Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Artificial intelligence techniques and their application to time series forecast

Autor: Antonio Luis González Hernández

Tutor: Juan Antonio Becerra González

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Artificial intelligence techniques and their application to time series forecast

Autor:

Antonio Luis González Hernández

Tutor:

Juan Antonio Becerra González

Profesor Titular

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado:	Artificial intelligence techniques and their application to time series forecast

Autor:	Antonio Luis González Hernández
Tutor:	Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Índice Abreviado

Contents

1 History and State of the Art

1.1 Evolution of Methods for Time Series Forecasting

Time series forecasting is a fundamental discipline in various fields such as economics, meteorology, engineering, and social sciences. Throughout history, this discipline has evolved significantly, developing increasingly sophisticated methods to improve the accuracy and reliability of predictions. In this chapter, we will explore the evolution of time series forecasting methods, from the most traditional approaches to the most modern ones based on machine learning techniques and neural networks.

1.1.1 Traditional Methods

Traditional time series forecasting methods have been the cornerstone of predictive analytics for decades. These methods typically rely on well-established statistical principles to analyze historical data and make future predictions based on observed patterns and relationships. A common feature of traditional methods is their emphasis on linear relationships and the assumption that historical patterns will persist into the future. They are grounded in mathematical theories and formulas, providing a structured and straightforward approach to forecasting. This simplicity contributes to their ease of interpretation, making it easier for practitioners to understand and communicate results.

Moreover, traditional methods often assume that the time series data is stationary, meaning its statistical properties, such as mean and variance, do not change over time. As a result, techniques within these methods frequently involve transforming non-stationary data into a stationary format. They focus on capturing dependencies between current and past values of the time series, often through lagged values, and incorporate mechanisms to handle seasonality and trends, adjusting for periodic fluctuations and long-term movements in the data. Parameter estimation is a key aspect, with these methods typically involving the calculation of a set number of parameters based on historical data.

Autoregressive and Moving Average Models (AR, MA, ARMA)

The first time series forecasting models included autoregressive (AR) and moving average (MA) models. These models are linear and are based on the assumption that the time series can be explained by a linear combination of its past values and an error term. The ARMA model combines both approaches to provide a more robust tool for time series analysis.

Autoregressive Models (AR) of order p (AR(p)) uses the p previous values of the time series to make predictions. The general formula for an AR(p) model is:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t$$

where:

- X_t is the value of the time series at time t.
- c is a constant.
- ϕ_i are the model coefficients.

- ε_t is the error term at time t.

Moving Average Models (MA) of order q (MA(q)) uses q previous error terms to make predictions. The general formula for an MA(q) model is:

$$X_t = \mu + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where:

- μ is the mean of the time series.
- θ_j are the coefficients of the model.
- ε_t is the error term at time t.

Autoregressive Moving Average (ARMA) models combines both approaches to capture both autoregressive and moving average dependence. The general formula for an ARMA(p,q) model is:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where:

- X_t is the value of the time series at time t.
- c is a constant.
- ϕ_i are the autoregressive coefficients.
- θ_j are the moving average coefficients.
- ε_t is the error term in time t.

Autoregressive Integrated Moving Average (ARIMA) and SARIMA Models

For non-stationary time series, integrated autoregressive moving average (ARIMA) models were introduced. These models include a differencing term to handle non-stationarity. The SARIMA model extends ARIMA by including seasonal components, providing a more robust tool for the analysis of time series with seasonal patterns.

Autoregressive Moving Average Integrated Models (ARIMA) combines the autoregressive (AR), differencing (I), and moving average (MA) components. The general formula for an ARIMA(p,d,q) model is:

$$\Delta^d X_t = c + \sum_{i=1}^{p} \phi_i \Delta^d X_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

where:

- X_t is the value of the time series at time t.
- Δ^d represents the order differentiation d.
- c is a constant.
- ϕ_i are the autoregressive coefficients.
- θ_j are the moving average coefficients.
- ε_t is the error term in time t.

The SARIMA model extends the ARIMA model by including seasonal components. The general formula for a SARIMA(p,d,q)(P,D,Q,m) model is:

$$\Delta^d \Delta_m^D X_t = c + \sum_{i=1}^{p} \phi_i \Delta^d \Delta_m^D X_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \sum_{k=1}^{P} \Phi_k \Delta_m^D X_{t-km} + \sum_{l=1}^{Q} \Theta_l \varepsilon_{t-lm}$$

where:

- X_t is the value of the time series at time t.
- Δ^d represents the differencing of order d.
- Δ_m^D represents the seasonal differencing of order D with periodicity m.
- c is a constant.
- ϕ_i are the autoregressive coefficients.
- θ_j are the moving average coefficients.
- Φ_k are the seasonal autoregressive coefficients.
- Θ_l are the seasonal moving average coefficients.
- ε_t is the error term at time t.

SARIMAX models

SARIMAX models (Seasonal ARIMA with eXogenous regressors) are an extension of SARIMA models that allow the inclusion of exogenous variables in time series analysis. These exogenous variables are external factors that can influence the time series and, by incorporating them, the accuracy of the predictions is improved. The general structure of a SARIMAX model can be expressed as:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \beta_1 X_{1,t} + \beta_2 X_{2,t} + \cdots + \beta_k X_{k,t} + \varepsilon_t$$

where:

- Y_t is the value of the time series at time t.
- ϕ_i are the autoregressive coefficients.
- θ_i are the moving average coefficients.
- ε_t is the error term at time t.
- $X_{i,t}$ are the exogenous variables at time t.
- β_i are the coefficients of the exogenous variables.

VARMA and VARMAX Models

VARMA (Vector AutoRegressive Moving Average) models are a multivariate extension of ARMA models, suitable for analyzing time series with multiple interrelated variables. VARMAX (Vector AutoRegressive Moving Average with eXogenous regressors) models include additional exogenous variables, allowing the incorporation of external information that can influence the time series. VARMA models are useful when you have multiple time series that can influence each other. The general structure of a VARMA(p, q) model can be expressed as:

$$Y_t = \Phi_1 Y_{t-1} + \Phi_2 Y_{t-2} + \cdots + \Phi_p Y_{t-p} + \Theta_1 \varepsilon_{t-1} + \Theta_2 \varepsilon_{t-2} + \cdots + \Theta_q \varepsilon_{t-q} + \varepsilon_t$$

where:

- Y_t is a vector of time series values at time t.
- Φ_i are autoregressive coefficient matrices.
- Θ_i are moving average coefficient matrices.
- ε_t is a vector of error terms at time t.

Simple Exponential Smoothing (SES) and Holt-Winters Exponential Smoothing (HWES)

Simple Exponential Smoothing (SES) and Holt-Winters Exponential Smoothing (HWES) are time series forecasting methods that use exponential smoothing techniques to capture patterns in the data. These methods are especially useful for time series with trend and seasonality components.

Simple Exponential Smoothing (SES) is a forecasting technique that assigns exponentially decreasing weights to past observations. The general formula for SES is:

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t$$

where:

- $\hat{Y}_{t+1}$ is the forecast for time $t+1$.
- Y_t is the observed value at time t.
- $\hat{Y}_t$ is the forecast for time t.
- α is the smoothing parameter $(0 < \alpha < 1)$.

Holt-Winters Exponential Smoothing (HWES) is an extension of SES that incorporates trend and seasonality components. There are two main variants of the Holt-Winters method: additive and multiplicative.

The Additive Method is suitable for time series with constant seasonality over time. The equations of the additive method are:

- Level:
$$L_t = \alpha(Y_t - S_{t-m}) + (1-\alpha)(L_{t-1} + T_{t-1})$$
- Trend:
$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$$
- Seasonality:
$$S_t = \gamma(Y_t - L_t) + (1-\gamma)S_{t-m}$$
- Forecast:
$$\hat{Y}_{t+h} = L_t + hT_t + S_{t+h-m(k+1)}$$

where:

- L_t is the level at time t.
- T_t is the trend at time t.
- S_t is the seasonal component at time t.
- m is the number of periods in a season.
- h is the forecast horizon.
- α, β, and γ are the smoothing parameters.

The Multiplicative Method is suitable for time series with seasonality that varies proportionally with the level of the series. The equations of the multiplicative method are:

- Level:
$$L_t = \alpha\frac{Y_t}{S_{t-m}} + (1-\alpha)(L_{t-1} + T_{t-1})$$
- Trend:
$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$$
- Seasonality:
$$S_t = \gamma\frac{Y_t}{L_t} + (1-\gamma)S_{t-m}$$

- Forecast:
$$\hat{Y}_{t+h} = (L_t + hT_t)S_{t+h-m(k+1)}$$

ARCH and GARCH Models

ARCH (Autoregressive Conditional Heteroskedasticity) and GARCH (Generalized ARCH) models are widely used in financial time series analysis, where the volatility of the data can change over time. These models are especially useful for capturing conditional variability in financial data, such as stock prices or exchange rates.

The ARCH model, proposed by Robert Engle in 1982, models the conditional variance of a time series as a function of past errors. The general structure of an ARCH(p) model can be expressed as follows:

$$Y_t = \mu + \varepsilon_t$$

$$\varepsilon_t = \sigma_t Z_t$$

$$\sigma_t^2 = \alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \alpha_2 \varepsilon_{t-2}^2 + \cdots + \alpha_p \varepsilon_{t-p}^2$$

where:

- Y_t is the value of the time series at time t.
- μ is the mean of the time series.
- ε_t is the error term at time t.
- σ_t^2 is the conditional variance at time t.
- Z_t is a random variable with standard normal distribution.
- $\alpha_0, \alpha_1, \ldots, \alpha_p$ are the parameters of the model.

