and key similarity. This approach achieves a similar complexity to LogTrans. In particular, the decoder operates in a generative style, allowing direct long-term prediction and minimising cumulative errors.
Pyraformer develops a hierarchical pyramidal attention system that uses a binary tree structure to capture temporal dependencies between different segments. This innovative design leads to linear complexity, provided that an appropriate kernel is used.
FEDformer employs attention operations in the frequency domain, using Fourier and Wavelet transforms. It achieves linear complexity by randomly selecting a fixed subset of frequencies.

Normalize Time Series Data In the field of time series forecasting, the normalization of time series data emerges as a critical factor, yet surprisingly only a single work has addressed this subject primarily. The Non-stationary Transformer [19] takes a distinctive approach by focusing on the issue of *over-stationarization* in time series forecasting. It introduces a novel Series Stationarization Module and De-stationary Attention designed to address this specific challenge.

Token Input Bias Autoformer, a segmentation-based representation mechanism, uses a simple seasonal-trend decomposition block with an auto-correlation mechanism. This last block calculates the time delay similarity between the input signal and aggregates the top k similar sub-series to reduce complexity.
PatchTST [22], which is based on two main components: the *segmentation* of time series into sub-series level patches that are provided as input tokens to Transformer; and *channel independence*, where each channel contains a single univariate time series that has the same embedding and weights across all series. The result is a significant reduction in computational and memory usage.

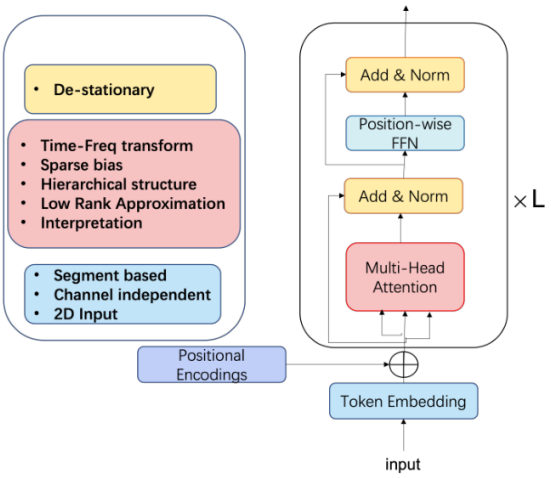


Figure 3.4. Transformer variants for time series forecasting schema

3.3.4 Linear Model

Tranformers have achieved great success in several long sequence tasks, but a recent paper has highlighted their limitations for time series tasks. In particular, the use of positional encoding and tokens to embed sub-series facilitates the preservation of some ordering information, but the permutation-invariant nature of the self-attention mechanism prevails, and this inevitably leads to temporal information loss. In [37] the LTSF linear models have been proposed to prove the latter claim; these models outperform existing transformer-based LTSF models. The insight of this

work is that the iterative multi-step forecasting strategy of the Transformer-based model leads to a significant accumulation error. Conversely, the direct multi-step forecasting strategies achieve better performance in this task. In LTSF-Linear baseline, the historical time series are directly regressed for future prediction via weighted sum operation. LTSF-Linear consists of a set of linear models.

Vanilla Linear This is a one-layer linear model and is the simplest architecture. In order to improve the performance in different domains, two different preprocessed variants have been developed.

D-Linear In its pre-processing, the row data in the input is decomposed into a trend and seasonal component, as well as Autoformer and FEDformer. Two linear layers are then applied to each component, and the two features are summed to produce the final prediction. This model is very effective when there is a clear trend in the data.

N-Linear This model was proposed to improve performance when there is a distribution shift in the data set by introducing a simple normalization. NLinear first subtracts the input by the last value of the sequence, then feeds the input to a linear layer, and finally adds back the subtracted input before making the final prediction.

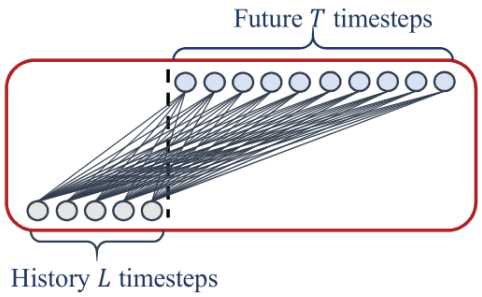


Figure 3.5. LTSF Vanilla Linear model schema

The main scope of this work is to demonstrate how the Transformer-based model's performance is exaggerated in LTSF, providing some insight for future work.

3.3.5 Unsupervised Representation Learning Method

All of the methods and architectures listed above are supervised methods, but there are many *unsupervised* approaches that deserve mention. The main goal of these unsupervised learning methods is to obtain a learned latent representation

that can capture potentially valuable information within time series and multiple underlying mechanisms. Of course, such approaches can be useful for all time series tasks, but we will focus on time series forecasting. The introduction of deep learning has led to the introduction of many *auto-encoder* and *seq2seq* models for representation learning.
However, the reconstruction of complex time series data remains challenging, so *self-supervised* learning has recently been introduced. This new paradigm uses pretext tasks to autonomously generate labels using instrinsic information derived from unlabeled data.
Recently, *contrastive-learning* techniques have emerged by improving performance in several representation learning tasks, including time series; this paradigm is based on mapping two samples as positive pairs to nearby features. The conventional end-to-end models focus only on optimizing a specific downstream task, ignoring the learning of meaningful representations. To test the effectiveness of a learned representation in a downstream task, it is sufficient to add a simple classifier or predictor to it. In particular, we focus on self-supervised learning techniques.

Self-supervised Learning Methods

Supervised learning methods learn semantic information directly from a large amount of labeled data, but this requires a large amount of annotated training data, often tailored to a specific task. *Self-supervised learning* emerged to address this problem. It doesn't require manually labeled samples, instead it generates a self-supervised signal from the raw data. We can classify these self-supervised techinques in 3 categories: *adversarial*, *predictive* and *contrastive* methods, reflecting the different kind of pretext task used to generate the self-supervised signal.

Adversarial methods In these methods, the pretext task is to distinguish real from fake data in order to learn robust time series representations. An example is the Generative Adversarials Network [10] (GAN), which can generate previously unseen data to augment smaller time series datasets. Conventional GAN-based methods use back-propagation signals through a competitive process involving a discriminator and a generator. This configuration typically creates a two-player min-max game, where the generator tries to improve its ability to trick the discriminator, which conversely tries to become the most accurate in distinguishing between real and fake data. This adversarial training allows the generator to capture the important property of the underlying data distribution, and differs from regression-based methods in that it considers not only the regression error, but also the prediction error. Several variants of GAN have been developed to capture temporal dependencies.

Predictive methods This type of methods learns meaningful representations that capture the shared information between different parts of the time series data by maximizing the mutual information derived from related slices of the original time series or different views generated by data augmentation. The *predictive methods* eliminate the need to reconstruct the full representation by predicting future, missing or contextual information of partial time series.

Contrastive Methods *Contrastive methods* learn a significant representation from time series by optimising self-discrimination tasks. The pretext task relies on the underlying similarity between samples, eliminating the need to reconstruct the complete input. Contrastive methods often generate an extended view of the raw data using multiple transformations, and then learn representations by constructing positive samples against negative ones. In computer vision, contrastive methods are classified into instance-level and prototype-level methods. For time series data, there is a specific method tailored for them, the *temporal level contrast*, we will focus on the latter.

