

# EXPLORATION OF SELF-ADAPTIVE LEARNING AND FORECASTING FOR TIME SERIES

ALVARO ARMADA RUIZ

**Thesis supervisor:** ARGIMIRO ARRATIA QUESADA (Department of Computer Science)

**Degree:** Master Degree in Artificial Intelligence

Thesis report

School of Engineering
Universitat Rovira i Virgili (URV)

Faculty of Mathematics
Universitat de Barcelona (UB)

Barcelona School of Informatics (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

# Abstract

Time series forecasting poses several inherent challenges. Typically, the targets of interest are non-stationary random variables characterized by both short-term and long-term autocorrelations. Controlling for the variation of data distributions over time is a must, as so it is the exploitation of the different levels of data dependency. This thesis tackles these challenges by integrating methodologies from three seminal papers—Self-Adaptive Forecasting (SAF), adaRNN, and transformers—into a unified model, the AdaTransformer. By leveraging the adaptive capabilities of adaRNN and SAF and incorporating transformer architecture, the AdaTransformer captures complex temporal dependencies with improved accuracy. Additionally, the AdaTransformer introduces an enhanced Temporal Dependency Characterization (TDC) segmentation algorithm, building upon prior work to improve segmentation quality. Through comprehensive evaluation across diverse datasets encompassing air quality indices, electric power consumption, and financial markets represented by EUR/USD and TSLA datasets, our results highlight the AdaTransformer's consistent outperformance of the adaRNN model alone, as evidenced by lower Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). This research underscores the effectiveness of integrating transformer-based models with adaptive techniques, offering a robust solution for accurate and reliable time series forecasting.

# Acknowledgements

I wish to extend my sincere gratitude to my supervisor, for the valuable advice and consistent support provided throughout this research project. Your guidance was essential for the completion of my thesis.

I must also acknowledge the support of my family, who have provided a stable base throughout my studies. Their unwavering support has been a significant factor in my pursuit of higher education.

Additionally, I thank my friends for their understanding and moral support during the challenging phases of this work. Your reassurance was invaluable.

I am grateful to each individual who contributed directly or indirectly to my academic journey and this research project.

# Contents

# List of Figures

# List of Tables

## Chapter 1

# Introduction

Time series forecasting is a critical task that influences many areas of our lives. Whether it's predicting stock market movements, forecasting weather patterns, or estimating electricity demand, we rely on models that analyze past data to make educated guesses about the future.

However, our dynamic world presents a challenge. Traditional forecasting models, which often base predictions on historical patterns, can struggle when faced with unexpected changes. These shifts can arise from various factors, including new technological developments, economic changes, or unforeseen global events.

Given the importance of accurate forecasting in decision-making, there's a pressing need for models that can adapt to the present and anticipate the future more effectively. In this research, we aim to address this need by exploring and integrating three promising approaches: AdaRNN (Adaptive Recurrent Neural Networks) [9], SAF (Self-Adaptive Forecasting) [1], and Transformers [28]. Each of these methods offers unique advantages in handling time series data, and their combined potential forms the core of our investigation. By integrating these diverse ideas, this study presents a novel model: the AdaTransformer. This model stands out for its ability to capture both short and long-term dependencies within time series data, thereby improving forecasting accuracy across various domains.

A significant contribution of this thesis is the development of a new Temporal Dependency Characterization (TDC) algorithm using dynamic programming. This algorithm surpasses the performance of the traditional greedy approach employed by adaRNN, demonstrating superior segmentation quality. This advancement in TDC methodology enhances the AdaTransformer's ability to identify and capture meaningful temporal dependencies within the data, leading to more accurate forecasting outcomes.

Furthermore, this study highlights the improved forecasting accuracy achieved by the AdaTransformer model. By leveraging the adaptive mechanisms from SAF, the adaptive segmentation concept from adaRNN, and the powerful architecture of transformers, the AdaTransformer outperforms existing models in terms of RMSE and MAE across multiple datasets. This enhancement in forecasting accuracy underscores the practical utility of the AdaTransformer model in real-world forecasting scenarios.

In summary, this thesis makes three key contributions to the field of time series forecasting: the novel integration of adaRNN, SAF, and transformers to form the AdaTransformer model, the development of an advanced TDC algorithm using dynamic programming, and the significant improvement in forecasting accuracy achieved by the AdaTransformer model. These contributions not only advance our understanding of time series forecasting, but also offer practical solutions to address the challenges posed by non-stationary data distributions.

The upcoming sections will delve deeper into the specifics of these three methods, laying the groundwork for our proposed integrative approach to adaptive time series forecasting.

Chapter 2

# Motivation

The landscape of time series forecasting is vast, with each method bringing its unique strengths and perspectives to the table. While individual methods like AdaRNN, SAF, and Transformers have shown promise in addressing specific challenges, there's a compelling case to be made for their integrative use. By combining these methods, we aim to harness their collective strengths, creating a model that's more adaptive, robust, and comprehensive.

## 2.1 AdaRNN's Adaptive Learning

AdaRNN's primary strength lies in its ability to handle changes in data over time, based on its evolving distribution. This adaptability ensures that the model remains attuned to shifts during training, making it particularly effective for non-stationary time series. However, while AdaRNN focuses on adaptation during training, it might benefit from mechanisms that allow for adaptability during testing or real-time forecasting.

## 2.2 SAF's Test-time Adaptation

This is where SAF comes into play. SAF's backcasting mechanism allows for continuous adaptation, not just during training, but also during testing. By predicting past values using current data, SAF recalibrates the model based on the most recent data distributions. This ensures that the model remains agile and responsive, especially when faced with unforeseen changes or anomalies in the data.

## 2.3 Transformers' Temporal Dynamics

While both AdaRNN and SAF offer adaptability, Transformers bring depth to the equation. Their self-attention mechanism allows them to capture intricate patterns and dependencies across different time steps in the series. This depth ensures a comprehensive understanding of both the patterns and their relevance in evolving contexts.

## 2.4 The Case for Integration

Each of the three methods addresses a facet of the challenges in time series forecasting. AdaRNN offers training-time adaptability, SAF ensures test-time agility, and Transformers provide depth. By integrating these methods, we can create a model that's equipped to handle the complexities and variabilities of real-world time series data.

Moreover, the methods complement each other's weaknesses. For instance, while Transformers can capture deep temporal relationships, they might struggle with sudden distributional shifts, a challenge that AdaRNN and SAF are designed to address. On the other hand, the complexities introduced by AdaRNN and SAF can be balanced by the depth and parallel processing capabilities of Transformers.

In essence, the integration of AdaRNN, SAF, and Transformers promises a state-of-the-art time series forecasting model that's adaptive, deep, and robust, ready to tackle the challenges of the dynamic world of time series data.

Chapter 3

# Related Work

The dynamic field of time series forecasting has seen considerable evolution, transitioning from traditional methods to sophisticated neural network models and adaptive techniques. This progression addresses the increasing complexity of data and the limitations posed by earlier approaches. In the related work outlined below, we delve into three key areas: traditional forecasting methods, the integration of advanced neural networks, and the implementation of adaptive strategies tailored for real-time adaptation.