The GARCH model, developed by Tim Bollerslev in 1986, is an extension of the ARCH model that allows for a more parsimonious and flexible representation of conditional variance. Instead of relying solely on past errors, the GARCH model also incorporates past conditional variance. This model is particularly useful because it can capture persistence in volatility, meaning that shocks to conditional variance can have long-lasting effects. This is especially relevant in financial analysis, where periods of high volatility tend to cluster.

The general structure of a GARCH(p, q) model can be expressed as:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

where:

- σ_t^2 is the conditional variance at time t.
- α_0 is a constant.
- α_i are the coefficients of the ARCH terms.
- β_j are the coefficients of the GARCH terms.
- p is the number of lags of the ARCH terms.
- q is the number of lags of the GARCH terms.

1.1.2 Modern Methods

Modern time series forecasting methods leverage advancements in machine learning and computational power to address the complexities of contemporary data. These approaches are designed to handle large datasets and uncover intricate patterns that traditional methods might overlook. One of the defining features of modern methods is their ability to model non-linear relationships, capturing complex interactions within the data that traditional linear models might miss. They are highly flexible and adaptable, accommodating various types of data structures and managing non-stationary series without requiring explicit transformations.

Modern methods rely heavily on data-driven techniques, utilizing vast amounts of data to train models and improve predictive accuracy. Sophisticated feature engineering is often employed to extract relevant patterns and trends from raw data, enhancing model performance. Additionally, these methods can handle high-dimensional datasets with multiple variables, effectively capturing interdependencies and interactions. Many modern approaches use ensemble learning techniques, combining multiple models to improve robustness and accuracy. They are also designed to scale efficiently with the growing size of datasets, leveraging parallel processing and advanced computational resources. Furthermore, advanced optimization algorithms are used to fine-tune model parameters, achieving optimal performance through iterative learning processes.

Recurrent Neural Networks (RNNs) and LSTMs

Recurrent Neural Networks (RNN) and their Long Short-Term Memory (LSTM) variants have revolutionized the field of time series forecasting. These deep learning techniques are capable of capturing long-term dependencies in data, which makes them especially useful for this type of task.

RNNs were introduced in the 1980s as an extension of traditional neural networks, designed to process sequences of data. Unlike feedforward networks, RNNs have recurrent connections that allow information to persist. This means that they can use their internal memory to process sequences of inputs, which is crucial for tasks where temporal context is important. RNNs are used in a variety of time series applications, such as stock price forecasting, energy demand, and weather data analysis. However, traditional RNNs have limitations when it comes to capturing long-term dependencies due to the gradient fading problem.

To overcome the limitations of traditional RNNs, Sepp Hochreiter and Jürgen Schmidhuber proposed **LSTMs** in 1997[1]. LSTMs are a variant of RNNs that include a memory architecture designed to handle long-term dependencies more efficiently. The key to LSTMs is their structure of memory cells, which can maintain and update information over many time steps. LSTMs are composed of memory cells that have three gates: the input gate, the forget gate, and the output gate. These gates control the flow of information in and out of the memory cell, allowing the network to remember or ignore information as needed.

LSTMs have proven to be extremely effective in time series forecasting. They are used in applications such as traffic forecasting, financial data anomaly detection, and sales forecasting. Thanks to their ability to handle long-term dependencies, LSTMs outperform traditional RNNs in many time series tasks.

Residual Networks (ResNet) and Fully Convolutional Networks (FCN)

Convolutional neural networks (CNNs) have revolutionized the field of machine learning, especially in computer vision tasks. However, their applications have been extended to other domains, including time series forecasting. Among the most prominent variants of CNNs are Residual Networks (ResNet) and Fully Convolutional Networks (FCN).

Both ResNets and FCNs have been successfully applied in time series forecasting. ResNets, with their ability to handle deep networks, are ideal for modeling complex and non-linear relationships in multivariate data. On the other hand, FCNs, with their focus on full convolution, allow detailed patterns to be captured and accurate predictions to be made in dense time series.

These architectures have proven to be powerful tools in time series analysis and forecasting, offering significant improvements in accuracy and efficiency compared to traditional methods such as ARIMA and SARIMAX.

Residual Networks (ResNet), introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in 2015, emerged as a solution to the problem of degradation in deep neural networks. This problem refers to the decrease in accuracy as more layers are added to a neural network. ResNets address this challenge by introducing skip connections that allow information to flow directly through the network, skipping one or more intermediate layers.

These skip connections facilitate the training of very deep networks, allowing the construction of models with hundreds of layers without significant degradation in performance. ResNets have proven effective in a wide range of applications, including time series forecasting, where their ability to handle complex, multivariate data is particularly valuable.

Fully Convolutional Networks (FCNs)[2] were proposed by Jonathan Long, Evan Shelhamer, and Trevor Darrell in 2015 as an extension of traditional CNNs for semantic segmentation tasks. Unlike conventional CNNs, FCNs replace fully connected layers with convolutional layers, allowing the network to process inputs of arbitrary size and produce outputs of corresponding size.

This architecture has been adapted for time series forecasting, taking advantage of its ability to capture spatial and temporal patterns in the data. FCNs are especially useful in scenarios where dense and detailed forecasting is required, such as in the case of multivariate time series.

ROCKET

ROCKET (Random Convolutional Kernel Transform)[3] is an innovative and recent method in the field of time series analysis. It was introduced by Angus Dempster, François Petitjean, and Geoffrey I. Webb in 2019. This method is notable for its ability to apply random convolutional transformations to time series data, extracting useful features that can be used by simple linear classifiers.

The fundamental principle of ROCKET is the use of a large number of one-dimensional (1-D) convolutional kernels that are randomly initialized. These kernels are applied to the time series to generate a transformed representation of the data. Unlike other methods that require extensive training of the kernels, in ROCKET the kernels are not trained, which significantly reduces the computational time.

Once the features have been extracted using convolutional transformations, linear classifiers, such as the RidgeClassifier, are used to perform time series classification. This approach allows ROCKET to be exceptionally fast and efficient, achieving high accuracy results at a fraction of the computational cost of other advanced methods.

ROCKET has proven to be highly effective in a variety of time series classification tasks. Its ability to handle large volumes of data and its computational efficiency make it ideal for real-time applications and in scenarios where computational resources are limited. In addition, its simplicity and speed allow it to be deployed on a wide range of platforms and environments.

Matrix Factorization with DTW

Matrix factorization is a fundamental technique in linear algebra that decomposes a matrix into products of simpler matrices. When combined with Dynamic Time Warping (DTW), a time alignment technique, it becomes a powerful tool for multivariate time series analysis.

Dynamic Time Warping (DTW)[4] is an algorithm that allows two time series to be aligned nonlinearly by adjusting for differences in the speed of the sequences. This is particularly useful in time series analysis where events may occur at different times but follow a similar pattern. Combining DTW with matrix factorization allows for improved temporal alignment between different series, facilitating the identification of common patterns.

Matrix factorization, on the other hand, decomposes a matrix into simpler matrix products, which facilitates data analysis and manipulation. By applying DTW in the context of matrix factorization, more accurate and aligned representations of multivariate time series can be obtained.

Matrix factorization combined with DTW represents an advanced and efficient technique for multivariate time series analysis. Its ability to improve time alignment and simplify data makes it a valuable tool for researchers and practitioners in various fields.

XGBoost

XGBoost[5], short for eXtreme Gradient Boosting, is a decision tree-based boosting algorithm that has gained popularity in the machine learning field due to its efficiency and accuracy. It was developed by Tianqi Chen and Carlos Guestrin in 2016. This algorithm has proven to be extremely effective for time series forecasting, especially because of its ability to handle tabular data efficiently.

XGBoost is an optimized implementation of the gradient boosting algorithm, which combines multiple weak decision trees to form a strong model. The boosting process involves the sequential construction of trees, where each new tree attempts to correct errors made by previous trees. This is achieved by minimizing a loss function, which allows the model to iteratively improve its accuracy.

One of the distinguishing features of XGBoost is its ability to handle tabular data efficiently. Tabular data, consisting of rows and columns, is common in many real-world applications, such as finance, medicine, and marketing. XGBoost excels in this type of data due to its ability to handle heterogeneous features and its computational efficiency.

Prophet

Prophet[6] is an open source tool developed by Facebook's Data Science team in 2017. It is designed for time series forecasting and is based on an additive component model, which facilitates the inclusion of trends and seasonalities automatically.

Prophet uses an additive model where the time series observations are decomposed into three main components: trend, seasonality, and holidays. The basic formula of the model is:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \tag{1.1}$$

Where:

- **Trend** $g(t)$: Captures the long-term evolution of the time series. It can be linear or logarithmic, and Prophet allows the inclusion of change points to model sudden changes in the trend.
- **Seasonality** $s(t)$: Models recurring patterns over different time periods, such as annual, weekly, or daily.
- **Holidays** $h(t)$: Allows the inclusion of effects of specific holidays that may influence the time series.
- **Error** ε_t: Represents noise or variations not explained by the previous components.