Temporal-Level Contrastive Learning Methods The application of classical computer vision contrastive learning techniques to time series data often fails to capture the intricate characteristics of such data. To address this limitation, a new paradigm, *temporal-level contrastive learning* models, has been developed. This type of methods mainly focuses on capturing scale-invariant representations at each individual time stamp, instead of instance-level contrastive learning focusing on capturing the overall characteristic of the entire time series. By considering both strategies, the research aims to improve the ability of contrastive learning methods to capture the complex information inherent in time series data. One of the most promising recent works is the CoST [33] framework, which exploits inductive biases within the model architecture to learn disentangled seasonal trend representations. It also introduces a novel frequency domain constriction loss to learn a more discriminative seasonal representation. CoST highlights the benefits of learning disentangled seasonal trend representations through constrastive learning for time series forecasting.

3.4 Expected Novelties

Recent studies have shown that the capabilities of transformer-based models, once widely considered to be the state of the art in forecasting accuracy, may have been somewhat overestimated. Surprisingly, simpler linear-based models have emerged as contenders, often significantly outperforming their transformer counterparts. It is

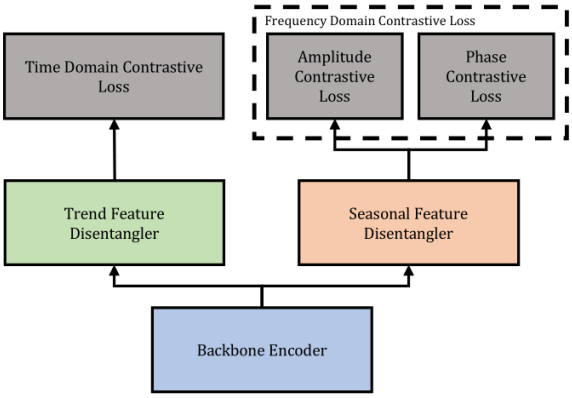


Figure 3.6. Overall CoST framework

important to note, however, that the conventional normalization techniques used in these models don't take into account all the tricky statistical properties inherent in time-series data, in particular the pervasive problem of *non-stationarity*.

Our research aims to address this problem. Through the development of novel strategies to account for non-stationarity, our goal is to provide a more robust framework for the data distributions we use. This in turn has the potential to significantly improve the predictability of Long Sequence Time Series Forecasting.

Furthermore, another aim of our work is to move beyond the realm of conventional transformer applications in LSTF to evaluate the real effectiveness of these models. We explore the novel territory of using transformer-based models as encoders within a representation learning architecture, using CoST [33] as a reference point. This approach represents a significant deviation from established practices and promises to open up new frontiers of insight and innovation in time series forecasting.

Through these diverse contributions, our research not only focuses on the design of a new architecture for dealing with non-stationary time series, but also offers a fresh perspective on the application and adaptation of transformer-based models in the domain of time series analysis and forecasting.

Chapter 4

Proposed Method: SLinear

Data normalization is a fundamental preprocessing step in the analysis of time series, but finding an appropriate method to deal with time series normalization is not an easy task. This is because most traditional normalization methods make assumptions that do not hold for most time series.
This difficulty in dealing with time series tasks is also reflected in the performance of neural network approaches, which declines significantly with improperly normalized data.

4.1 Non-Stationarity of Time Series in Deep Learning

The main challenge in predicting future time series in real-world datasets is their inherent *non-stationarity*. This non-stationarity manifests itself as a continuous shift in statistical properties and distributions over time, rendering time series less predictable. Previous attempts have tried to mitigate this challenge through preprocessing techniques that aim to produce a more stable data distribution.
However, it's important to note that non-stationarity is an intrinsic property of real-world datasets. The use of a static normalization technique, such as z-score, involves the use of fixed statistics for both training and inference. Other approaches have explored more complex methods or the use of manually crafted stationary features. These methods rely on heuristic feature extraction schemes, with no guarantee that the chosen scheme is optimal for the task at hand.
In response to this problem, innovative solutions have emerged. For example, Deep Adaptive Input Normalization [24] (DAIN) introduces a layer capable of learning how to appropriately normalize the data, and dynamically adjusting the applied normalization during inference to fit the current time series. Another proposed model is Adaptive Normalization [23], which addresses non-stationary time series through a three-step process: transforming non-stationary sequences into stationary

ones and generating a series of disjoint sliding windows; outlier removal; and finally, data normalization.
While these direct stationarity designs can mitigate non-stationarity in time series to improve predictability, there is a possibility that they may overlook hidden properties of real-world datasets.
In an effort to address this concern, recent work has introduced Non-stationary Transformers [19]. This model incorporate both series stationarization and de-stationarity attention mechanisms to reintegrate non-stationary information from the raw series. This allows the model to improve its predictability while maintaining its adaptability and capability.
Previous work on non-stationary time series has mainly revolved around the use of complicated deep neural networks and transformer-based models. However, [37] landmark work highlights the remarkable effectiveness of simple linear layer-based models, outperforming even more complex counterparts. Attracted by this discovery, we delved into the study of stationary methods tailored to this simple class of models. Our main focus was on the series stationarization module of [19].
This work led to the development of a novel model we've called *SLinear*, which is the subject of this section.

4.2 Architecture

The overall architecture of our model is based on three components: the Normalization module, which first deals with non-stationary series; the Linear Regressor, which regresses the normalized historical time series. Finally, the predicted time series are provided to a De-normalization module that transforms the output back to the original statistics. The overall architectural scheme of our model is shown in Figure 4.1

4.2.1 Normalization Module

This components ($\mathcal{N}$) adopts a sliding window technique over the time dimension to normalize the raw input time series $x \in \mathbb{R}^{B\times L\times C}$. Through translations and scaling operations, using mean μ_x and variance σ_x respectively, we obtain the transformed time series $x' \in \mathbb{R}^{B\times L\times C}$. Where B is the batch size, L is the input length and C is the number of channels (time series variables).
The normalization process is characterised by the following equations:

$$\mu_x = \frac{1}{L}\sum_{i=1}^{S} x_{:,i,:}, \quad \sigma_x = \frac{1}{L}\sum_{i=1}^{S}(x_{:,i,:} - \mu_x)^2, \quad x'_{:,i,:} = \frac{1}{\sigma_x} \odot (x_{:,i,:} - \mu_x) \tag{4.1}$$

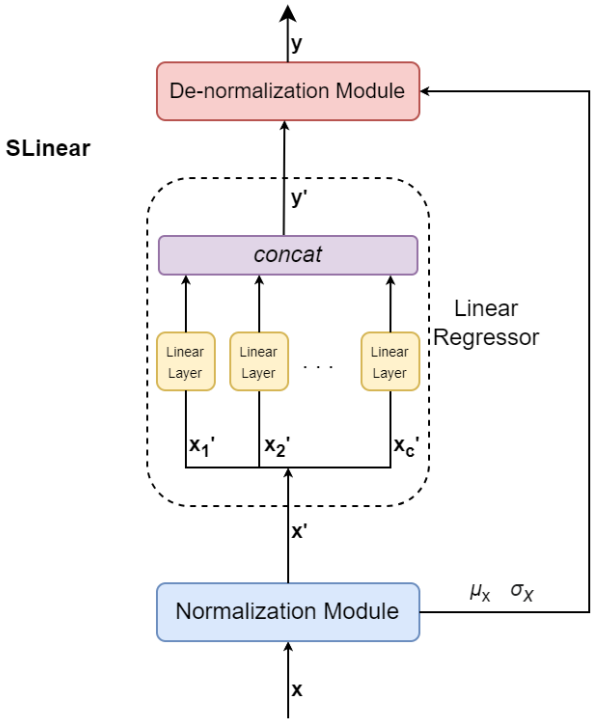


Figure 4.1. Overall *SLinear* schema

where $\mu_x, \sigma_x \in \mathbb{R}^{B\times 1\times C}$, $\frac{1}{\sigma_x}$ and $\odot$ are respectively the element-wise division and product.
The output $x' = [x'_{:,1,:}, x'_{:,2,:}, ..., x'_{:,L,:}]^T$ is passed directly to the $\mathcal{L}$ module, while μ_x and σ_x are passed to the $\mathcal{D}$ module. This normalization process results in a more stable distribution of the input data, effectively reducing discrepancies between the input time series.