## 3.1 Traditional time series forecasting methods

The analysis of time series data has traditionally employed a variety of methodologies, each with its own strengths and limitations. We find distance-based methods [13, 20], feature-based methods [23] and ensemble methods [3]. Distance-base methods utilize metrics such as Euclidean distance or Dynamic Time Warping (DTW) [15] to measure the similarity and distance between segments of time series data, enabling classification and regression based on pattern similarity. Feature-based methods involves extracting either manually identified or algorithmically learned features to capture the global or local patterns within the time series, which are then used for further analysis. Ensemble methods combine multiple models or classifiers, enhancing forecasting performance by leveraging the strengths of various approaches simultaneously. These traditional methods, however, often require extensive data preprocessing and feature engineering, making them less effective for handling large-scale data or complex patterns. Moreover, they generally lack the flexibility to adapt to dynamic changes in data distributions over time.

Addressing the challenge of non-stationarity in time series forecasting, various adaptive techniques have been developed to enhance accuracy under these conditions. Historically, autoregressive models have been modified to include time-varying coefficients, providing a means to adapt to changes over time [8]. Similarly, Gaussian process models have been employed with non-stationary covariance functions, offering flexibility in adapting to data changes [5]. Techniques such as spectral decomposition, often combined with recursive smoothing algorithms, have been used in adaptive state-space models to adjust forecasting models for seasonality [19]. Modifications to Support Vector Machines (SVMs), including the use of an exponentially increasing regularization constant and a decreasing tube size, have also been implemented to accommodate structural changes in data [6]. While

these methods are innovative, they often remain confined to specific model frameworks and do not integrate well with modern deep learning approaches, highlighting the need for more flexible and generalizable solutions.

## 3.2 Advances in Neural Network Models for Time Series

With the limitations of traditional methods, there has been a shift towards using advanced neural network architectures. Recurrent Neural Networks (RNNs), including Gated Recurrent Units (GRUs) and Long Short-Term Memory cells (LSTMs), have gained popularity due to their capability to automatically extract high-quality features and manage long-term dependencies. The incorporation of attention mechanisms [17, 22] and tensor factorization techniques [24] has further improved the ability of these models to capture shared information across different segments of data, enhancing the forecasting accuracy.

Among the most significant advancements in neural network architectures is the introduction of the Transformer model [28]. This model revolutionizes data processing by employing self-attention mechanisms that simultaneously assess the importance of different segments of input data. Unlike traditional models that process data sequentially, Transformers manage long-range dependencies more effectively and reduce the computational burden associated with increasing sequence length. This makes them particularly suitable for complex sequence learning tasks found in time series forecasting, where efficiency and scalability are crucial.

Additionally, approaches like seq2seq models [12, 10, 18] and the integration of state space models [10] with deep learning have been explored to model uncertainty and provide multistep predictions, addressing some of the complexities of time series data. The Transformer model, relying entirely on self-attention to compute representations of its input and output, represents a paradigm shift by eliminating the need for sequence-aligned RNNs or convolutional layers, thus offering a promising alternative for enhancing time series forecasting.

Despite these advances, deep neural networks (DNNs) often suffer significantly when test distributions deviate from those seen during training. Studies have demonstrated that DNNs can experience severe performance degradation under such conditions [21, 7, 16].

## 3.3 Adaptive Approaches and Real-Time Adaptation

To mitigate these issues, distribution matching has become increasingly important. Traditional domain adaptation (DA) techniques [11, 27, 29, 30, 32] and domain generalization (DG) [31] strategies have attempted to bridge these gaps by learning domain-invariant features. However, these methods generally do not account for the temporal dynamics inherent in time series data, and are predominantly designed for static images or classification tasks in convolutional contexts. AdaRNN [9] tackles these challenges by focusing on temporal distribution characterization and matching. This framework dynamically adapts to distribution shifts over time, enhancing forecasting accuracy by aligning the model's predictions with the evolving data characteristics.

Building on the concept of dynamic adaptation, test-time training introduces a novel approach to real-time model updating. This technique involves a self-supervised learning objective optimized both during and after the initial training phase, allowing the model to adjust to new data continuously. Originally demonstrated in fields like image classification [26], test-time training has been enhanced by integrating techniques from other advanced learning strategies.

For instance, this approach has been expanded beyond multi-task learning by incorporating gradient-descent based meta-learning [4]. Further expanding the scope of test-time training, this concept has also been adapted to reinforcement learning through self-supervised policy adaptation [14], demonstrating its versatility and effectiveness in dynamic environments.

SAF [1] builds upon these developments and is the first to apply the test-time training concept specifically to time series forecasting. By integrating the time component into the self-supervised learning objective, we enable our models to adjust not only to the current data but also to anticipate future changes, making this approach especially effective in sectors like finance and healthcare where non-stationarity is a major challenge.

Chapter 4

# Basic Definitions

## 4.1 AdaRNN

In the domain of time series forecasting, one of the primary tasks is the r-step ahead prediction. This involves predicting future values based on past observations. For instance, given a sequence of daily stock prices, the goal might be to forecast the stock price 7 days from the last observed value. While this seems straightforward, the underlying data's dynamic nature introduces complexities that can significantly impact the accuracy of these predictions.

A primary challenge arises from the Covariate Shift [25]. In many machine learning scenarios, models are trained on one dataset (the training set) and then validated or tested on another (the test set). A covariate shift occurs when the input data distribution in the training set doesn't match that of the test set. This shift can arise due to several reasons, such as:

- **Sampling bias**: where the training data might have been collected under different conditions or from different sources compared to the test data.
- **Temporal changes**: as with time series data, where the older data used for training might not represent the current or future state due to evolving conditions.
- **Non-stationarity**: in the data-generating process, where the underlying mechanisms that generate the data might change over time or across conditions.

For instance, consider a model trained to predict disease outbreaks based on historical data from a particular city. If this model is then tested on data from a different city with different demographics, climate, and healthcare infrastructure, there might be a covariate shift due to these differences in data distribution. Despite this shift, the relationship between the inputs and the output remains unchanged. However, this misalignment can degrade the performance of models, especially if they aren't designed to account for these distributional changes.

Time series data, with its temporal dimension, adds an additional layer of complexity known as the Temporal Covariate Shift (TCS) [9]. Unlike a standard covariate shift where the distributional change is across datasets, TCS involves changes within the same dataset over time. As the series progresses, earlier data points might have a different distribution than later ones. This continuous evolution poses a significant challenge for models, as they need to remain adaptive to these shifts to maintain prediction accuracy. This is illustrated in Figure 4.1.