One of the advantages of Prophet is its ability to handle data with multiple and non-linear seasonalities, as well as its robustness to missing data and outliers.

Prophet represents a powerful and accessible tool for time series forecasting. Its approach based on an additive component model and its ability to handle complex data make it a preferred choice for analysts and data scientists worldwide.

TS-Chief and HIVE-COTE

TS-Chief[7] and HIVE-COTE[8] are advanced time series classification techniques based on ensemble learning. These techniques combine multiple classifiers to improve the accuracy and robustness of predictions by leveraging the strengths of different approaches.

TS-Chief is a time series classification technique that uses an approach based on decision trees and time series transformations. This method stands out for its ability to handle multivariate data and capture complex patterns in the time series. TSChef applies specific transformations to the time series to extract relevant features, which are then used by an ensemble of classifiers to make accurate predictions.

HIVE-COTE (Hierarchical Vote Collective of Transformation-based Ensembles) is a heterogeneous meta-assembly for time series classification. It was introduced by Anthony Bagnall and his team in 2016 and has been continuously updated to improve its performance. HIVE-COTE combines classifiers from multiple domains, including phase-independent shapelets, bag-of-words based dictionaries, and phase-dependent intervals. This combination allows HIVE-COTE to capture a wide variety of patterns in the time series, resulting in high accuracy and robustness in predictions.

1.2 Evolution of Transformers as a Method for Time Series Forecasting

The advent of Transformer models has revolutionized various fields of artificial intelligence, particularly natural language processing (NLP). Introduced by Vaswani et al. in 2017[9], the original Transformer architecture demonstrated unprecedented capabilities in handling sequential data, leading to significant advancements in machine translation, text generation, and more.

Initially, some researchers concluded that Transformers might not be well-suited for time series forecasting. For instance, the article "Are Transformers Effective for Time Series Forecasting?"[10] argues that simpler models, like the one-layer linear model LTSF-Linear, often outperform Transformer-based models in this specific task. The authors suggest that the permutation-invariant nature of the self-attention mechanism in Transformers can lead to a loss of temporal information, which is crucial for time series forecasting.

However, this perspective has been evolving with the development of new models and advancements in the field. The application of Transformers to time series forecasting began to gain traction around 2019, with researchers exploring how the self-attention mechanism could be leveraged to model temporal dependencies more effectively. Since then, a variety of Transformer-based models have been proposed, each addressing specific challenges associated with time series data, such as scalability, interpretability, and handling of seasonality and trends. Much of this information is compiled in the article “Transformers in Time Series: A Survey”[11], which provides a comprehensive overview of the evolution of Transformers in time series analysis.

This section delves into the evolution of Transformers in the context of time series forecasting. We will explore the chronological development of key Transformer variants, highlighting their unique contributions and innovations. From the Original Transformer to recent advancements like Crossformer and PatchTST, this section provides a comprehensive overview of how Transformers have been adapted and enhanced to meet the demands of time series forecasting.

1.2.1 Introduction of Transformers (2017)

Introduced by Vaswani et al.[9], the original Transformer architecture was primarily designed for natural language processing (NLP) tasks. It revolutionized the field by using a self-attention mechanism, which allowed the model to weigh the importance of different words in a sentence, regardless of their position. This architecture enabled parallel processing of data, significantly improving efficiency and performance in tasks like machine translation and text generation.

1.2.2 Application to Time Series (2019)

Temporal Fusion Transformer (TFT)
Proposed by Lim et al.[12], the Temporal Fusion Transformer was designed specifically for interpretable multi-horizon time series forecasting. TFT integrates both static and time-varying covariates, allowing it to handle complex temporal patterns and provide insights into the importance of different features over time. Its interpretability makes it particularly useful in domains where understanding the model’s decision-making process is crucial.

Informer
Introduced by Zhou et al.[13], the Informer model aimed to improve the efficiency and scalability of Transformers for long sequence time-series forecasting. It introduced a ProbSparse self-attention mechanism, which reduces the computational complexity by focusing on the most informative parts of the input sequence. This makes Informer more suitable for handling long sequences of time series data, where traditional Transformers would be computationally expensive.

1.2.3 Enhancements and New Variants (2020)

LogTrans
Developed by Li et al.[14], LogTrans focuses on capturing long-term dependencies in time series data. It introduces a logarithmic self-attention mechanism, which scales the attention scores logarithmically with the distance between elements in the sequence. This allows LogTrans to effectively model long-range dependencies without the quadratic complexity of standard self-attention, making it more efficient for long-term forecasting.

Reformer
Proposed by Kitaev et al.[15], the Reformer model aims to reduce the computational complexity of Transformers. It achieves this by using locality-sensitive hashing (LSH) to approximate the self-attention mechanism, significantly reducing the memory and computational requirements. Reformer also introduces reversible layers, which further decrease memory usage by allowing intermediate activations to be recomputed during backpropagation. This makes Reformer highly efficient and scalable for large-scale time series forecasting tasks.

1.2.4 Further Innovations (2021 - 2022)

Autoformer
Introduced by Wu et al.[16], Autoformer is designed for the automatic decomposition of time series into trend and seasonal components. This model leverages a decomposition block that separates the input time series into trend and seasonal parts, allowing the model to handle these components independently. By focusing

on these distinct aspects of the data, Autoformer can achieve more accurate and interpretable forecasts, particularly in scenarios where understanding the underlying patterns is crucial.

FEDformer
Proposed by Zhou et al.[17], FEDformer (Frequency Enhanced Decomposition Transformer) focuses on frequency domain decomposition for time series forecasting. This model applies a Fourier transform to decompose the time series into different frequency components, which are then processed separately by the Transformer. By working in the frequency domain, FEDformer can capture periodic patterns more effectively and improve the accuracy of forecasts, especially for data with strong seasonal or cyclical behavior.

Pyraformer
Developed by Liu et al.[18], Pyraformer aims to improve the efficiency of Transformers for long sequence time series forecasting. It introduces a pyramidal attention mechanism that progressively reduces the sequence length at each layer, focusing on the most relevant parts of the input. This hierarchical approach allows Pyraformer to handle long sequences more efficiently, reducing computational complexity while maintaining high forecasting accuracy.

ETSformer Introduced by Zhou et al.[19], ETSformer combines exponential smoothing with Transformer architecture to better handle seasonality and trend in time series data. By integrating exponential smoothing, which is effective for capturing trends and seasonal patterns, with the powerful self-attention mechanism of Transformers, ETSformer can provide more accurate and robust forecasts. This hybrid approach leverages the strengths of both methods, making it particularly useful for time series with pronounced seasonal and trend components.

1.2.5 Recent Advances (2023 - 2024)

Crossformer Proposed by Zhang et al.[20], Crossformer focuses on cross-dimension dependency modeling for multivariate time series. This model introduces a cross-dimension attention mechanism that captures dependencies between different dimensions of the data, allowing for more accurate and holistic forecasting in multivariate settings. By effectively modeling the interactions between various time series, Crossformer enhances the predictive performance for complex datasets.

PatchTST Developed by Nie et al.[21] , PatchTST leverages patch-based input representations for time series forecasting. This approach divides the input time series into smaller patches, which are then processed independently before being aggregated. This patch-based method allows the model to capture local patterns more effectively and improves scalability for long sequences. PatchTST is particularly useful for handling large and complex time series data.

CARD Developed by Xue et al.[22], Channel Aligned Robust Blend Transformer (CARD) focuses on aligning and blending information across different channels of multivariate time series data. This model introduces a channel alignment mechanism that ensures consistent and robust integration of information from various channels, enhancing the model's ability to capture intricate dependencies and improve forecasting accuracy.

SageFormer Introduced by Zhang et al.[23]SageFormer is a Series-Aware Framework designed for long-term multivariate time series forecasting. This model incorporates series-aware attention mechanisms that specifically account for the unique characteristics of each time series in the dataset. By tailoring the attention mechanism to the properties of individual series, SageFormer achieves more accurate and reliable long-term forecasts.

InParformer Developed by Cao et al.[24], InParformer (Interactive Parallel Attention Transformer) introduces evolutionary decomposition with interactive parallel attention for long-term time series forecasting. This model decomposes the input time series into multiple components and applies parallel attention mechanisms to each component. The interactive nature of the attention mechanisms allows for better integration of information across components, leading to improved forecasting performance.

Stecformer Proposed by Sun et al.[25] Stecformer (Spatio-temporal Encoding Cascaded Transformer) is designed for multivariate long-term time series forecasting. This model employs spatio-temporal encoding to capture both spatial and temporal dependencies in the data. The cascaded architecture allows for hierarchical processing of information, enhancing the model's ability to handle complex multivariate time series and produce accurate long-term forecasts.

SAMformer Designed by Ilbert et al.[26], SAMformer (Sharpness-Aware Minimization and Channel-Wise Attention Transformer) aims to unlock the potential of Transformers in time series forecasting by incorporating sharpness-aware minimization and channel-wise attention mechanisms. Sharpness-aware minimization helps in optimizing the model's parameters for better generalization, while channel-wise attention focuses on capturing the importance of different channels in the data. This combination results in more robust and accurate forecasts.

2 Transformers, Informers and Autoformers.

Of the Transformer variants that have been previously exposed, three pivotal models have been selected for this thesis: the Original Transformer, the Informer, and the Autoformer. These models have been chosen due to their foundational impact and the significant advancements they introduced in the field of time series forecasting.