4.2.2 Linear Regressor

This module ($\mathcal{L}$) uses a simple temporal linear layer to directly regress historical time series data that has been normalized by the normalization module. It performs this operation in a channel-independent manner, ultimately concatenating the outputs along the channel dimension.
The mathematical representation of this process is

$$\mathcal{L}(x') = y' \quad s.\ t.$$

$$x' = [x'_{:,:,1}, x'_{:,:,2}, ..., x'_{:,:,C}] \quad y'_{:,:,j} = x'_{:,:,j} \cdot W_j \quad \forall j \in \{1, 2, ..., C\}, \tag{4.2}$$

$$y'_{:,i,:} = concat(y'_{:,i,1}, y'_{:,i,2}, ..., y'_{:,i,C})$$

Where $y'_{:,:,j} \in \mathbb{R}^{B \times F \times 1}$, $y' = [y'_{:,1,:}, y'_{:,2,:}, ..., y'_{:,L,:}]^T \in \mathbb{R}^{B \times F \times C}$ and $W_j \in \mathbb{R}^{L \times F}$; F is the forecasting horizon. The output y' is passed directly to the $\mathcal{D}$ module.

4.2.3 De-normalization Module

Given the predictions from the $\mathcal{L}$ module, denoted as $y' = [y'_{:,1,:}, y'_{:,2,:}, ..., y'_{:,L,:}]^T$, a de-normalization operation is applied. This process restores the original statistics (μ_x and σ_x) to y', yielding the final forecast results $\hat{y} = [\hat{y}_{:,1,:}, \hat{y}_{:,2,:}, ..., \hat{y}_{:,L,:}]^T$. The de-normalization module ($\mathcal{D}$) can be formalised as follows

$$\hat{y}_{:,i,:} = \mathcal{D}(y'_{:,i,:}) = (\sigma_x \odot y'_{:,i,:}) + \mu_x \tag{4.3}$$

This architecture ensures that the main module $\mathcal{L}$ receives stationarized inputs that follow a stable distribution. This feature enhances its ability to generalize, especially in real-world data sets.

4.3 Datasets and Preprocessing

To evaluate the performance of our proposed method, SLinear, compared to the LTSF-Linear Baseline, we employ five primary datasets: ETTh1, ETTh2, ETTm1, ETTm2 and Electricity.

4.3.1 ETDataset

The first four datasets, namely ETTh1, ETTh2, ETTm1 and ETTm2, together form the Electricity Transformer Dataset (ETDataset [1] [38]). This dataset contains two years of oil temperature measurements of electricity transformers from two different regions within one province in China. Each data point within this dataset consists of eight features, including the predicted value (oil temperature) and six different categories of external power load features. The data is sampled at minute intervals, resulting in a total of 70,080 data points. This rich dataset has facilitated

[1] `https://github.com/zhouhaoyi/ETDataset`

the creation of two 15-minute level datasets, ETTm1 and ETTm2, as well as two hourly level datasets, ETTh1 and ETTh2.

4.3.2 Electricity

The Electricity dataset[2] contains 15-minute electricity consumption values from a pool of 370 customers over the period 2011 to 2014. Each consumption value is expressed in kilowatts (kW).

4.3.3 Preprocessing

Ensuring the integrity and quality of our data set is critical to the success of our forecasting model. We begin by partitioning the data into separate sets for training, validation and testing, following a common 60/20/20 split. This partitioning not only allows us to train our model on a substantial portion of the data, but also provides a dedicated validation set that allows us to apply some regularization techniques (such as *early stopping*) in order to avoid overfitting; a test set is also fundamental for an unbiased evaluation.
We then move on to data preprocessing, a crucial step in preparing our dataset for effective training. Using the Standard Scaler from the Sklearn Python library[3], we standardize the raw data. This process normalizes the scale of the features, mitigating potential problems caused by data points that look less like standard normally distributed data.
In addition to standardization, we augment our dataset with a set of extra time-related features, often referred to as *covariates*. These covariates encapsulate various temporal attributes such as hour-of-day, day-of-week, day-of-month, and day-of-year. By integrating these covariates into our dataset, we provide the model with a richer contextual understanding of temporal patterns. This extension enables the model to recognise intricate variations that can occur at different points in time, allowing it to make more accurate predictions.

4.4 Training and Evaluation phases

We follow the well-known training procedure commonly used in end-to-end LSTF tasks. This involves providing the model with a time series $x \in \mathbb{R}^{L \times C}$, characterised by a specific lookback window L and a defined number of variables C. The model then generates a time series $\hat{y} \in \mathbb{R}^{F \times C}$ with a certain prediction length F. Subsequently,

[2] `https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014`

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

the Mean Squared Error (MSE) is calculated to measure the prediction training performance.
Given our objective to comprehensively evaluate the predictive capacity of our model over different prediction horizons and to facilitate a fair comparison with other evaluated models, it is crucial to train different models for each prediction horizon considered.
For the evaluation phase, we use a dedicated test dataset that is compared to the corresponding $y \in \mathbb{R}^{F \times C}$ labels using two key metrics: MSE and Mean Absolute Error (MAE). A full discussion of these metrics is given in Chapter 5.

4.5 Implementation Details

Our implementation is based on PyTorch Lightning, a powerful deep learning framework that extends the capabilities of PyTorch while maintaining high performance. This choice provides us with a harmonious blend of flexibility and computational efficiency. Conversely, we use the official implementation provided by the authors to train the benchmark models. All of our training and experimentation takes place on Google Colab T4 GPUs.
In line with previous studies, we rigorously evaluate our models focusing on multivariate prediction.
Our training scheme spans a maximum of 10 epochs, guided by an early stopping callback on the validation set, with a patience parameter set to 3. The batch size and learning rate, tailored to each dataset, are meticulously outlined in Table 4.1, along with the prediction length (Table 4.1). Meanwhile, the lookback window, often referred to as the sequence length, remains fixed at 336. Anchoring our optimization strategy is the Adam optimizer, initialised with a learning rate of 1E-4.

	Prediction Length	Learning Rate	Batch Size
ETTh1	24, 48, 168, 336, 720	5E-3	32
ETTh2	24, 48, 168, 336, 720	5E-2	32
ETTm1	24, 48, 96, 288, 672	1E-4	8
ETTm2	24, 48, 96, 288, 672	1E-3	8
Electricty	24, 48, 168, 336, 720	1E-3	16

Table 4.1. Hyperparameters considered for each dataset

Chapter 5

Experimental Results

This chapter provides a rigorous analysis of model performance, supported by a robust evaluation framework. In this section, we discuss of the evaluation metrics and explain their importance. We also provide a detailed description of the evaluation setup, describing the scenarios used for comprehensive testing.

5.1 Evaluation Metrics

Following previous research in the field of Long Sequence Time Series Forecasting (LSTF), we use established evaluation metrics, namely Mean Squared Error (MSE) and Mean Absolute Error (MAE), to assess the predictive ability of our model. These metrics are computed on a test set of N data points. Specifically, given the predicted values $\hat{y} = [\hat{y}_{1,:,:}, \hat{y}_{2,:,:}, ..., \hat{y}_{N,:,:}] \in \mathbb{R}^{N \times F \times C}$ and the corresponding ground truth $y = [y_{1,:,:}, y_{2,:,:}, ..., y_{N,:,:}] \in \mathbb{R}^{N \times F \times C}$, the MSE and MAE are defined as follows:

$$\text{MSE}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2, \quad \text{MAE}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y} - y| \tag{5.1}$$

We can notice how in MSE, due to the nature of the power function, higher errors penalise the metric more than lower ones.