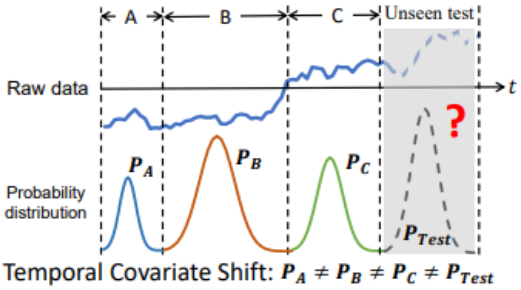


FIGURE 4.1: Illustration of Temporal Covariate Shift (TCS) [9]

Temporal Covariate Shift (TCS) occurs within a time series dataset $\mathcal{D}$ comprised of labeled segments. This dataset $\mathcal{D}$ can be partitioned into $K$ intervals or periods, symbolized as $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K\}$. Each interval $\mathcal{D}_k$ consists of data points that fall between two indices, such that the $k$-th interval $\mathcal{D}_k$ encompasses data points from the index $n_{k-1} + 1$ to $n_k$. The first index $n_0$ is set to 0 and the last index $n_K$ equals $n$, which denotes the total number of segments.

The key characteristics of Temporal Covariate Shift are:

1. Within a given interval $\mathcal{D}_k$, all segments adhere to the same data distribution, which can be represented as $P_{\mathcal{D}_k}(x, y)$.

2. Across different intervals, say $\mathcal{D}_i$ and $\mathcal{D}_j$ for any $1 \leq i \neq j \leq K$, the marginal distributions of the covariates are different, i.e., $P_{\mathcal{D}_i}(x) \neq P_{\mathcal{D}_j}(x)$.

3. Despite the difference in marginal distributions, the conditional distributions of the output given the covariates remain the same, meaning $P_{\mathcal{D}_i}(y|x) = P_{\mathcal{D}_j}(y|x)$ for any two intervals $i$ and $j$.

Building upon the foundational idea of r-step ahead prediction, the r-step TS prediction under TCS emerges as a more intricate challenge. It's not merely about forecasting future values; it's about ensuring that these forecasts are accurate even as the underlying data distribution evolves. This task demands a model that can predict while simultaneously adapting to the ever-changing data landscape.

To solve the challenges of time series forecasting, a model proposed in [9] named AdaRNN stands out as a tailored solution. It's designed to handle the complexities brought about by the Temporal Covariate Shift (TCS), and its structure revolves around two main components.

The first component, Temporal Distribution Characterization (TDC), focuses on understanding the time series data. It aims to identify and describe the different patterns in the data over time. To do this, AdaRNN breaks down the training data into specific sections or periods. Each of these periods has its own unique pattern or distribution. This breakdown helps AdaRNN capture the most varied and distinct patterns in the data.

After understanding the data, the next step is to make sure the prediction model can work well with these different patterns. This is where the second component, Temporal Distribution Matching (TDM), comes into play. TDM's job is to adjust the prediction model to match these different data patterns. It tweaks the model so that it can predict accurately, even when the data pattern changes. The end result is

a prediction model, called M, that's both accurate and flexible. This model is then used to make predictions on new data, ensuring that the forecasts are reliable, even if the data changes over time.

In short, these two components make AdaRNN a strong tool for time series forecasting. It first understands the data and then adjusts its predictions to match any changes, ensuring accuracy throughout.

If we delve deeper into the details, the TDC works by solving an optimization problem. The primary objective of this optimization is to maximize the dissimilarity between the identified periods. Specifically, the aim is to segment the time series into periods that are as distinct from each other as possible. Each segment is consecutive and has a minimum and maximum length. This ensures that the entire time series is utilized, and no segment is either too minute or overly large. It's essential to highlight that the aggregate of these segments' cardinalities sums up to the whole length of the sequence, emphasizing the polynomial nature of the problem. The mathematical representation of this objective is as follows:

$$\begin{aligned} &\max_{0<K\leq K_0} \max_{n_1,\cdots,n_K} \frac{1}{K} \sum_{1\leq i\neq j\leq K} d\left(\mathcal{D}_i, \mathcal{D}_j\right) \\ &\text{s.t. } \forall i, \Delta_1 < |\mathcal{D}_i| < \Delta_2; \sum_i |\mathcal{D}_i| = n \end{aligned} \tag{4.1}$$

where $d$ is a distance metric, $\Delta_1$ and $\Delta_2$ are predefined parameters to avoid trivial solutions (e.g., very small values or very large values may fail to capture the distribution information), and $K_0$ is the hyperparameter to avoid over-splitting. The idea can be seen at Figure 4.2:

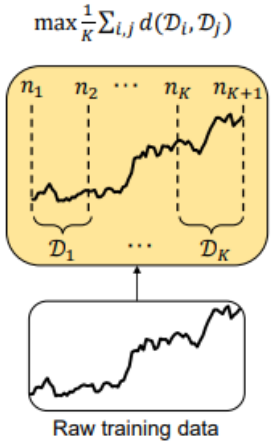


FIGURE 4.2: Temporal distribution characterization (TDC)[9]

The authors mention that the time series splitting optimization problem in Equation 4.1 is computationally intractable. They also say that by selecting a proper distance metric, the optimization problem can be solved with dynamic programming. However, they do not provide proof for these statements and opt for a greedy algorithm instead, that can be seen in algorithm 1.

Then, once we have the learned time periods, the TDM module learns the common knowledge shared by different periods via matching their distributions. Figure 4.3 is an overall representation of the TDM module. The loss of TDM for prediction, $\mathcal{L}_{\text{pred}}$ , can be formulated as:

$$\mathcal{L}_{\text{pred}}(\theta) = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{|\mathcal{D}_j|}\sum_{i=1}^{|\mathcal{D}_j|}\ell\left(\mathrm{y}_i^j, \mathcal{M}\left(\mathrm{x}_i^j;\theta\right)\right),$$

where $\left(\mathrm{x}_i^j, \mathrm{y}_i^j\right)$ denotes the $i$-th labeled segment from period $\mathcal{D}_j$, $\ell(\cdot,\cdot)$ is a loss function, e.g., the MSE loss, and $\theta$ denotes the learnable model parameters.

TDM introduces an importance vector denoted as $\alpha$. This vector is used to learn the relative importance of the hidden states inside the RNN. By weighting all the hidden states with a normalized $\alpha$, the module can dynamically reduce the distribution divergence across different periods. For each pair of periods, there is an specific $\alpha$.

Given two specific periods, let's say $D_i$ and $D_j$ the loss of temporal distribution matching is formulated as:

$$\mathcal{L}_{tdm}\left(\mathcal{D}_i, \mathcal{D}_j;\theta\right) = \sum_{t=1}^{V}\alpha_{i,j}^t d\left(\mathbf{h}_i^t, \mathbf{h}_j^t;\theta\right),$$

where $\alpha_{i,j}^t$ denotes the distribution importance between the periods $\mathcal{D}_i$ and $\mathcal{D}_j$ at state $t$.