Analyzing these three models is crucial because the Original Transformer laid the groundwork for all subsequent Transformer models. Understanding its architecture and innovations provides essential insights into the core principles that underpin all Transformer-based models.

Moreover, each of these models introduced significant advancements that addressed critical challenges in time series forecasting. The Informer improved efficiency and scalability, while the Autoformer brought a novel approach to decomposing time series data. Studying these innovations helps in understanding how the field has evolved to tackle specific issues.

Furthermore, these three models have served as the basis for many other variants. By analyzing them, one can trace the evolution of ideas and techniques that have shaped the development of more specialized and advanced models. This historical perspective is essential for appreciating the progress and future directions in the field.

Additionally, focusing on these models allows for a comprehensive understanding of the essence of time series Transformers. They encapsulate the major breakthroughs and provide a clear picture of how Transformer models have been adapted and optimized for time series forecasting.

Finally, these models are not only theoretically significant but also practically relevant. They have been widely adopted and tested in various applications, making them valuable case studies for real-world time series forecasting challenges. By focusing on the Original Transformer, the Informer, and the Autoformer, this thesis aims to build upon their strengths and explore their applications in time series forecasting. This approach ensures a deep and nuanced understanding of the fundamental principles and innovations that drive the effectiveness of Transformer models in this domain.

In this chapter, we are going to first explain what the Transformer architecture is in general. Following that, we will delve into the specifics of the time series-oriented Transformer, then the Informer, and finally the Autoformer. This structured approach will provide a clear and comprehensive understanding of each model's architecture and innovations, highlighting their contributions to the field of time series forecasting.

2.1 Transformer architecture

The Transformer architecture, introduced by Vaswani et al. in 2017, has revolutionized the field of natural language processing (NLP). Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), the Transformer relies entirely on self-attention mechanisms, eliminating the need for sequential data processing. This chapter delves into the intricacies of the Transformer architecture, highlighting its components, mechanisms, and advantages.

This architecture is composed of an encoder-decoder structure, both of which are built from layers of self-attention and feed-forward neural networks. This architecture allows for efficient parallelization and better handling of long-range dependencies in sequences, making it particularly suitable for tasks such as translation, text summarization, and question answering.

The Transformer model is built on a sophisticated encoder-decoder architecture, which is crucial for tasks such as machine translation and text summarization. This architecture consists of a stack of encoders and decoders that work together to transform an input sequence into an output sequence.

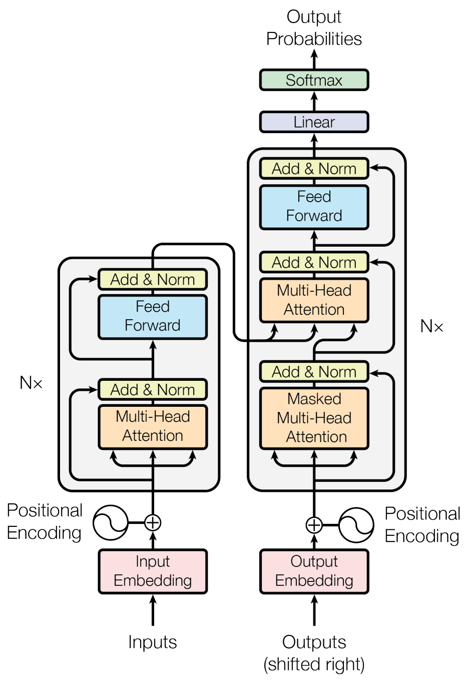

Figure 2.1 The Transformer - model architecture.

Encoder-Decoder Structure

Encoder

Each encoder layer in the Transformer model is composed of two primary sub-layers. The first sub-layer is the multi-head self-attention mechanism. This mechanism allows the model to focus on different parts of the input sequence simultaneously, capturing various aspects of the input data. By using multiple attention heads, the model can learn to attend to different positions in the sequence, which helps in understanding the context and relationships between different elements in a sequence more effectively.

The second sub-layer is the fully connected feed-forward network. This network is applied to each position independently and identically, consisting of two linear transformations with a ReLU activation in between. This sub-layer helps in further processing the information extracted by the self-attention mechanism, adding non-linearity and enabling the model to learn more complex patterns.

Both sub-layers are followed by layer normalization and residual connections. Layer normalization helps in stabilizing the training process by normalizing the inputs to each sub-layer, while residual connections allow the model to retain information from previous layers, facilitating better gradient flow and improving the overall training efficiency.

Decoder

The decoder in the Transformer model also consists of multiple layers, each containing three main sub-layers. The first sub-layer is the masked multi-head self-attention mechanism. Unlike the encoder's self-attention, this mechanism is masked to prevent the decoder from attending to future positions in the sequence, ensuring that the prediction for a particular position only depends on the known outputs up to that position.

The second sub-layer is the encoder-decoder attention mechanism. This mechanism allows the decoder to focus on relevant parts of the input sequence by attending to the encoder's output. It helps the decoder to incorporate information from the entire input sequence, which is crucial for generating accurate and contextually appropriate outputs.

The third sub-layer is the same fully connected feed-forward network used in the encoder. This sub-layer further processes the combined information from the self-attention and encoder-decoder attention mechanisms.

Similar to the encoder, each of these sub-layers in the decoder is followed by layer normalization and residual connections. These components ensure stable training and efficient information flow throughout the network.

Self-Attention Mechanism

At the core of the Transformer architecture is the self-attention mechanism, which enables the model to weigh the importance of different elements in an input sequence dynamically. The self-attention mechanism computes a set of attention scores for each word, determining how much focus each word should have on every other word in the sequence.

Scaled Dot-Product Attention
The self-attention mechanism operates through a process called scaled dot-product attention, which can be broken down into several key steps:

1. **Calculate Query (Q), Key (K), and Value (V) Vectors**: For each word in the input sequence, the model generates three vectors: a query vector (Q), a key vector (K), and a value vector (V). These vectors are derived from the word embeddings using learned linear transformations.
2. **Compute Attention Scores**: The attention score for each word pair is computed by taking the dot product of the query vector of the current word with the key vector of another word. This measures the relevance of one word to another.
3. **Scale the Attention Scores**: To prevent the dot product values from becoming too large and destabilizing the learning process, the attention scores are scaled by dividing by the square root of the dimensionality of the key vectors ($\sqrt{d_k}$).
4. **Apply Softmax Function**: The scaled attention scores are then passed through a softmax function to convert them into probabilities, ensuring that the scores for each word sum to 1. These probabilities are known as attention weights.
5. **Weighted Sum of Value Vectors**: The final attention output for each word is obtained by taking a weighted sum of the value vectors, using the attention weights as coefficients.

The entire process is summarized by the following formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

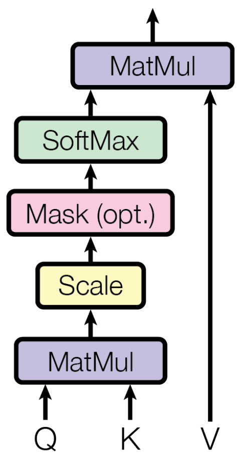

Figure 2.2 Scaled Dot-Product Attention.

Multi-Head Attention
Instead of performing a single attention function, the Transformer employs multiple attention heads. Each head performs its own attention operation in parallel, allowing the model to capture different aspects of

relationships between words. The outputs of all heads are then concatenated and linearly transformed to produce the final attention output.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O$$

where each head is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

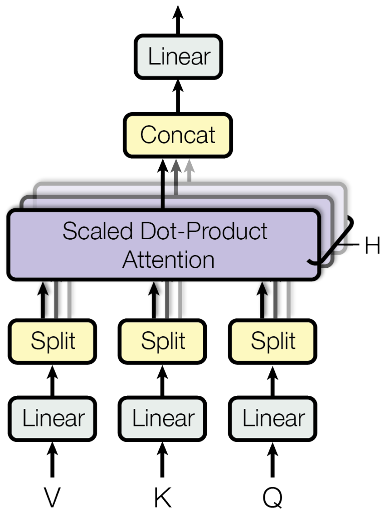

Figure 2.3 Scaled Dot-Product Attention.

2.2 Time Series Transformer

Previously, we explored the Transformer architecture. Now, we turn our focus to the Time Series Transformer, designed specifically for time series forecasting. As well as the original Transformer, this model is built around two main components: the Encoder and the Decoder. The Encoder processes past time series values, capturing essential patterns and dependencies. Meanwhile, the Decoder uses this encoded information to predict future values, integrating temporal features to enhance prediction accuracy.

Time series forecasting involves predicting future values based on historical data. The Time Series Transformer tackles this through embeddings, which convert categorical data into continuous vectors, capturing temporal dependencies effectively. These embeddings include channel projection and timestamp embeddings, which help the model understand both local and global temporal dynamics.

Training the model involves a method called teacher-forcing, where the model learns correct sequence patterns by using actual previous values during training. Additionally, the Time Series Transformer employs probabilistic forecasting to quantify uncertainty in predictions, which is crucial for real-world decision-making.

In this section, we will dive into the specifics of the Encoder and Decoder, the role of embeddings, and the training methodologies that make the Time Series Transformer a powerful tool for accurate and reliable time series forecasting.