5.2 Experimental Setup

While achieving absolute reproducibility across different machines and frame versions can be challenging, Pytorch Lightning offers configurable options to exert control over sources of randomness. By explicitly setting a *seed*, we ensure that specific random numbers, such as those initialising the weights of a Neural Network Layer, remain consistent across multiple runs within the same environment. This

approach increases the reliability of performance evaluations by minimizing potential bias.

5.3 Forecasting Results

5.3.1 Quantitive Results

In line with the specified experimental setup, we systematically evaluate the performance of our linear-based models across five different multivariate forecasting benchmarks, using the linear model as a baseline. Notably, our model shows a remarkable superiority over the linear model in almost every benchmark, demonstrating an impressive MSE reduction (averaged over prediction lengths) of 22% on ETTh2 and 23% on ETTm2. Furthermore, compared to the NLinear approach, our model maintains a consistent average MSE reduction across the entire benchmark set, showing a notable improvement of 1.67% under optimal conditions.
A comprehensive overview of the quantitative results, covering different forecast horizons and benchmarks, can be found in Table 5.1. In addition, Table 5.2 provides a detailed breakdown of the performance gains achieved by SLinear relative to other linear-based models, providing a comprehensive view of our model's performance.

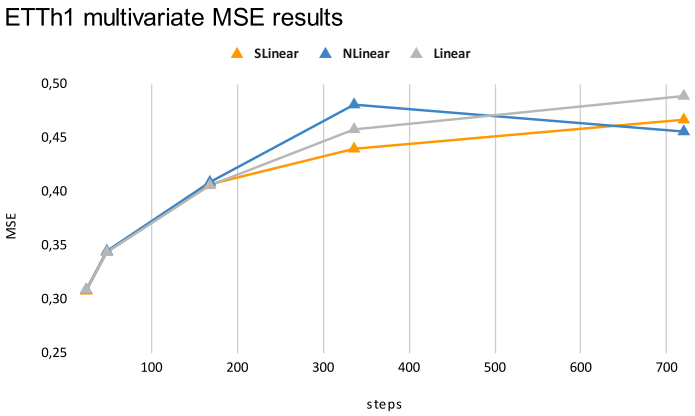
In Figure 5.1, we present the main results obtained on the ETDataset. It is evident that our model (SLinear) outperforms the Linear model for almost all prediction lengths, and also shows superior performance compared to NLinear.

Methods		SLinear		NLinear		Linear	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	**0.308**	**0.355**	0.309	0.357	0.309	0.356
	48	**0.344**	**0.375**	0.345	0.377	**0.344**	0.376
	168	0.407	**0.416**	0.409	0.417	**0.406**	**0.416**
	336	**0.44**	**0.437**	0.481	0.464	0.458	0.451
	720	0.467	0.47	**0.456**	**0.461**	0.489	0.487
ETTh2	24	**0.176**	**0.274**	0.184	0.28	0.194	0.286
	48	**0.231**	**0.313**	0.257	0.329	0.252	0.33
	168	**0.344**	0.395	0.345	**0.389**	0.457	0.454
	336	**0.392**	**0.434**	0.398	**0.434**	0.677	0.556
	720	0.459	0.474	**0.432**	**0.461**	0.929	0.655
ETTm1	24	0.209	0.286	**0.205**	**0.284**	**0.205**	**0.284**
	48	0.264	0.323	**0.263**	**0.322**	**0.263**	**0.322**
	96	**0.294**	**0.34**	0.3	0.348	0.3	0.348
	288	**0.362**	**0.379**	0.364	0.382	0.364	0.382
	672	**0.427**	**0.415**	0.431	0.421	0.431	0.421
ETTm2	24	**0.095**	**0.189**	**0.095**	0.191	0.102	0.203
	48	**0.125**	**0.22**	0.126	0.223	0.143	0.244
	96	**0.163**	**0.25**	0.164	0.252	0.18	0.276
	288	**0.256**	**0.314**	0.26	0.318	0.422	0.421
	672	**0.359**	**0.379**	0.36	0.38	0.707	0.546
Electricity	24	**0.104**	**0.202**	0.105	0.203	**0.104**	0.203
	48	**0.119**	**0.216**	0.12	0.217	**0.119**	0.217
	168	**0.148**	**0.241**	0.149	0.243	**0.148**	0.244
	336	0.169	**0.261**	0.169	0.263	**0.166**	0.265
	720	0.207	**0.293**	0.208	0.296	**0.203**	0.299

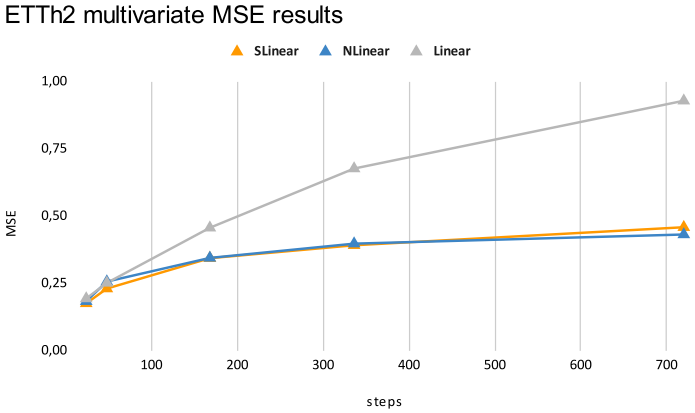
Table 5.1. LTSF forecast results in multivariate settings on several prediction lenghts, we report MSE and MAE metrics on 5 benchmarks, we highlight the best results in **bold** and the second best in underline.

Methods		NLinear		Linear	
Metrics (%)		MSE	MAE	MSE	MAE
ETTh1	24	0.32	0.56	0.32	0.28
	48	0.29	0.53	0	0.27
	168	0.49	0.24	-0.25	0
	336	8.52	5.82	3.93	3.1
	720	-2.41	-1.95	4.5	3.49
ETTh2	24	4.35	2.14	9.28	4.2
	48	10.12	4.86	8.33	5.15
	168	0.29	-1.54	24.73	13
	336	1.51	0	42.1	21.94
	720	-6.25	-2.82	50.59	27.63
ETTm1	24	-1.95	-0.7	-1.95	-0.7
	48	-0.38	-0.31	-0.38	-0.31
	96	2	2.3	2	2.3
	288	0.55	0.79	0.55	0.79
	672	0.93	1.43	0.93	1.43
ETTm2	24	0	1.05	6.86	6.9
	48	0.79	1.35	12.59	9.84
	96	0.61	0.79	9.44	9.42
	288	1.54	1.26	39.34	25.42
	672	0.28	0.26	49.22	30.59
Electricity	24	0.95	0.49	0	0.49
	48	0.83	0.46	0	0.46
	168	0.67	0.82	0	1.23
	336	0	0.76	-1.81	1.51
	720	0.48	1.01	-1.97	2.01
Average Reduction	ETTh1	1.44	1.04	1.70	1.43
	ETTh2	1.67	0.44	22.50	11.99
	ETTm1	0.23	0.70	0.23	0.70
	ETTm2	0.64	0.94	23.49	16.43
	Electricity	0.59	0.71	-0.76	1.14
	Total	0.98	0.78	10.33	6.81

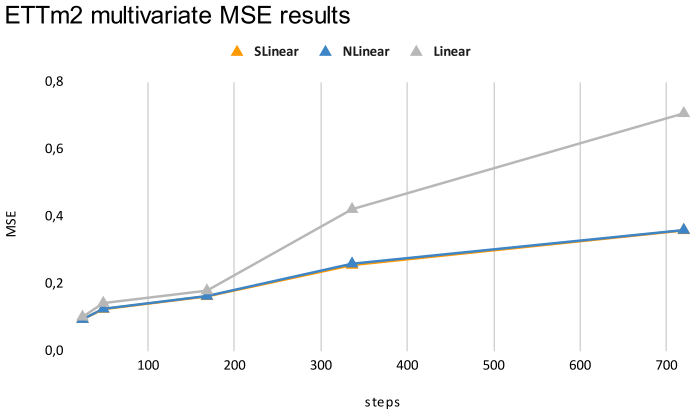
Table 5.2. Percentage Reduction in LTSF Prediction Performance: *SLinear* vs. Linear-Based Models. The table shows results for both Mean Square Error (MSE) and Mean Absolute Error (MAE) across 5 benchmarks. The *Average Reduction* row shows the averaged results across different datasets, as well as the global average reduction for each model.