By integrating the two previous equations, the final objective of temporal distribution matching (one RNN layer) is:

$$\mathcal{L}(\theta,\boldsymbol{\alpha}) = \mathcal{L}_{\text{pred}}(\theta) + \lambda\frac{2}{K(K-1)}\sum_{i,j}^{i\neq j}\mathcal{L}_{tdm}\left(\mathcal{D}_i, \mathcal{D}_j;\theta,\boldsymbol{\alpha}\right),$$

In order to learn $\alpha$, the authors propose a Boosting-based importance evaluation algorithm. Before that, the network parameter $\theta$ undergoes pre-training across all periods. The objective of this is to ensure that richer hidden states representations are achieved, facilitating the way for effective learning of $\alpha$.

The essence of the algorithm lies in assessing the importance of hidden states within the RNN. For every layer, an importance weight is defined, drawing from the cross-domain distribution distance. At each time step, during epoch $n$, the distance between the distribution states is evaluated using the updating function $G()$ defined below. If the distribution distance at epoch n+1 exceeds that of epoch n, the importance value is boosted. This process can be expressed mathematically as:

$$\alpha_{i,j}^{t,(n+1)} = \begin{cases}\alpha_{i,j}^{t,(n)} \times G\left(d_{i,j}^{t,(n)}, d_{i,j}^{t,(n-1)}\right) & d_{i,j}^{t,(n)} \geq d_{i,j}^{t,(n-1)} \\ \alpha_{i,j}^{t,(n)} & \text{otherwise}\end{cases}$$

where

$$G\left(d_{i,j}^{t,(n)}, d_{i,j}^{t,(n-1)}\right) = \left(1+\sigma\left(d_{i,j}^{t,(n)} - d_{i,j}^{t,(n-1)}\right)\right)$$

is the updating function computed by distribution matching loss at different learning stages.

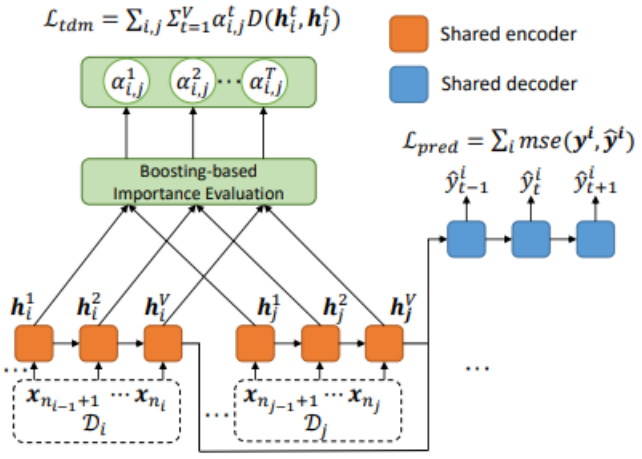


FIGURE 4.3: Temporal distribution matching (TDM) [9]

**Algorithm 1:** AdaRNN

**Data:** Time series dataset $D$, $\lambda$, max iteration number *Iter*, hyperparameters $\Delta_1, \Delta_2$, distance metric $d(.,.)$
**Result:** Optimal parameters $\theta^*$, $\alpha^*$

**begin**
  `// Temporal Distribution Characterization`
  Set $N = 10$;
  Split $D$ into $N$ equal-length segments $\{S_1, S_2, \ldots, S_N\}$;
  Label the start and end points of $D$ as $A$ and $B$, respectively;
  Initialize $K_{\text{values}} = \{2, 3, 5, 7, 10\}$;
  **for** *each* $K$ *in* $K_{values}$ **do**
    **if** $K = 2$ **then**
      Find a splitting point $C$ from the 9 potential splitting points to maximize $d(S_{AC}, S_{CB})$;
    **else**
      Using previously determined splitting points, find additional points to achieve $K$ segments, maximizing distance between adjacent segments;
  Store the periods as $\{D_i\}$ from the optimized splits;
  `// End-to-end pre-training`
  **for** *each period* $D_j$ *in* $\{D_i\}$ **do**
    **for** *each labeled segment* $(x'_ij, y'_ij)$ *in* $D_j$ **do**
      Calculate segment loss: $f(y'_ij, M(x'_ij; \theta))$;
    Average the segment losses for $D_j$ to get period loss;
  Average period losses across all $K$ periods to get $L_{pred}$;
  Minimize $L_{pred}$ with respect to model parameters $\theta$ with $\lambda = 0$;
  **for** *iteration* $i = 1$ *to Iter* **do**
    Compute all hidden states of the RNN as: $h'_i = \delta(x_i, h'_{i-1})$;
    **for** *each pair of periods* $(D_j, D_k)$ **do**
      i. Compute the loss of temporal distribution matching $L_{dm}(D_j, D_k; \theta)$ using Equation (3);
    Calculate the final objective of temporal distribution matching for one RNN layer:
    $L(\theta, \alpha) = L_{pred}(\theta) + \lambda \times \frac{2}{K(K-1)} \times \sum\sum L_{dm}(D_j, D_k; \alpha, \theta)$;
    Boosting-based importance evaluation for hidden states:
      i. Pre-train on network parameter $\theta$ using $L_{pred}$ to learn $\alpha$;
      ii. Use boosting-based method (Equations (6) and (7)) to adjust $\alpha$ values based on distribution distance of epochs;
    Dynamically update the distribution divergence of cross-periods using $\alpha$ and $\theta$;
  Return Optimal parameters $\theta^*$, $\alpha^*$;

## 4.2 SAF

The Self-Adaptive Forecasting (SAF) method is rooted in several foundational principles that set it apart in the realm of time-series forecasting:

1. **Considering the Current State of Data**: At its core, SAF believes that to make accurate predictions, it's not enough to just look at past data points. Instead, it's crucial to also consider the overall 'mood' or 'state' of the data around the time of prediction. This approach ensures that the forecasting is always in tune with the current dynamics of the data.

2. **Learning on the Go with Self-supervised Learning**: One of SAF's standout features is its ability to teach itself using the data it has. This is known as self-supervised learning. Since future data is unknown during prediction, SAF continuously adjusts its approach based on the data it's currently analyzing. This self-teaching mechanism ensures that SAF remains adaptable and can handle new or unexpected data trends.

3. **Backcasting for Better Pattern Recognition**: SAF employs a unique technique called 'backcasting'. In this process, it hides a portion of the past data and then tries to guess the hidden part using the data it can see. This exercise is like a self-test, helping SAF to understand and recognize patterns in the data better. If there's an unexpected event in the data that changes the usual patterns, both the backcasting and forecasting will reflect this change. This continuous self-checking ensures that SAF is always aligned with the true nature of the data.

4. **Using Errors as Learning Opportunities**: Mistakes are often seen as setbacks, but for SAF, they're valuable learning opportunities. After the backcasting process, SAF evaluates how accurate its guesses were. If there are discrepancies between its guesses and the actual data, SAF uses this error information to refine its predictions. This principle ensures that SAF is always learning and improving, making its future predictions even more accurate.