2.2.1 Time Series Forecasting (TSF) Problem Formulation

Time series forecasting (TSF) involves predicting future values based on previously observed data. For a time series containing C variates, the historical data can be represented as:

$$X = \{X_1^t, \ldots, X_C^t\}_{t=1}^L$$

where L is the look-back window size and X_i^t is the value of the i-th variate at the t-th time step. The goal of TSF is to predict the future values:

$$\hat{X} = \{\hat{X}_1^t, \ldots, \hat{X}_C^t\}_{t=L+1}^{L+T}$$

for the next T time steps.

When $T > 1$, there are two main approaches to multi-step forecasting:

- **Iterated Multi-Step (IMS) Forecasting:** This approach learns a single-step forecaster and iteratively applies it to obtain multi-step predictions. While IMS predictions tend to have smaller variance due to the autoregressive estimation procedure, they are prone to error accumulation effects. Therefore, IMS forecasting is preferable when there is a highly accurate single-step forecaster and T is relatively small.

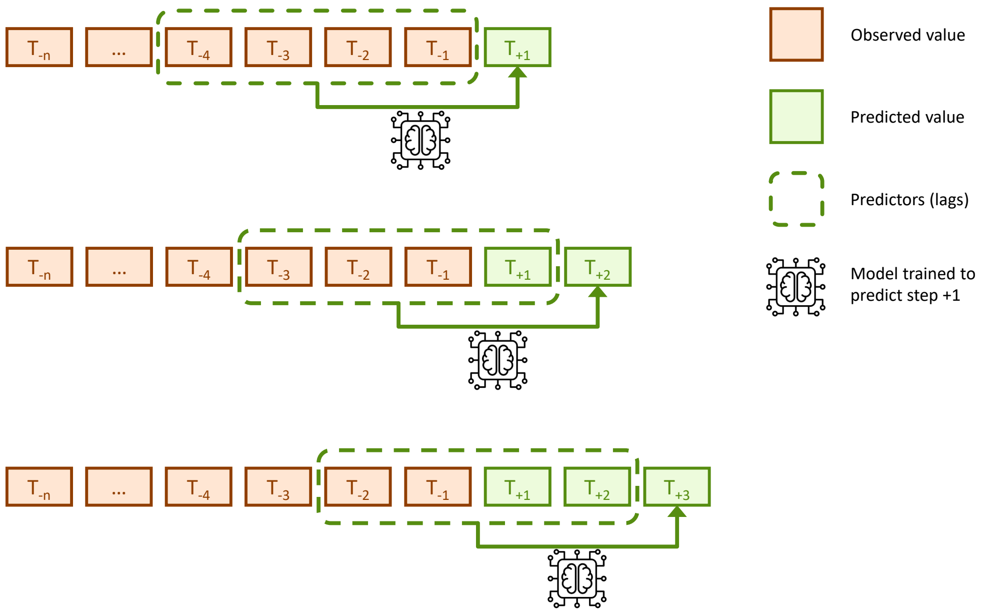

Figure 2.4 Illustration of Iterated Multi-Step (IMS) Forecasting.

- **Direct Multi-Step (DMS) Forecasting:** This approach directly optimizes the multi-step forecasting objective at once. DMS forecasting generates more accurate predictions when it is challenging to obtain an unbiased single-step forecasting model or when T is large.

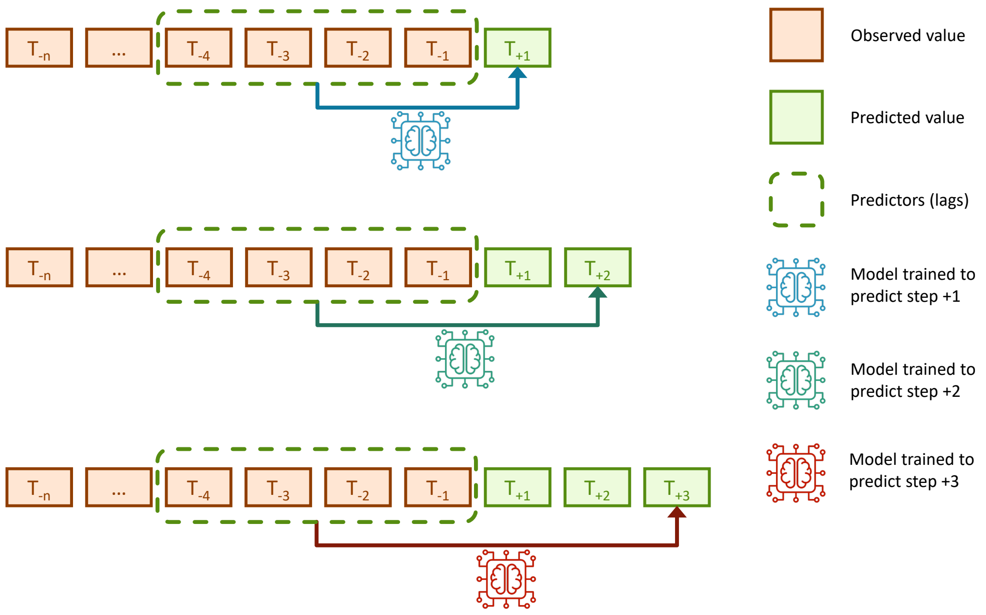
T-n
...
T-4
T-3
T-2
T-1
T+1
T+2
T+3
Observed value
Predicted value
Predictors (lags)
Model trained to predict step +1
Model trained to predict step +2
Model trained to predict step +3

Figure 2.5 Illustration of Direct Multi-Step (DMS) Forecasting.

2.2.2 Embedding

Embeddings are a crucial component in the field of machine learning and natural language processing. They are used to convert categorical data into continuous vector representations, which can then be fed into machine learning models. In the context of time series forecasting, embeddings help in capturing the temporal dependencies and patterns within the data, enabling the model to make accurate predictions.

To adapt the Transformer architecture for time series forecasting, the embedding process involves several key components: channel projection, fixed position, local timestamp, and global timestamp. Each of these components plays a vital role in ensuring that the model effectively captures the temporal dynamics of the data.

Channel Projection

Channel projection is the process of transforming the input time series data into a higher-dimensional space. This is done to capture the complex relationships between different channels (or features) of the time series data. By projecting the data into a higher-dimensional space, the model can better understand the interactions between different channels and make more accurate predictions.

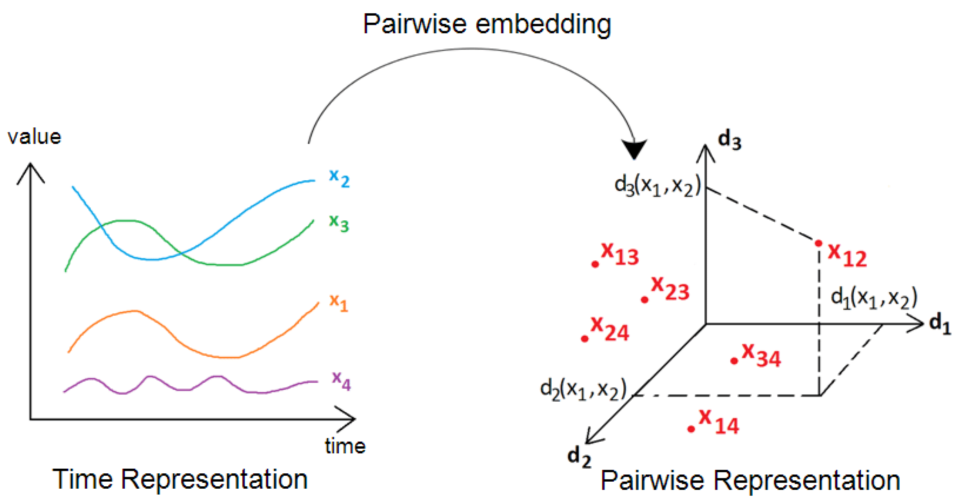

Figure 2.6 Example of embedding of time series xi from the temporal space (left) into the pairwise space (right). In this example, a pair of time series (x1, x2) is projected into the pairwise space as a vector x12 described by p = 3 basic metrics: x12 = [d1(x1, x2), d2(x1, x2), d3(x1, x2)] T, from "Multiple Metric Learning for large margin kNN Classification of time series" [27].

Fixed Position Embedding

As we explained before, fixed position embedding is used to encode the positional information of the time series data. In the Transformer architecture, positional encodings are added to the input embeddings to provide the model with information about the order of the data points. This is crucial for time series forecasting, as the temporal order of the data points is essential for making accurate predictions. Fixed position embeddings are typically generated using sinusoidal functions, which provide a unique encoding for each position in the sequence.

Local Timestamp Embedding

Local timestamp embedding captures the local temporal information within the time series data. This includes information such as the time of day, day of the week, or any other relevant local temporal features. By incorporating local timestamp embeddings, the model can better understand the short-term patterns and trends within the data, leading to more accurate forecasts.

For example, consider a data point recorded at 3 PM on a Wednesday. The local timestamp embedding would capture the hour of the day (15), the day of the week (3 for Wednesday), and the day of the month (15).

Global Timestamp Embedding

Global timestamp embedding captures the global temporal information across the entire time series. This includes long-term trends and seasonal patterns that may not be immediately apparent from the local temporal information. By incorporating global timestamp embeddings, the model can better understand the overall temporal dynamics of the data, leading to more accurate long-term forecasts.