(a) ETTh1 dataset



(b) Etth2 dataset

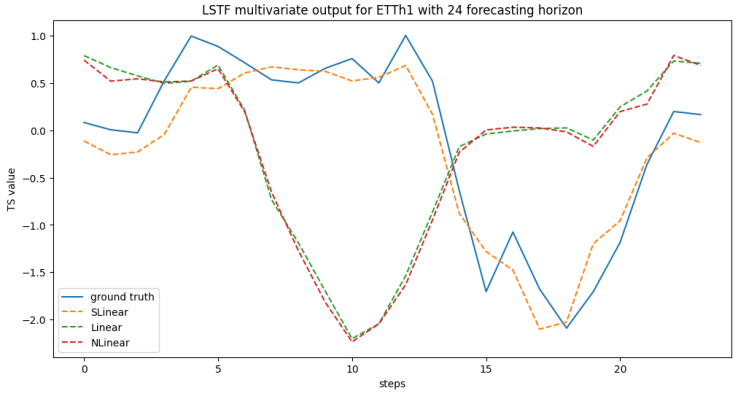


(c) Ettm2 dataset

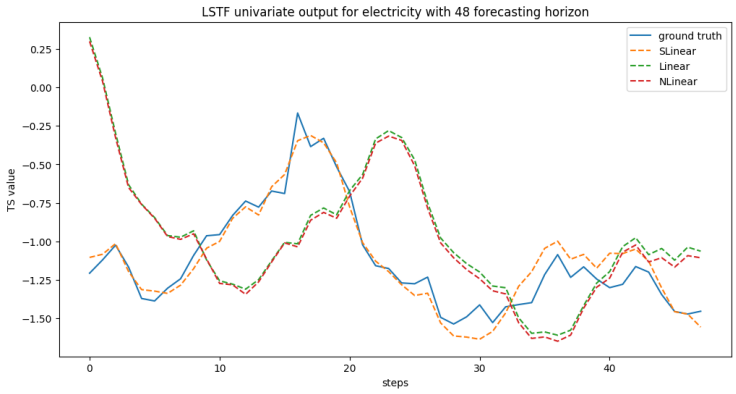
Figure 5.1. MSE results in multivariate settings for: SLinear, NLinear and Linear models

5.3.2 Qualitative Results

To determine the effectiveness of our novel approach, we perform a thorough comparison of the predictions made by SLinear, NLinear and Linear against the ground truth. Through various use cases, we demonstrate how SLinear excels at capturing scale and future data trends.
Figure 5.2 provides a visual comparison of all models against two benchmarks. These plots highlight how SLinear's Normalization module enables robust generalization on real-world datasets such as Electricity. This improvement is even more noticeable in Figure 5.3, where our model shows superior prediction accuracy.
In addition, our models show improved performance in dealing with the non-stationary nature of real-world data. This is particularly evident in Figure 5.4, where the linear model generates a series that lacks generalization, failing to account for the high non-stationarity characteristic of real-world datasets. Conversely, both NLinear and SLinear produce a non-stationary series that closely follows the ground truth.

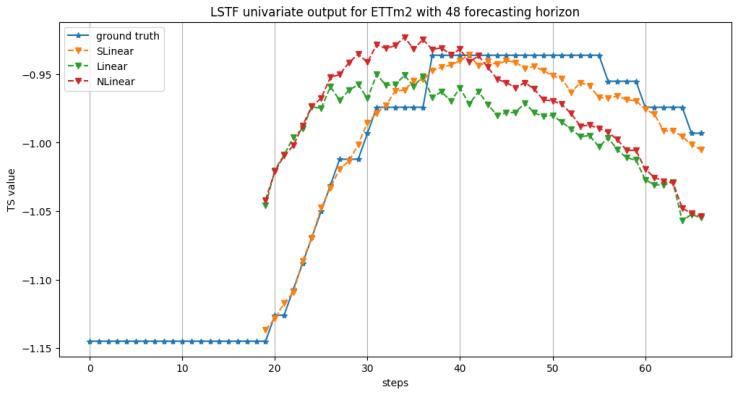


(a) ETTh1 univariate dataset, 24 prediction lenght

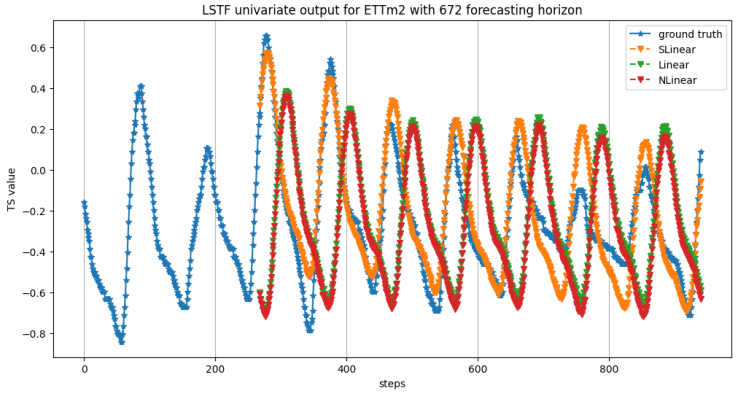


(b) Electricity univariate dataset, 48 prediction lenght

Figure 5.2. 24,48 prediction lenght comparative plots



(a) ETTm2 univariate dataset, 48 prediction lenght



(b) ETTm2 univariate dataset, 672 prediction lenght

Figure 5.3. ETTm2 comparative plots

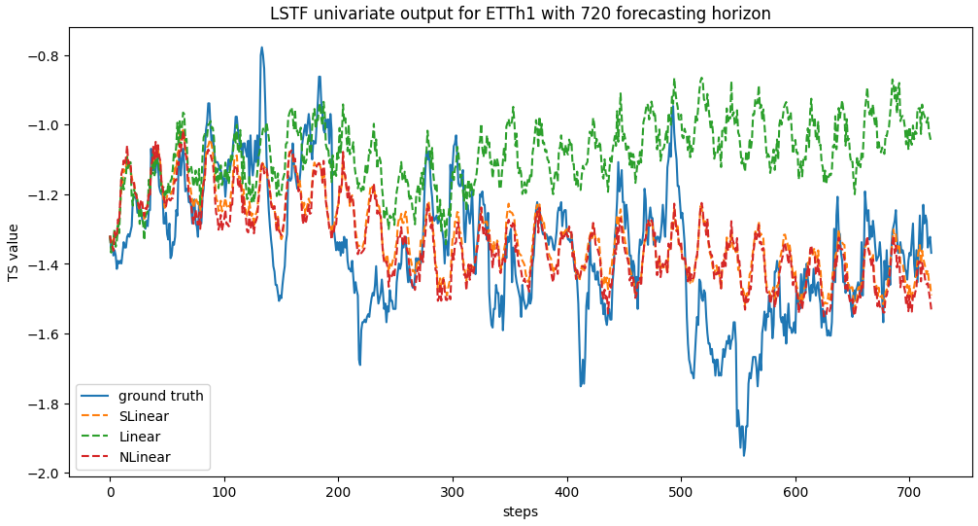


Figure 5.4. ETTh1 720 prediction lenght comparative plots

5.4 Ablation Studies

5.4.1 Normalization Module Analysis

We performed a comparative analysis with other popular normalization techniques commonly used in the LSTF literature in order to fully evaluate the effectiveness of the proposed SLinear Normalization Module. Inspired by the work of [24], we constructed a new model that exploits the linear baseline and incorporates the DAIN layer. This specialised layer autonomously learns the optimal data normalization and dynamically adjusts the applied normalization scheme during inference. We call this architecture DaLinear, a general scheme is shown in Figure 5.5.