To understand the learning mechanism of the Self-Adaptive Forecasting (SAF) model, let's analyze its different parts. Its core consists of three main components. The Encoder processes the input data, producing encoded representations that capture the essence of the data within the encoding window. The Forecast Decoder takes these encoded representations and uses them to predict future data points. The Backcast Decoder employs the encoded representations to predict past data points, aiding in the self-supervised learning process.

Central to SAF's adaptability is its unique backcasting technique. Here, a segment of the input data is intentionally masked, and SAF then tries to predict this segment using the available unmasked data. The primary goal of this exercise is to minimize any discrepancy between the predicted values and the actual data points of the masked segment.

The difference between the backcasts produced by the backcast decoder and the actual data is quantified using a loss function. By continuously trying to minimize this loss, SAF ensures that its backcasting, and by extension, its forecasting, remains precise and accurate.

Now that the different parts and components are clear, let's explain how everything comes together during training.

Initially, the encoder processes the input data to capture its temporal dynamics, creating a rich, contextual representation. This representation, with some masked parts, is then utilized by the backcast decoder to predict the hidden segments of the data, an operation that mimics inference by requiring the model to fill in gaps based on learned patterns. Following this prediction, a crucial adaptation step occurs: the encoder and backcast decoder are updated based on the backcast prediction loss. This update is important to recalibrate the model based on the current data, allowing the model to refine its parameters to better align with the data's underlying temporal structure, enhancing its capacity to accurately reconstruct past values.

After updating the encoder and backcast decoder, the SAF model proceeds to generate forecasts for future values. Like in a standard training loop, the updated encoder is used to get the representations from the current data. Then, the processed data is given to the decoder, where it takes the encoded representation and generates the prediction. The forecasting loss, derived from comparing these predictions against actual future values, is used to update both the encoder and the forecasting decoder. This ensures that the model's forecasting capabilities are continually refined, aligning its predictions more closely with real-world outcomes. The idea behind this explanation is can be seen at Figure 4.4.

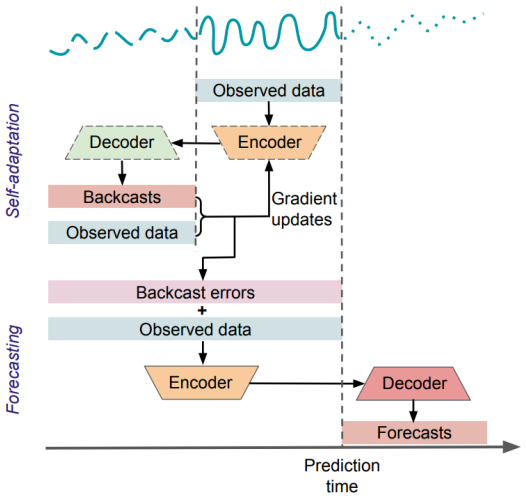


FIGURE 4.4: SAF Overview [1]

When it comes to forecasting outside of training, the model is still updated, but only the encoder and backcast decoder, based on the current data that we are trying to predict. As before, the data is firstly processed by the encoder and backcast decoder, and after the self-adaptation, the data is processed by the updated encoder and then given to the already fixed forecasting decoder.

## 4.3 Transformers

The Transformer architecture, introduced in the seminal paper "Attention Is All You Need," [28] presents a novel approach to sequence-to-sequence tasks, avoiding traditional recurrent and convolutional layers in favor of attention mechanisms. At the heart of this architecture is the attention function, which maps a query and a set of key-value pairs to an output. This output is essentially a weighted sum of the values, where each weight is determined by the compatibility of the query with its corresponding key.

On this section, I will get into the details of the Transformer's model architecture:

### 4.3.1 Encoder and Decoder Stacks

The **encoder** is structured with multiple identical layers. Each of these layers is composed of two primary components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. To enhance the flow of information through the network, residual connections are used around both sub-layers, with the output being normalized. All sub-layers, including the embedding layers, produce outputs with a consistent dimensionality, ensuring uniformity across the encoder.

The **decoder** mirrors the encoder's structure with an equal number of identical layers but introduces an additional third sub-layer. This sub-layer conducts multi-head attention over the encoder's output. Like the encoder, the decoder uses residual connections around its sub-layers, followed by normalization. A unique feature of the decoder is the modification of its self-attention mechanism, which restricts positions from attending to future positions, preserving the model's auto-regressive property.

### 4.3.2 Attention Mechanisms in the Transformer Model

The Transformer model employs an attention mechanism to weigh the importance of different parts of the input data. The output is a weighted sum of the values, with the weight of each value determined by the compatibility of its corresponding key with the query.

The model introduces the "Scaled Dot-Product Attention" mechanism. It operates on queries and keys of a certain dimension, and values of another. The attention weights are determined by computing the dot products of the query with all keys, scaling the result, and then applying a softmax function. In practical scenarios, attention is computed on multiple queries simultaneously, grouped into a matrix. The keys and values are similarly organized into matrices, and the resulting output matrix is derived from a specific formula involving these matrices.

The model further enhances the attention mechanism with "Multi-Head Attention". Instead of a singular attention function, multiple attention heads are employed. The queries, keys, and values are projected multiple times using different learned linear projections. Each set of projected queries, keys, and values undergoes the attention function in parallel. The outputs are then concatenated and

subjected to another linear projection. This design allows the model to focus on information from varied representation subspaces at different sequence positions.

The Transformer uses multi-head attention in three distinct ways:

1. **Encoder-Decoder Attention**: Queries come from the preceding decoder layer, while the memory keys and values are derived from the encoder's output. This ensures every position in the decoder can focus on all positions within the input sequence.

2. **Encoder Self-Attention**: The keys, values, and queries all originate from the same location, specifically the output of the encoder's previous layer. This allows each position in the encoder to attend to every other position in its preceding layer.

3. **Decoder Self-Attention**: Each position in the decoder attends to all prior positions, including its own. To maintain the model's auto-regressive property, information flow from future positions is prevented by masking out certain values in the softmax input.

### 4.3.3 Embedding and Softmax

In the Transformer model, learned embeddings are employed to convert both input and output tokens into vectors of a consistent dimensionality. This approach is consistent with many sequence-to-sequence models. The decoder's output is transformed into predicted next-token probabilities using a learned linear transformation coupled with a softmax function.

A distinctive feature of the Transformer is the sharing of the weight matrix across the two embedding layers and the pre-softmax linear transformation. Within the embedding layers, these weights are scaled by a factor related to the dimensionality of the embeddings to ensure appropriate magnitude and variance.

### 4.3.4 Positional Encoding

The Transformer model, given its lack of recurrence and convolution, requires a mechanism to account for the order of tokens in a sequence. To address this, "positional encodings" are added to the input embeddings at the foundational levels of both the encoder and decoder stacks. These encodings are dimensionally compatible with the embeddings, allowing for their summation.

While there are various strategies for positional encoding, both learned and fixed, the Transformer model employs sinusoidal functions of differing frequencies for the purpose. The positional encoding for a given position and dimension is defined using sine and cosine functions, with the arguments of these functions designed to capture the relative positions of tokens in the sequence.