For example, imagine a data point recorded in July 2024. The global timestamp embedding would include the month of the year (7 for July), the quarter of the year (3 for July to September), and the year (2024).

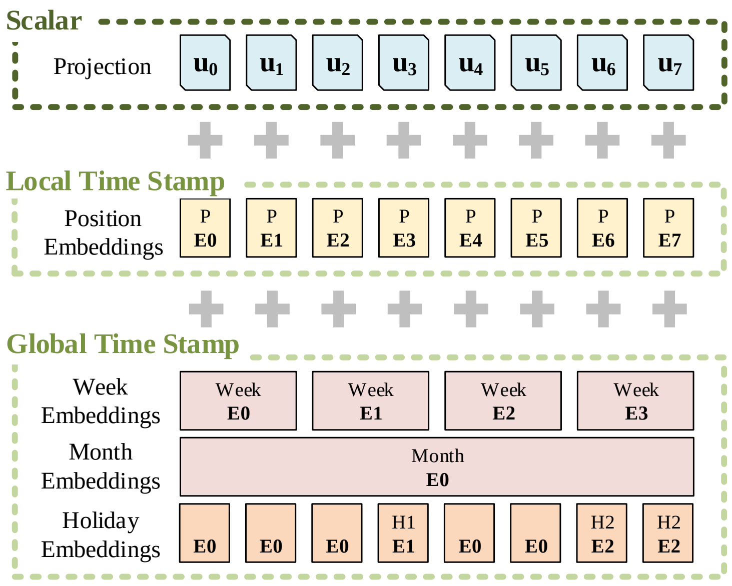

Figure 2.7 The input representation of a Time Series Transformer, from Informer (Zhou, et al., 2019)[13].

2.2.3 Encoder

The Encoder's role is to process a sequence of past time series values, known as *past_values*. It effectively captures the temporal patterns and dependencies within these values. In addition to the raw *past_values*, the Encoder takes the data from the embedding, which includes local timestamps and other relevant temporal features, to provide the model with more temporal context.

2.2.4 Decoder

The Decoder's role is to predict future time series values, known as *future_values*, based on the encoded information from *past_values*. The Decoder generates a sequence of predictions for the desired forecast horizon. Like the Encoder, the Decoder can incorporate future time features to provide temporal context for the predictions. These features are analogous to the past time features.

2.2.5 Training Methodology

Training a Time Series Transformer involves a process called *teacher-forcing*. This method is crucial for effectively training sequence-to-sequence models, such as those used in time series forecasting.

Teacher-forcing is a technique where the model is trained to predict the next value in a sequence by using the actual previous value rather than the model's own previous prediction. This helps the model learn the correct sequence patterns more efficiently and reduces the accumulation of errors during training.

Here's a step-by-step breakdown of how we train the Time Series Transformer:

1. **Preparing the Training Data:**
   - We start with pairs of *past_values* and *future_values*.
   - *past_values* represent the known historical data up to the current time step.
   - *future_values* represent the target values we aim to predict.
2. **Shifting Future Values:**
   - The *future_values* sequence is shifted one position to the right.

- This creates a new sequence where each time step corresponds to the next time step in the original *future_values*.

3. **Initial Input for the Decoder:**
   - The last value of *past_values* is used as the initial input for the Decoder.
   - This value acts as the starting point for generating the first prediction in the *future_values* sequence.

2.2.6 Probabilistic Forecasting

Unlike classical point forecasting methods that output a single value per time step, the Time Series Transformer is designed for probabilistic forecasting. This approach models a distribution from which predictions can be sampled, providing a measure of uncertainty in the forecasts. This is particularly useful in real-world decision-making processes where understanding the range of possible outcomes is crucial.

Deep learning models, including Transformers, excel in this context by learning from multiple related time series and modeling data uncertainty effectively. Probabilistic forecasting can be implemented by learning future parameters of a parametric distribution or using techniques like conformal prediction adapted for time series.

According to Adrian E. Raftery, probabilistic forecasts are becoming increasingly available and are essential for various types of users, including general assessors, change assessors, risk avoiders, and decision theorists. These forecasts provide valuable insights by quantifying the uncertainty and offering a range of possible outcomes, which can be summarized using probabilities of adverse events and percentiles of the predictive distribution. Effective communication of probabilistic forecasts involves interacting with users to understand their goals and minimizing cognitive load by presenting the information in a clear and concise manner. This ensures that the forecasts are not only accurate but also trusted and actionable in practical applications.

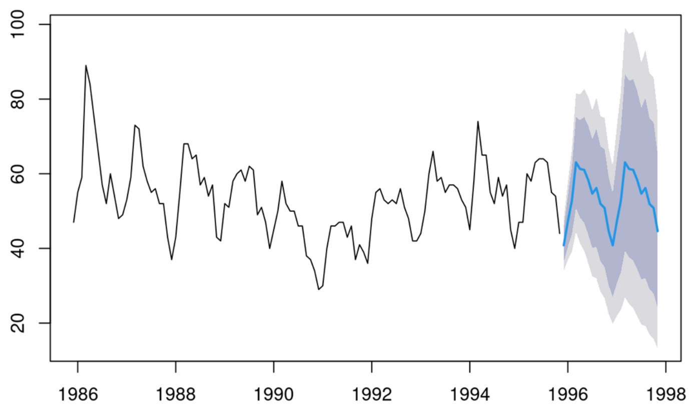

Figure 2.8 Illustration of Probabilistic Forecasting.

2.3 Informer

In December 2020, Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang introduced a groundbreaking paper titled *"Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting"*. This paper received the prestigious AAAI-21 Best Paper Award, recognizing its significant contributions to the field.

Next, we will delve into the core components of the Informer model. We will explore the ProbSparse Attention mechanism, which enhances efficiency and performance, and the Self-Attention Distilling technique, which optimizes the handling of long sequences. By understanding these innovations, we can appreciate how the Informer model significantly advances the capabilities of time-series forecasting.

2.3.1 ProbSparse Attention

The core concept of ProbSparse attention is based on the observation that canonical self-attention scores follow a long-tail distribution. In this distribution, "active" queries are located in the "head" scores, while "lazy" queries are found in the "tail" area. An "active" query, denoted as q_i, is one where the dot-product $\langle q_i, k_i \rangle$

significantly contributes to the attention mechanism. Conversely, a "lazy" query generates a dot-product that results in trivial attention. Here, q_i and k_i represent the i-th rows in the Q and K attention matrices, respectively.

Given the distinction between "active" and "lazy" queries, ProbSparse attention focuses on selecting the "active" queries to form a reduced query matrix Q_{reduced}. This matrix is then used to compute the attention weights with a computational complexity of $O(T \log T)$.

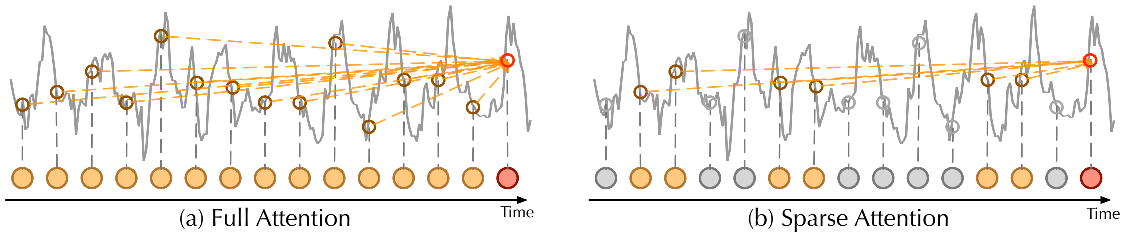

Figure 2.9 Vanilla self attention vs ProbSparse attention, from Autoformer (Wu, Haixu, et al., 2021)[16].

To illustrate this, let's revisit the canonical self-attention formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where $Q \in \mathbb{R}^{L_Q \times d}$, $K \in \mathbb{R}^{L_K \times d}$, and $V \in \mathbb{R}^{L_V \times d}$. In practice, the input lengths of queries and keys are typically equivalent in self-attention computation, i.e., $L_Q = L_K = T$, where T is the time series length. Consequently, the QK^T multiplication has a computational complexity of $O(T^2 \cdot d)$.

In ProbSparse attention, the goal is to create a new Q_{reduced} matrix and define:

$$\text{ProbSparseAttention}(Q,K,V) = \text{softmax}\left(\frac{Q_{\text{reduced}}K^T}{\sqrt{d_k}}\right)V$$

The Q_{reduced} matrix selects only the top u "active" queries, where $u = c \cdot \log L_Q$ and c is the sampling factor hyperparameter for ProbSparse attention. Since Q_{reduced} selects only the top u queries, its size is $c \cdot \log L_Q \times d$. Therefore, the multiplication $Q_{\text{reduced}}K^T$ has a computational complexity of $O(L_K \log L_Q) = O(T \log T)$.

The next step is to determine how to select the top u "active" queries to form Q_{reduced}. This involves defining the Query Sparsity Measurement.

Query Sparsity Measurement

The Query Sparsity Measurement $M(q_i, K)$ is used to identify the u "active" queries q_i within Q to construct Q_{reduced}. The fundamental idea is that the dominant $\langle q_i, k_i \rangle$ pairs cause the "active" q_i's probability distribution to deviate from a uniform distribution. This deviation can be quantified using the KL divergence between the actual query distribution and the uniform distribution.