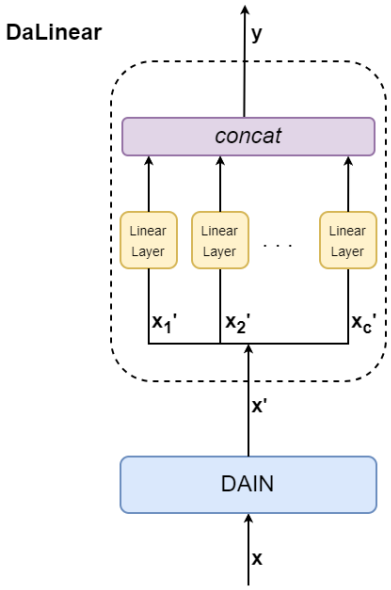


Figure 5.5. Overall *DaLinear* schema

Quantitative results for multivariate settings across all benchmarks are detailed in Table 5.3. It is notable that SLinear consistently outperforms DaLinear on almost all benchmarks, showing a significant improvement. On average, SLinear shows a 17% reduction in MSE compared to DaLinear. This underscores the importance of directly addressing the stationarity of time series for the effectiveness of the model in dealing with specific time dependencies.
In particular, it is imperative to reintegrate the statistics prior to the final model output. To address this, we have incorporated a De-Normalization module within the SLinear architecture. The overall percentage improvements based on the MSE

metric are shown in Table 5.4.

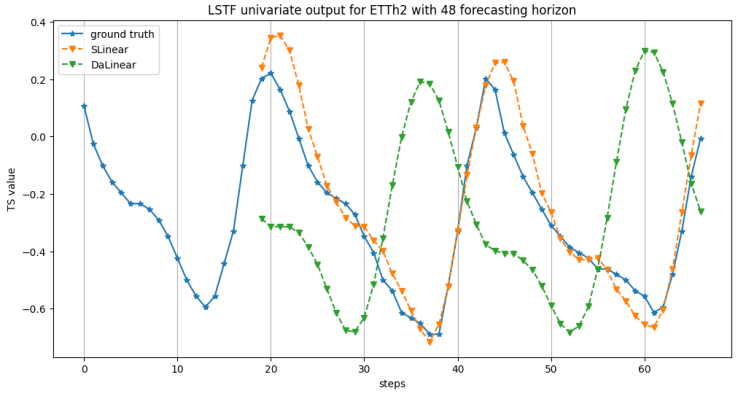
Methods		SLinear		DaLinear	
Metrics		MSE	MAE	MSE	MAE
ETTh1	24	**0.308**	**0.355**	0.313	0.365
	48	**0.344**	**0.375**	0.354	0.389
	168	**0.407**	**0.416**	0.424	0.437
	336	**0.44**	**0.437**	0.465	0.459
	720	**0.467**	**0.47**	0.5	0.494
ETTh2	24	**0.176**	0.274	0.278	**0.272**
	48	**0.231**	**0.313**	0.387	0.437
	168	**0.344**	**0.395**	0.69	0.583
	336	**0.392**	**0.434**	0.926	0.71
	720	**0.459**	**0.474**	1.336	0.895
ETTm1	24	**0.209**	**0.286**	0.212	0.293
	48	**0.264**	**0.323**	0.269	0.331
	96	**0.294**	**0.34**	0.298	0.345
	288	**0.362**	**0.379**	0.368	0.389
	672	**0.427**	**0.415**	0.431	0.425
ETTm2	24	**0.095**	**0.189**	0.102	0.205
	48	**0.125**	**0.22**	0.157	0.26
	96	**0.163**	**0.25**	0.207	0.304
	288	**0.256**	**0.314**	0.565	0.516
	672	**0.359**	**0.379**	0.953	0.67
Electricity	24	0.104	**0.202**	**0.103**	**0.202**
	48	0.119	**0.216**	**0.118**	0.217
	168	0.148	**0.241**	**0.147**	0.245
	336	0.169	**0.261**	**0.166**	0.267
	720	0.207	**0.293**	**0.203**	0.3

Table 5.3. LTSF forecast results in multivariate settings on several prediction lenghts, we report MSE and MAE metrics on 5 benchmarks, we highlight the best results in **bold**.

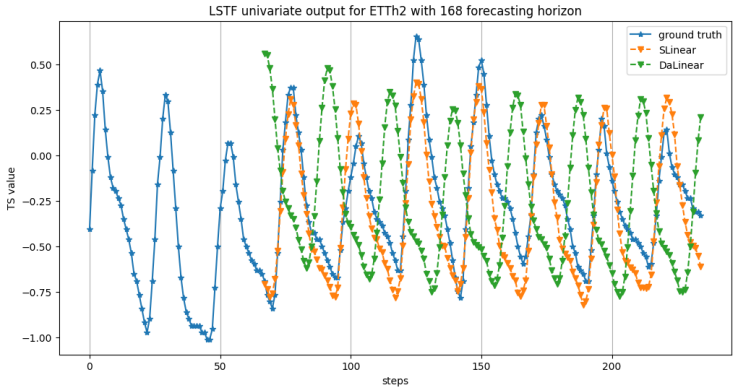
Metrics (%)		MSE	MAE
ETTh1	24	1.60	2.74
	48	2.82	3.60
	168	4.01	4.81
	336	5.38	4.79
	720	6.60	4.86
ETTh2	24	36.69	-0.74
	48	40.31	28.38
	168	50.14	32.25
	336	57.67	38.87
	720	65.64	47.04
ETTm1	24	1.42	2.39
	48	1.86	2.42
	96	1.34	1.45
	288	1.63	2.57
	672	0.93	2.35
ETTm2	24	6.86	7.80
	48	20.38	15.38
	96	21.26	17.76
	288	54.69	39.15
	672	62.33	43.43
Electricity	24	-0.97	0.00
	48	-0.85	0.46
	168	-0.68	1.63
	336	-1.81	2.25
	720	-1.97	2.33
Average Reduction	ETTh1	4.08	4.16
	ETTh2	50.09	29.16
	ETTm1	1.44	2.24
	ETTm2	33.10	24.70
	Electricity	-1.26	1.33
	Total	17.49	12.32

Table 5.4. Percentage reduction in LTSF prediction performance of SLinear vs. DaLinear. The table shows results for both Mean Square Error (MSE) and Mean Absolute Error (MAE) across 5 benchmarks. The *Average Reduction* row shows the averaged results across different datasets, as well as the global average reduction.