The choice of sinusoidal functions was motivated by the hypothesis that they would enable the model to attend based on relative positions. This is because for any consistent offset, the positional encoding for a given position plus the offset can be expressed as a linear function of the positional encoding for the given position.

Chapter 5

# Our approach

This section outlines our approach for enhancing time series prediction models, which includes two proposals for Temporal Distribution Characterization (TDC) and a customized transformer model implementation.

## 5.1 Temporal Distribution Characterization

### 5.1.1 Dynamic Programming Proposal

For the temporal distribution characterization, we formalize and propose the dynamic programming approach suggested (but not implemented) in the paper AdaRNN[9].

It is designed to find an effective partitioning of a time series dataset for a given number of splits, $K_{\max}$. The method constructs a dynamic programming table, $DP[i][k]$, where $DP[i][k]$ represents the maximum dissimilarity achievable by partitioning the first $i$ elements of the time series into $k$ segments.

An auxiliary table $SPLIT[i][k]$ is also used to trace back the splits leading to this effective segmentation, storing the length of the last segment ending at the $i$-th point for $k$ segments.

The algorithm's design is based on incremental optimization, and thus it is important to understand its implications.

1. **Local optimality for segments**: For each $i$, the algorithm determines the optimal segmentation of the first $i$ elements. This segmentation is optimal based on the information available up to that point.

2. **Incremental Nature**: As the algorithm progresses to $i + 1$, it builds upon the segmentation determined for $i$, without reconsidering all possible segmentations from scratch. This approach is efficient but means that the segmentation for $i + 1$ is based on the assumption that the previous segmentation for $i$ was globally optimal.

In the context of the entire time series, the proposed segmentation can be seen as a good approximation. It's derived from locally optimal decisions at each step, but without re-evaluating all past decisions at each new step. Thus, global optimality for the entire series is not guaranteed.

This approach provides a good balance between computational efficiency and segmentation quality. While it may not always yield the globally optimal solution for the entire series, it offers a practically effective method for segmenting large time series data. The algorithm can be seen at algorithm 2.

To address the complexity of the proposed method, we will analyze both time and space.

In terms of time complexity, the main contributing factor to time complexity is the three nested loops iterating up to $K_{\max}$, $n$, and $\Delta_2$ respectively. Therefore, the time complexity is $O(n \times \Delta_2 \times K_{\max})$. The computational cost inside the loops consists mainly of distance calculations and simple arithmetic, which doesn't add higher-order terms to the complexity. Given that we can divide the sequence at most n times, we can set an upper bound for $k_{\max}$. Also, the maximum length of any segment can be at most $n$, therefore setting an upper bound for $\Delta_2$. With all this, our proposed method can be bounded by $O(n^3)$.

For space complexity, we maintain two tables, $DP$ and $SPLIT$, both of size $n \times K_{\max}$. Hence, the space complexity is $O(n \times K_{\max})$. As before, it can be bounded by $O(n^2)$.

To improve the understanding of the dynamic programming approach, a high-level overview of algorithm 2 is presented:

1. **Initialization**: The algorithm initializes the dynamic programming table $DP$ and the auxiliary table $SPLIT$. $DP[i][k]$ starts with a value representing the lowest possible dissimilarity (e.g., negative infinity), and $SPLIT[i][k]$ starts with zeros. The base case of $DP[i][0]$ is set to zero, as there is no dissimilarity with zero segments.

2. **Exploration of Segment Counts**: The outer loop iterates through possible values of segment counts from 2 to $K_{max}$. For each $K$, a new $DP$ table and *splits* array are initialized to keep track of the best configuration specific to that $K$.

3. **Iterative computation**: For each value of $K$, the algorithm iterates through each data point (up to $n$) and, for each data point $i$, explores potential segment lengths from *lower_bound* to *upper_bound*. Within these bounds, the algorithm calculates the dissimilarity measure for a potential new segment and evaluates whether adding this segment improves the total dissimilarity for the first $i$ elements partitioned into $k$ segments.

4. **Updating the tables**: If adding the current segment improves the total score for partitioning the first $i$ elements into $k$ segments, the algorithm updates $DP[i][k]$ with this new value and records the segment length in $splits[i][k]$. This process ensures that at each step, $DP[i][k]$ maintains the highest dissimilarity found for partitioning the first $i$ elements into $k$ segments, and $SPLIT[i][k]$ keeps track of the last segment's length in this optimal partitioning.

5. **Selecting the Best Configuration**: After processing each $K$, the algorithm checks if the score for partitioning the entire dataset into $K$ segments (i.e., $DP[n][k]$) is better than the maximum score encountered so far. If it is, the maximum score and corresponding segment configuration are updated.

6. **Reconstruction of Optimal Segmentation**: Once all potential segment counts have been considered, the algorithm reconstructs the optimal segmentation for the best $K$ found, starting from the end of the time series. The reconstruction uses the $splits$ array to determine the lengths of the segments that contributed to the optimal partitioning, building the segmentation from the last segment backwards to the beginning of the time series.

### 5.1.2 Simulated annealing proposal

In order to try to achieve the best possible result in the optimization task (TDC), simulated annealing has also been implemented and tested.

It is an optimization technique inspired by the physical process of annealing in metallurgy. Annealing involves heating and then slowly cooling a material to alter its physical properties through changes in its internal structure. In the context of optimization, it is used to find a good approximation to the global optimum of a given function in a large search space.

Let's explain briefly how it works, before diving into the actual implementation.

1. **Initial solution**: We start with a random solution.
2. **Temperature**: The temperature is a control parameter that decreases over time. High temperatures allow the algorithm to explore the search space more freely, accepting worse solutions with higher probability, which helps in avoiding local optima. As the temperature decreases, the algorithm becomes more conservative in accepting worse solutions.
3. **Random Perturbations**: At each step, the algorithm makes a small change to the current solution (a "perturbation") to generate a new solution.
4. **Acceptance Criterion**: The new solution is evaluated using a certain acceptance criterion, which determines whether to move to this new solution or stay with the current one. The probability of moving to a worse solution decreases as the temperature drops.
5. **Cooling Schedule**: The temperature is gradually decreased based on a cooling schedule.

In the context of our problem, it will be used to optimize the segmentation of the series to achieve maximizing dissimilarity between segments.

The implementation of this approach can be seen in algorithm 3.