In practical terms, the measurement is defined as:

$$M(q_i,K) = \max_j \left(\frac{q_i k_j^T}{\sqrt{d}}\right) - \frac{1}{L_k}\sum_{j=1}^{L_k}\left(\frac{q_i k_j^T}{\sqrt{d}}\right)$$

The key insight here is that a larger $M(q_i, K)$ value indicates that the query q_i should be included in Q_{reduced}, while a smaller value suggests otherwise.

To efficiently compute this measurement, it is important to note that most dot-products $\langle q_i, k_i \rangle$ generate trivial attention due to the long-tail distribution property. Therefore, it is sufficient to consider a representative subset of keys from K rather than the entire set.

2.3.2 Distilling

The Informer model employs a ProbSparse self-attention mechanism, which introduces some redundancy in the encoder's feature map. To address this, a distilling operation is used to reduce the input size between

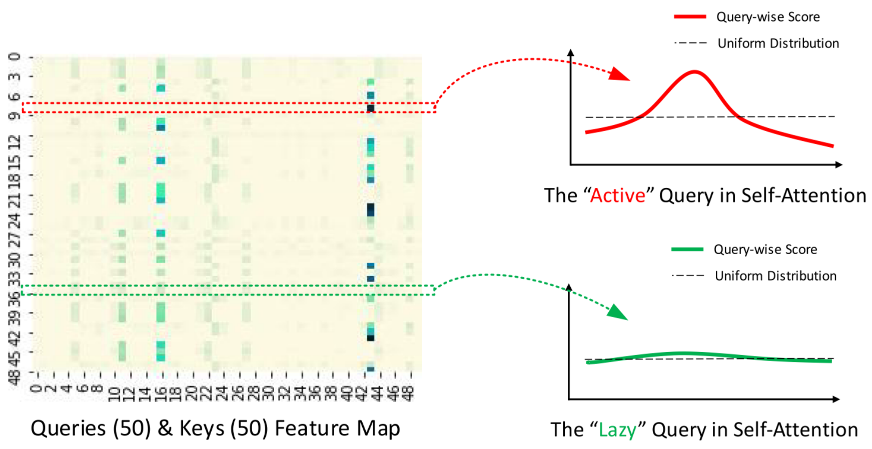
Query-wise Score
Uniform Distribution
The "Active" Query in Self-Attention
Query-wise Score
Uniform Distribution
The "Lazy" Query in Self-Attention
Queries (50) & Keys (50) Feature Map

Figure 2.10 The illustration of ProbSparse Attention, from Informer (Zhou, et al., 2019)[13].

encoder layers by half, effectively removing this redundancy. In practice, the distilling operation in Informer involves adding 1D convolution layers followed by max pooling between each of the encoder layers.

Let X_n be the output of the n-th encoder layer. The distilling operation is then defined as:

$$X_{n+1} = \mathrm{MaxPool}(\mathrm{ELU}(\mathrm{Conv1d}(X_n)))$$

By applying this operation, the input size of each subsequent layer is reduced by half. This reduction leads to a more efficient memory usage. Specifically, the memory usage is reduced from $O(N \cdot T^2)$ to $O(N \cdot T \log T)$, where N is the number of encoder/decoder layers and T is the sequence length. This optimization is crucial for handling long sequence time-series forecasting tasks efficiently.

2.4 Autoformer

Autoformer is a significant advancement in the realm of time series forecasting, building on the transformer architecture by integrating time series decomposition and a novel auto-correlation mechanism. In the following, we will explain the main contributions of Autoformer, focusing on the Decomposition Layer and the Attention (Autocorrelation) Mechanism. These innovations allow Autoformer to capture and leverage period-based dependencies, enhancing its performance over traditional transformer models.

2.4.1 Decomposition Layer

Concept of Decomposition in Time Series

Time series decomposition is a method of breaking down a time series into three systematic components: trend-cycle, seasonal variation, and random fluctuations. The trend component reflects the long-term direction of the series, which can be increasing, decreasing, or stable. The seasonal component captures recurring patterns within the series, such as yearly or quarterly cycles. The random component represents the noise that cannot be explained by the trend or seasonal components.

Decomposition can be either additive, where the components are summed, or multiplicative, where the components are multiplied. This technique, commonly implemented in libraries like *statsmodels*, helps in understanding and modeling the underlying patterns in the data.

Decomposition in Autoformer

Autoformer incorporates a decomposition block within its architecture. This block enables the model to progressively aggregate the trend-cyclical part and extract the seasonal part from the series. The encoder and decoder of Autoformer use this decomposition block, significantly enhancing the model's ability to capture and utilize these components.

Formally, for an input series $X \in \mathbb{R}^{L\times d}$ with length L, the decomposition layer returns X_{trend} and X_{seasonal} defined as:

$$X_{\text{trend}} = \text{AvgPool}(\text{Padding}(X))$$

$$X_{\text{seasonal}} = X - X_{\text{trend}}$$

This decomposition layer allows Autoformer to explicitly model the trend and seasonal components, which has proven beneficial in various time series forecasting applications.

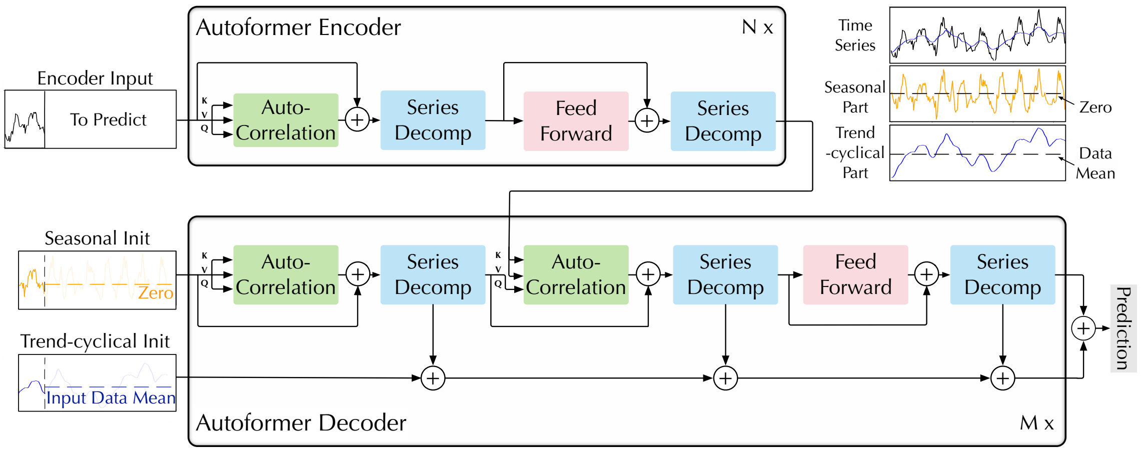

Figure 2.11 Autoformer architecture. The encoder eliminates the long-term trend-cyclical part by series decomposition blocks (blue blocks) and focuses on seasonal patterns modeling. The decoder accumulates the trend part extracted from hidden variables progressively. The past seasonal information from encoder is utilized by the encoder-decoder Auto-Correlation (center green block in decoder), from Autoformer (Wu, Haixu, et al., 2021)[16].

2.4.2 Attention (Autocorrelation) Mechanism

Autoformer introduces an innovative approach to the attention mechanism used in traditional transformers. Instead of the conventional self-attention, which calculates attention weights directly in the time domain, Autoformer utilizes an autocorrelation mechanism that operates in the frequency domain using the Fast Fourier Transform (FFT). This method helps the model better capture periodic dependencies in the data, thereby enhancing its forecasting performance.

Understanding Autocorrelation

Autocorrelation measures how a time series correlates with a lagged version of itself. For a given time lag τ, autocorrelation quantifies the relationship (often measured by Pearson correlation) between the current value of the series at time t and its past value at time $t-\tau$. Mathematically, it is expressed as:

$$\text{Autocorrelation}(\tau) = \text{Corr}(y_t, y_{t-\tau})$$

In the context of Autoformer, this concept replaces the traditional dot-product attention. Instead of directly comparing queries (Q) and keys (K) through dot-products, Autoformer uses autocorrelation to capture dependencies over different time lags.

Implementing Autocorrelation with FFT

The autocorrelation mechanism computes the attention weights using FFT. This approach allows all lags' autocorrelations to be calculated simultaneously, making the process efficient with a time complexity of $O(L\log L)$, where L is the length of the input time series. Here's how it works in practice:

1. **FFT Transformation:** Convert the queries (Q) and keys (K) to the frequency domain using FFT.
2. **Frequency Domain Multiplication:** Multiply the transformed Q and K element-wise.
3. **Inverse FFT:** Apply the inverse FFT to transform the results back to the time domain, yielding the autocorrelations.

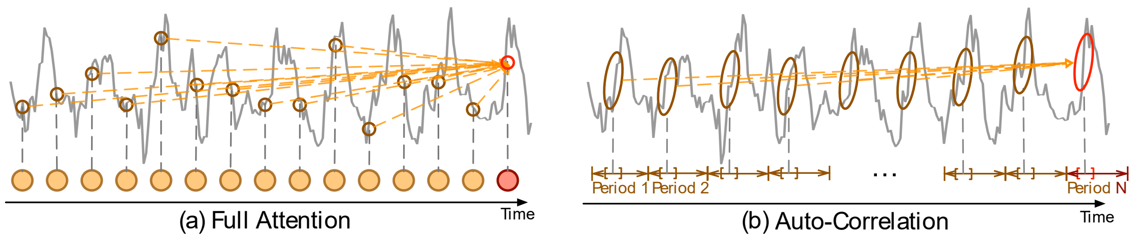

Figure 2.12 Vanilla self attention vs Autocorrelation mechanism, from Autoformer (Wu, Haixu, et al., 2021)[16].