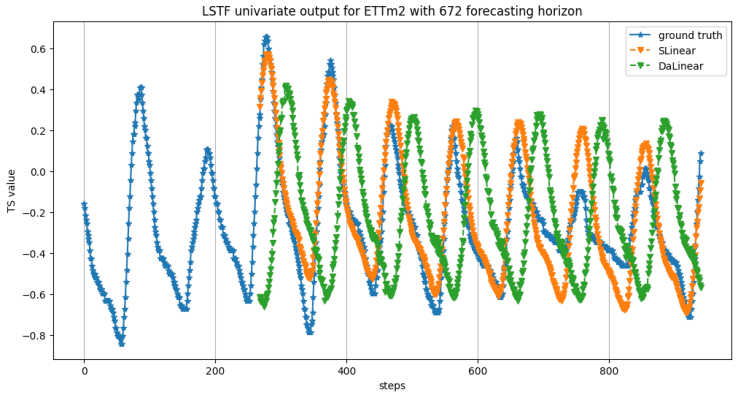
We further validated our predictions by comparing them to several benchmarks, allowing for a more comprehensive qualitative analysis. These results, shown in Figure 5.6, are consistent with previously reported findings across all prediction lengths.



(a) ETTh2 univariate dataset, 48 prediction lenght



(b) ETTh2 univariate dataset, 336 prediction lenght



(c) ETTm2 univariate dataset, 672 prediction lenght

Figure 5.6. Comparative prediction plots, SLinear vs. DaLinear

Chapter 6

Studies on Representation Learning Framework

In this chapter, we will focus our studies in finding a new effective way to explore transformers in LSTF task, specifically, we want to explore the effectiveness of transformer-based model as encoder for representation learning framework.

6.1 CoST-based Models Analysis

The work of Zeng et al.,2023[37] conducted a thorough investigation and comparison of transformer-based and linear-based models, with a predominant focus on the conventional end-to-end approach to transformers. Our aim is to explore the potential limitations of transformers in different contexts, particularly their role as encoders within a temporal contrastive learning framework.

This investigation will focus specifically on the CoST framework, mentioned in Section 3.3.5. This will involve a detailed exploration of different encoders to assess the efficacy of this architectural approach, together with an assessment of the capabilities of transformers when used as encoders within this network paradigm. To facilitate our research, we have implemented a modular CoST pipeline that allows us to easily swap and evaluate different encoders in the CoST architecture. For the completeness, we can briefly recall

6.1.1 Training and Testing Phases

All implementation details remain consistent with those outlined in the original paper. The training phase is divided into two distinct phases:

- Learning the time series representation coder, using a dedicated loss function.

- Then, a ridge regressor is trained on the learned representation, using the ground truth as labels.

The evaluation also consists of two phases:

- Computing the time series latent representations using the learned encoder.
- Predicting the future values using the trained ridge regressor.

Note how the representation encoder has to be trained once for different forecast horizons.

6.1.2 Results

While CoST uses a TCN [2] as its encoder, we have chosen to test several alternatives. For transformer-based encoders, we've chosen the Pyraformer [17]. For linear-based models, we've considered both Linear and NLinear architectures.
The comprehensive results are presented in Table 6.1, where we've performed a comparative analysis of the baseline models and our proposed variants. In particular, it appears that the Pyraformer is not as effective in capturing a significant latent representation. In fact, its inclusion seems to lead to a significant reduction in the initial CoST performance. Surprisingly, a simple linear encoder skilfully captures a meaningful latent representation, outperforming both the original TCN and the transformer-based encoder. This highlights that the transformer, especially in the context of LSTF, may exhibit overstated performance, not only in end-to-end tasks, but also when used as the backbone of a contrastive learning network.

Methods		CoST		Pyraformer		CoSPy		CoST-Nlinear		CoST-Linear	
Metrics		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.39	0.431	0.493	0.504	0.506	0.514	1.226	0.846	**0.371**	**0.414**
	48	0.439	0.465	0.55	0.542	0.544	0.537	1.25	0.861	**0.42**	**0.449**
	168	0.642	0.582	0.785	0.677	0.726	0.636	1.332	0.905	**0.627**	**0.571**
	336	**0.793**	0.672	0.899	0.738	0.854	0.705	1.392	0.934	0.795	**0.67**
	720	**0.903**	**0.748**	0.968	0.784	1.032	0.797	1.381	0.938	0.928	0.755
ETTh2	24	0.458	0.511	0.479	0.542	1.519	1.004	1.71	1.027	**0.352**	**0.431**
	48	0.718	0.648	0.893	0.743	1.656	1.052	1.721	1.045	**0.572**	**0.571**
	168	1.513	0.98	4.964	1.748	2.296	1.234	1.958	1.1	**1.459**	**0.924**
	336	**1.717**	1.044	4.511	1.78	2.166	1.196	1.952	1.068	1.732	**1.001**
	720	**1.924**	1.085	4.439	1.816	2.296	1.244	2.478	1.165	1.982	**1.077**
ETTm1	24	0.247	0.329	0.334	0.387	0.395	0.444	0.884	0.66	**0.23**	**0.313**
	48	0.332	0.387	0.469	0.471	0.436	0.466	0.963	0.7	**0.308**	**0.367**
	96	0.377	0.418	0.553	0.522	0.465	0.486	0.938	0.694	**0.357**	**0.401**
	288	0.473	0.486	0.763	0.667	0.54	0.534	0.962	0.715	**0.449**	**0.467**
	672	0.624	0.575	0.9	0.728	0.65	0.602	1.088	0.785	**0.611**	**0.564**
ETTm2	24	0.137	0.265	0.204	0.33	0.513	0.566	0.762	0.671	**0.11**	**0.225**
	48	0.2	0.325	0.343	0.5	0.593	0.614	0.804	0.697	**0.165**	**0.287**
	96	0.308	0.408	0.415	0.442	0.703	0.677	0.884	0.741	**0.271**	**0.376**
	288	0.758	0.662	1.203	0.866	1.183	0.891	1.275	0.904	**0.693**	**0.636**
	672	1.556	0.976	4.345	1.586	2.674	1.322	1.935	1.075	**1.533**	**0.941**
Electricity	24	0.134	0.24	0.17	**0.199**	0.187	0.299	**0.121**	**0.219**	0.131	0.234
	48	0.152	0.256	0.264	**0.214**	0.201	0.309	**0.139**	**0.236**	0.149	0.25
	168	0.175	0.275	0.719	**0.255**	0.224	0.325	**0.165**	**0.26**	0.172	0.27
	336	0.195	0.296	1.544	0.285	0.242	0.339	**0.185**	**0.282**	0.192	0.291
	720	0.231	0.328	4.151	0.325	0.269	0.36	**0.224**	**0.317**	0.229	0.325

Table 6.1. LTSF forecast results in multivariate settings on several prediction lenghts, we report MSE and MAE metrics on 5 benchmarks, we highlight the best results in **bold** and the second best in underline. The formatting of the model name in the table has is CoST-[ENCODER], except for CoSPy, which consists of a CoST architecture with a modified Pyraformer encoder.

6.2 Lightening of CoST Training

During the training of CoST-based models, we observed that the results of the lightweight training model showed negligible deviation from those obtained by full training. We deliberately chose to focus only on the vanilla CoST implementation, omitting the temporal contrastive learning training altogether. In principle, we initialise all model weights at random.

The results provided some interesting insights. The overall results remained largely consistent, but an unexpected revelation emerged in the ETTm2 benchmark. Here, the untrained iteration of CoST performed better than its trained counterpart. This

surprising result suggests that temporal contrastive training may not be as effective in capturing a meaningful latent representation of the dataset. Instead, a significant portion of the model's effectiveness is attributed to the ridge regressor integrated on top of the CoST framework, which is responsible for making the final prediction.