**Algorithm 2:** Dynamic Programming Approach for Time Series Segmentation and Segment Reconstruction

```
Function TimeSeriesSegmentation(data, K_max, lower_bound, upper_bound):
    n ← length(data)
    max_score ← -∞
    best_K ← 0
    best_splits ← array[n + 1]
    for K ← 2 to K_max do
        DP ← array[n + 1][K + 1] initialized with -∞
        splits ← array[n + 1][K + 1] initialized with 0
        for i ← 0 to n do
            DP[i][0] ← 0
        for i ← 1 to n do
            for seg_len ← lower_bound to upper_bound do
                if i - seg_len ≥ 0 then
                    curr_segment ← data[i - seg_len : i]
                    for j ← 1 to K do
                        if j == 1 then
                            if i == seg_len then
                                DP[i][j] ← 0
                                splits[i][j] ← seg_len
                            continue
                        segments ← ReconstructSegments(data, i - seg_len, j - 1, splits)
                        average_distance ← sum(segment_distance(seg, curr_segment) for seg in segments) / len(segments)
                        if DP[i][j] < DP[i - seg_len][j - 1] + average_distance then
                            DP[i][j] ← DP[i - seg_len][j - 1] + average_distance
                            splits[i][j] ← seg_len
        if DP[n][K] > max_score then
            max_score ← DP[n][K]
            best_K ← K
            best_splits ← splits
    return max_score, ReconstructSegments(data, n, best_K, best_splits)

Function ReconstructSegments(data, end, num_segments, splits):
    segments ← list()
    current_end ← end
    for k ← num_segments to 1 step -1 do
        seg_len ← splits[current_end][k]
        segment ← data[current_end - seg_len : current_end]
        segments.append(segment)
        current_end ← current_end - seg_len
    segments.reverse()
    return segments
```

**Algorithm 3:** Simulated Annealing for Time Series Segmentation

```
Function TimeSeriesSegmentation(data, K, lower_bound, upper_bound):
    Initialize best_solution as None and best_score as -∞
    for K ← 2 to K_Max do
        for trial ← 1 to 10 do
            initial_solution ← Generate an initial solution with K segments;
             initial_temperature ← 1.0
            cooling_rate ← 0.05; num_iterations ← 10000
            current_solution ← initial_solution, current_score ←
             ComputeObjective(initial_solution, data)
            for i ← 1 to num_iterations do
                temperature ← max(initial_temperature × (cooling_rate ^ i),
                 0.01)
                neighbor ← GenerateNeighbor(current_solution,
                 min_length, max_length)
                neighbor_score ← ComputeObjective(neighbor, data)
                if neighbor_score > current_score or Random(0, 1) <
                 exp((neighbor_score - current_score) / temperature) then
                    current_solution ← neighbor, current_score ←
                     neighbor_score
            if current_score > best_score then
                best_score ← current_score, best_solution ←
                 current_solution
    return best_score, best_solution
Function GenerateNeighbor(current_solution, min_length, max_length,
 data_length):
    if len(current_solution) < 2 then
        return current_solution
    neighbor ← current_solution.copy()
    segment_to_adjust ← randint(0, len(neighbor) - 2)
    current_start ← neighbor[segment_to_adjust][0]
    current_end ← neighbor[segment_to_adjust][1]
    if segment_to_adjust == 0 then
        max_end ← min(data_length - 1, current_start + max_length)
        min_end ← max(current_start + min_length, 1)
    else
        max_end ← min(neighbor[segment_to_adjust + 1][1] - min_length,
         current_start + max_length)
        min_end ← max(neighbor[segment_to_adjust - 1][1] + min_length,
         current_start + min_length)
    if min_end < max_end then
        new_end ← randint(min_end, max_end)
        neighbor[segment_to_adjust][1] ← new_end
        if segment_to_adjust < len(neighbor) - 1 then
            neighbor[segment_to_adjust + 1][0] ← new_end
    return neighbor
```

Following the detailed exposition of the dynamic programming approach and the implementation of simulated annealing for Temporal Distribution Characterization (TDC), it is crucial to evaluate the efficacy and outcomes of these methodologies in comparison to the original TDC method as originally proposed. This comparative analysis aims to highlight the improvements, if any, in terms of computational efficiency, segmentation quality, and overall performance in time series prediction tasks.

To this end, a comprehensive comparison will be conducted in chapter 6: "Methodology" and chapter 7: "Results".

## 5.2 Adaptive Transformer Model

To complement our dynamic programming approach to TDC, we introduce an adaptive transformer model specifically designed for time series prediction. This model incorporates the identified temporal segments into its architecture, enabling it to better capture and utilize the underlying temporal dynamics.

### 5.2.1 Model Architecture

Our AdaTransformer model is specifically designed to address the unique challenges presented by time series data. In a quick overview of its architecture we see the integration of an input projection layer, a Transformer encoder, dual decoders for forecasting and backcasting, a gating mechanism similar to the approach in adaRNN, and an optional bottleneck layer.

The model begins its processing with an input projection layer, a critical component that reshapes the dimensionality of incoming time series data. By applying a linear transformation, the input projection layer adjusts the size of the input features, ensuring they match the expected dimensions required by the subsequent stages of the model.

This dimensionality transformation is important for several reasons. Firstly, it enhances the model's ability to capture and represent complex patterns within the data. By projecting the input into a higher-dimensional space, the AdaTransformer can explore a richer feature landscape, potentially uncovering temporal dynamics that might be hidden in lower-dimensional representations. This capability is particularly beneficial in time series forecasting, where the intricacies of the data usually demand a precise approach to feature analysis.

Moreover, the input projection layer ensures that the AdaTransformer model maintains its structural integrity and flexibility across various data sources. It acts as a bridge, allowing the model to process diverse datasets without necessitating changes to its core architecture.

Another advantage of this layer is the option to balance between computational efficiency and model performance. Depending on the task at hand, projecting the data into a lower or higher-dimensional space can either streamline the model's operations or enhance its analytical depth.

Following the initial transformation through the input projection layer, we add the positional encoding information to our transformed data. This process is essential for embedding temporal information into the model, enabling it to recognize and utilize the sequence order of the time series data.

Positional encoding is applied by generating a unique encoding for each position in the sequence. This encoding is then added to the input features, ensuring that the model can distinguish between data points based on their position in the sequence. The method used to generate these encodings involves both sine and cosine functions, which vary according to the position of each data point in the sequence. This variation allows the model to capture the temporal relationship between points, a critical aspect of time series forecasting.

The mathematical formulation behind positional encoding is designed to provide a balance between preserving the position's information and allowing the model to easily learn to attend to these positions. By applying these encodings, the AdaTransformer is equipped with the ability to understand the sequential nature of the data, an understanding that is essential for accurate forecasting.

This step is particularly important because, unlike traditional data types, time series data is inherently ordered, and the relationship between consecutive data points can carry significant predictive power. The positional encoding ensures that this temporal ordering is not lost during the model's processing, thereby enhancing its ability to make informed predictions about future data points.

Following the application of positional encoding, the AdaTransformer model advances to the next stage, involving the Transformer encoder. As the name indicates, it encodes the temporal dependencies and patterns embedded within the time series data. The encoder operates on the inputs, now enriched with positional information, employing a self-attention mechanism to analyze the interrelations among various time steps within the sequence. This process allows the model to differentiate the relative importance of each observation within the broader context of the series, thereby unlocking the potential to exploit the underlying temporal dynamics to its advantage.