This function computes the autocorrelation by performing FFT on the queries and keys, multiplying them in the frequency domain, and then applying the inverse FFT to obtain the autocorrelations.

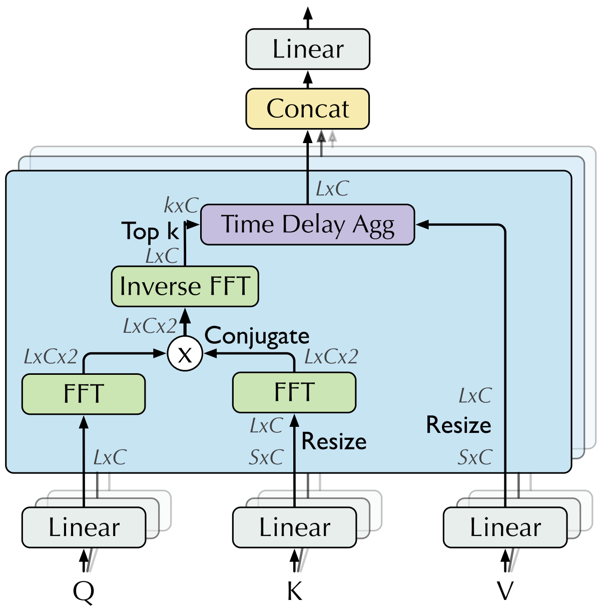

Figure 2.13 Attention weights computation in frequency domain using FFT, from Autoformer (Wu, Haixu, et al., 2021)[16].

Time Delay Aggregation

Once the autocorrelation weights (referred to as *attn_weights*) are computed, they need to be aggregated with the values (V) to produce the final attention output. This process, known as Time Delay Aggregation, involves the following steps:

1. **Rolling:** For each time delay τ, align the values (V) by shifting them by τ. This process is called rolling.
2. **Element-wise Multiplication:** Multiply the rolled values by the corresponding autocorrelation weights.
3. **Summation:** Sum the weighted values to get the final attention output.

The aggregation process can be formalized with the following equations:

1. Identify the top-k lags with the highest autocorrelation values:

$$\tau_1, \tau_2, \ldots, \tau_k = \arg \text{Top-k}(R_{Q,K}(\tau))$$

2. Apply softmax to normalize the autocorrelation values:

$$\hat{R}_{Q,K}(\tau_1), \hat{R}_{Q,K}(\tau_2), \ldots, \hat{R}_{Q,K}(\tau_k) = \text{Softmax}(R_{Q,K}(\tau_1), R_{Q,K}(\tau_2), \ldots, R_{Q,K}(\tau_k))$$

3. Compute the final attention output by summing the element-wise products of the rolled values and the normalized autocorrelation weights:

$$\text{Autocorrelation-Attention} = \sum_{i=1}^{k} \text{Roll}(V, \tau_i) \cdot \hat{R}_{Q,K}(\tau_i)$$

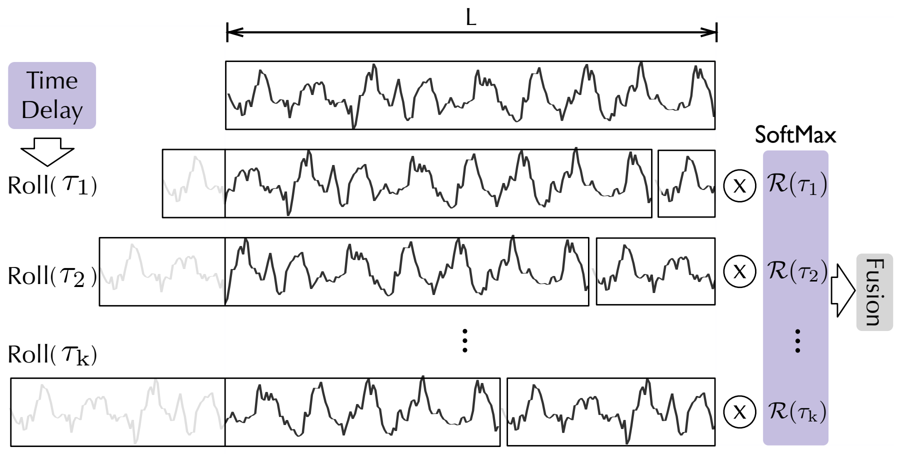

Figure 2.14 Aggregation by time delay, from Autoformer (Wu, Haixu, et al., 2021)[16].

List of Figures

List of Tables

List of Codes

Bibliography

[1] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[2] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2015. [Online]. Available: https://arxiv.org/abs/1411.4038

[3] A. Dempster, F. Petitjean, and G. I. Webb, “Rocket: exceptionally fast and accurate time series classification using random convolutional kernels,” *Data Mining and Knowledge Discovery*, vol. 34, no. 5, p. 1454–1495, Jul. 2020. [Online]. Available: http://dx.doi.org/10.1007/s10618-020-00701-z

[4] K. Bringmann, N. Fischer, I. van der Hoog, E. Kipouridis, T. Kociumaka, and E. Rotenberg, “Dynamic dynamic time warping,” 2023. [Online]. Available: https://arxiv.org/abs/2310.18128

[5] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. ACM, Aug. 2016. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939785

[6] S. J. Taylor and B. Letham, “Prophet: Forecasting at scale,” https://facebook.github.io/prophet/, 2017, accedido: 14 de julio de 2024.

[7] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, “Ts-chief: a scalable and accurate forest algorithm for time series classification,” *Data Mining and Knowledge Discovery*, vol. 34, no. 3, p. 742–775, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1007/s10618-020-00679-8

[8] J. Lines, S. Taylor, and A. Bagnall, “Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 1041–1046.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need*, 2023.

[10] A. Zeng, M. Chen, L. Zhang, and Q. Xu, “Are transformers effective for time series forecasting?” 2022. [Online]. Available: https://arxiv.org/abs/2205.13504

[11] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, “Transformers in time series: A survey,” 2023. [Online]. Available: https://arxiv.org/abs/2202.07125

[12] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” 2020. [Online]. Available: https://arxiv.org/abs/1912.09363

[13] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, “Informer: Beyond efficient transformer for long sequence time-series forecasting,” 2021. [Online]. Available: https://arxiv.org/abs/2012.07436

[14] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting,” 2020. [Online]. Available: https://arxiv.org/abs/1907.00235

[15] N. Kitaev, Łukasz Kaiser, and A. Levskaya, “Reformer: The efficient transformer,” 2020. [Online]. Available: https://arxiv.org/abs/2001.04451

[16] H. Wu, J. Xu, J. Wang, and M. Long, “Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting,” 2022. [Online]. Available: https://arxiv.org/abs/2106.13008

[17] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting,” 2022. [Online]. Available: https://arxiv.org/abs/2201.12740

[18] S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. X. Liu, and S. Dustdar, “Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting,” in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=0EXmFzUn5I

[19] G. Woo, C. Liu, D. Sahoo, A. Kumar, and S. Hoi, “Etsformer: Exponential smoothing transformers for time-series forecasting,” 2022. [Online]. Available: https://arxiv.org/abs/2202.01381

[20] W. Wang, L. Yao, L. Chen, B. Lin, D. Cai, X. He, and W. Liu, “Crossformer: A versatile vision transformer hinging on cross-scale attention,” 2021. [Online]. Available: https://arxiv.org/abs/2108.00154

[21] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, “A time series is worth 64 words: Long-term forecasting with transformers,” 2023. [Online]. Available: https://arxiv.org/abs/2211.14730

[22] W. Xue, T. Zhou, Q. Wen, J. Gao, B. Ding, and R. Jin, “Card: Channel aligned robust blend transformer for time series forecasting,” 2024. [Online]. Available: https://arxiv.org/abs/2305.12095

[23] Z. Zhang, L. Meng, and Y. Gu, “Sageformer: Series-aware framework for long-term multivariate time series forecasting,” 2023. [Online]. Available: https://arxiv.org/abs/2307.01616

[24] H. Cao, Z. Huang, T. Yao, J. Wang, H. He, and Y. Wang, “Inparformer: Evolutionary decomposition transformers with interactive parallel attention for long-term time series forecasting,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 6, pp. 6906–6915, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/25845

[25] Z. Sun, Y. Wei, W. Jia, and L. Yu, “Stecformer: Spatio-temporal encoding cascaded transformer for multivariate long-term time series forecasting,” 2023. [Online]. Available: https://arxiv.org/abs/2305.16370

[26] R. Ilbert, A. Odonnat, V. Feofanov, A. Virmaux, G. Paolo, T. Palpanas, and I. Redko, “Samformer: Unlocking the potential of transformers in time series forecasting with sharpness-aware minimization and channel-wise attention,” 2024. [Online]. Available: https://arxiv.org/abs/2402.10198

[27] C. T. Do, A. Douzal, S. Marie, and M. Rombaut, “Multiple metric learning for large margin knn classification of time series,” 09 2015.