Furthermore, it is worth noting that contrastive training phase is computationally intensive, which significantly slows down the training process. While this observation is an additional insight, we expect that it may provide valuable considerations for other researchers within the LSTF community.

Methods		CoST		CoST*	
Metrics		MSE	MAE	MSE	MAE
ETTh1	24	**0.39**	0.431	**0.39**	0.431
	48	**0.439**	0.465	0.44	0.465
	168	**0.642**	0.582	0.654	0.589
	336	**0.793**	0.672	0.841	0.689
	720	**0.903**	0.748	1.019	0.784
ETTh2	24	0.458	0.511	**0.359**	**0.453**
	48	0.718	0.648	**0.6**	**0.598**
	168	**1.513**	0.98	1.595	1.007
	336	**1.717**	1.044	1.845	1.097
	720	**1.924**	1.085	2.038	1.172
ETTm1	24	**0.247**	0.329	0.249	0.331
	48	**0.332**	0.387	**0.332**	0.388
	96	**0.377**	0.418	0.388	0.429
	288	**0.473**	0.486	0.484	0.5
	672	**0.624**	0.575	0.637	0.587
ETTm2	24	0.137	0.265	**0.126**	**0.255**
	48	0.2	0.325	**0.186**	**0.32**
	96	0.308	0.408	**0.294**	**0.405**
	288	0.758	0.662	**0.755**	0.666
	672	**1.556**	0.976	1.732	1.044

Table 6.2. Comparison between fully trained version of CoST and untrained version of cost, shown as CoST*.

Chapter 7

Conclusion

7.1 Summary of Findings

In our work, we have explored the potential and limitations of linear-based models in LSTF tasks, through an exhaustive study of normalization strategies. We have found how these techniques can help to better deal with non-stationary real-world datasets, achieving a significant improvement in terms of MSE and MAE. We demonstrate the ineffectiveness of transformer-based models as encoders in a representation learning framework, and the clear superiority of a simple linear encoder. Finally, we have seen how heavy temporal contrastive training in CoST does not help much in improving the performance of the overall model, and we can obtain a similar (even better) result with a randomly initialised model.

7.2 Practical Implications

The results of this research have several practical implications for the field of long sequence time series forecasting and related fields. Our simple linear-based model could deal with non-stationary real-world datasets, which has important implications in all application areas of LSTF, including economics, finance, energy and healthcare. In a world where resources are becoming an increasingly important commodity, avoiding waste through efficient management is a necessity. In this scenario, the ability to accurately predict when a given event will occur becomes critical. Our studies on representation learning frameworks and transformer-based models are also fundamental, as they highlight how often computational resources are not optimally exploited.

7.3 Limitation and Future Research Directions

While our studies have provided some interesting insights, we need to be aware of some limitations.
Starting from our novel model SLinear, we evaluate and test its performance on the most popular LSTF benchmarks. Furthermore, we focus our research on a limited set of normalization techniques applied only to a simple linear-based model.
Considering the research on transformers encoders and contrastive representation learning frameworks, we focus mainly on Pyraformer and CoST respectively.
In terms of computational resources, our study was carried out on a low-resource machine. Extending our experiments to a better configuration could provide some valuable insights into the robustness of our methods.

These limitations provide a good starting point for further research. In particular

- It may be interesting to evaluate the real generalizability of the model to different real-world scenarios, as well as to additionally study normalization techniques and their integration within linear-based models, but also some variations and enhancement of them.
- Exploring alternative transformer-based encoders and representation learning frameworks, as well as identifying different applications of transformer-based models in the LSTF domain, could provide invaluable insights into the true potential of these architectures.

In conclusion, we believe that our study can guide future researchers towards more efficient and accurate solutions, as well as be of benefit to the entire LSTF community.

Bibliography

[1] Charu C Aggarwal et al. *Neural networks and deep learning*. Springer, 2018.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[3] Pierre Baldi and Peter J Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26, 2013.

[4] Michelangelo Ceci, Roberto Corizzo, Nathalie Japkowicz, Paolo Mignone, and Gianvito Pio. Echad: embedding-based change detection from multivariate time series in smart grids. *IEEE Access*, 8:156053–156066, 2020.

[5] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. *Advances in neural information processing systems*, 30, 2017.

[6] Christopher Chatfield. *The analysis of time series: theory and practice*. Springer, 2013.

[7] Zonglei Chen, Minbo Ma, Tianrui Li, Hongjun Wang, and Chongshou Li. Long sequence time-series forecasting with deep learning: A survey. *Information Fusion*, 97:101819, 2023.

[8] Shengdong Du, Tianrui Li, Yan Yang, and Shi-Jinn Horng. Multivariate time series forecasting via attention-based encoder–decoder framework. *Neurocomputing*, 388:269–279, 2020.

[9] Chenyou Fan, Yuze Zhang, Yi Pan, Xiaoyue Li, Chi Zhang, Rong Yuan, Di Wu, Wensheng Wang, Jian Pei, and Heng Huang. Multi-horizon time series forecasting with temporal attention learning. In *Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery & data mining*, pages 2527–2535, 2019.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[11] Siteng Huang, Donglin Wang, Xuehan Wu, and Ao Tang. Dsanet: Dual self-attention network for multivariate time series forecasting. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2129–2132, 2019.

[12] Yoonhwa Jung, Jaehoon Jung, Byungil Kim, and SangUk Han. Long short-term memory recurrent neural network for modeling temporal patterns in long-term power forecasting for solar pv facilities: Case study of south korea. *Journal of Cleaner Production*, 250:119476, 2020.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Pedro Lara-Benítez, Manuel Carranza-García, and José C Riquelme. An experimental review on deep learning architectures for time series forecasting. *International journal of neural systems*, 31(03):2130001, 2021.

[15] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in neural information processing systems*, 32, 2019.

[16] Minhao Liu, Ailing Zeng, Muxi Chen, Zhijian Xu, Qiuxia Lai, Lingna Ma, and Qiang Xu. Scinet: Time series modeling and forecasting with sample convolution and interaction. *Advances in Neural Information Processing Systems*, 35:5816–5828, 2022.

[17] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*, 2021.

[18] Xiaolei Liu and Zi Lin. Impact of covid-19 pandemic on electricity demand in the uk based on multivariate time series forecasting with bidirectional long short term memory. *Energy*, 227:120455, 2021.

[19] Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Exploring the stationarity in time series forecasting. *Advances in Neural Information Processing Systems*, 35:9881–9893, 2022.

[20] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

[21] Qianwen Meng, Hangwei Qian, Yong Liu, Yonghui Xu, Zhiqi Shen, and Lizhen Cui. Unsupervised representation learning for time series: A review. *arXiv preprint arXiv:2308.01578*, 2023.

[22] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

[23] Eduardo Ogasawara, Leonardo C Martinez, Daniel De Oliveira, Geraldo Zimbrão, Gisele L Pappa, and Marta Mattoso. Adaptive normalization: A novel data normalization approach for non-stationary time series. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010.

[24] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Deep adaptive input normalization for time series forecasting. *IEEE transactions on neural networks and learning systems*, 2019.

[25] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[26] H Robert et al. Time series analysis and its applications with r examples second edition, 2006.

[27] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*, 2019.

[28] Slawek Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

[29] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing*, 93:106401, 2020.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[31] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.

[32] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

[33] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. CoST: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. In *International Conference on Learning Representations*, 2022.

[34] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[35] Shuhei Yoshimi and Koji Eguchi. Forecasting corporate financial time series using multi-phase attention recurrent neural networks. In *EDBT/ICDT Workshops*, 2020.

[36] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[37] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, 2023.

[38] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, 2021.

[39] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pages 27268–27286. PMLR, 2022.