An important feature of the AdaTransformer's encoder is its composition of multiple stacked encoder layers. This multi-layered approach amplifies the model's capacity to capture and process temporal dependencies. Each layer in the stack contributes to refining the representation of the input sequence, allowing the model to capture a wide spectrum of temporal relationships, from immediate connections to long-range dependencies. This stacked architecture ensures a depth of analysis that single-layer encoders might not achieve, enabling the AdaTransformer to construct a more detailed and comprehensive understanding of the time series data. This method contrasts with traditional recurrent neural networks (RNNs), which process data sequentially and may struggle with long-range dependencies due to the vanishing gradient problem. The parallel processing capability of the Transformer encoder, facilitated by its self-attention mechanism, alliviates these limitations, offering enhanced efficiency and a more robust framework for capturing temporal dynamics.

The output from the final encoder layer represents a rich, contextually informed summary of the input sequence. This high-dimensional representation encapsulates the different temporal patterns within the data, setting a solid foundation for

the model's forecasting capabilities.

Upon completing the encoding process, the AdaTransformer model reapplies positional encoding to the output of the encoder stack. This step reintroduces temporal information to the encoded representations, ensuring that the subsequent processing stages remain sensitive to the sequence's temporal structure. The reapplication of positional encoding is crucial for maintaining the temporal context that might have been abstracted during the encoding phase, thereby preserving the model's ability to interpret and leverage time-based patterns effectively.

Following the reapplication of positional encoding, some part of the data is masked. This is where the SAF integration comes into play.

The model channels the masked data through the backcast decoder stack, which mirrors the encoder in size and structure. This symmetry between the encoder and decoder stacks is a deliberate design choice, aimed at harmonizing the processing capabilities of both components. The backcast decoder stack, equipped with its own self-attention mechanisms, is tasked with reconstructing the input sequence with the available information. Through this mechanism, the AdaTransformer not only enhances its understanding of the data's temporal dynamics for a more comprehensive internal representation, but also dynamically adapts to new data variations during inference.

At this point, as we explained previously when describing the SAF procedure, a backcast loss is computed based on the predicted and actual values for the masked positions. This loss is then used to update the encoder and the backcast decoder through gradient descent. This process can be seen at Figure 5.1.

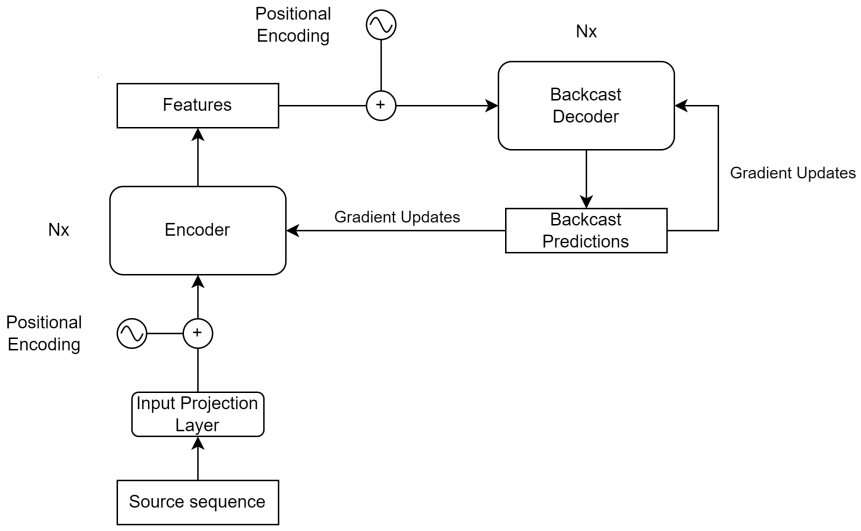


FIGURE 5.1: Self-update process of the architecture

Once both components are updated, the process begins "again". The data that was enriched with positional information is processed through the updated encoder.

As before, positional encoding is again applied to this new representation of the data, in order to reintroduce the temporal information.

Now, the output of the encoder serves two purposes. First of all, the standard forecasting task, where the stack of the forecasting decoder is in charge of generating the predictions for the given data.

As before, the stack size remains the same as the previous components. Each decoder layer is designed to build upon the contextualized representations provided by the encoder, refining and translating them into predictive outputs. This stacked approach allows the model to iteratively refine its predictions, layer by layer, drawing on a progressively enriched understanding of the input sequence. The decoder's ability to access the entire encoded sequence at each step enables it to consider both local and global temporal relationships, facilitating accurate forecasting.

To finish this process, the model directs the decoder stack's output towards the final transformation stage, which maps the high-dimensional representations to the desired output size. This stage can manifest in two forms, depending on the complexity of the task and the model's configuration:

- **Direct Linear Transformation:** For tasks where the temporal dynamics are relatively straightforward, the model employs a direct linear layer. This approach is efficient and effective for scenarios where the relationship between the temporal features and the output is linear or when computational simplicity is prioritized.
- **Bottleneck Layer Configuration:** In contrast, for more complex forecasting challenges, where the data exhibits intricate nonlinear patterns, the model opts for a bottleneck architecture. This configuration introduces additional layers of nonlinearity and normalization, enabling the model to capture and model complex relationships within the data. The bottleneck layer, through its sequence of transformations, including activation and normalization, allows for a higher level of understanding and representation of the data, potentially leading to enhanced forecasting accuracy.

The other purpose of the encoder's output is the temporal distribution matching, as explained in the adaRNN section. This module is only used during training. Once the training phase finished, and we need to predict new data the temporal distribution matching part is no longer used, and the prediction is obtained based on the encoder and pair of decoders.

For this last task, we have incorporated a gating mechanism designed to dynamically modulate the contribution of different layers within the model, based on

their relevance to the task at hand. By adjusting the flow of information through the model, the gating mechanism allows the AdaTransformer to harmonize the learned representations across different time periods, thereby capturing the shared knowledge and reducing distribution diversities. This adaptive attention is crucial for handling the diverse and often complex patterns present in time series datasets. The gating mechanism's influence is manifested in how the model's internal representations are shaped and refined during training, contributing to the loss function's calculation to enhance overall model performance. At its core, the gating mechanism is implemented through a series of linear and batch normalization layers, each corresponding to a layer within the stack of Transformer encoders. For each layer, a gate weight is computed, which serves to dynamically adjust the influence of that particular layer's output based on its relevance to the task at hand.

The gating mechanism operates by evaluating the outputs from the Transformer encoder layers. The idea is to learn a weight vector using this network. This vector, assigns relative importance to the layers within the encoder, thereby guiding the model to emphasize certain temporal features over others. This process is design to adaptively match the distributions between different periods, ensuring that the model can generalize well across unseen data by leveraging common knowledge extracted from the learned time periods.

The gating mechanism, as explained in more detail later, is only used during the "pre-training" phase, where an estimation for the weight vector is created. At the following stage, this network remains unused, and the weight vector is updated following the adaRNN methodology.

The last part of the architecture, without including the backcasting part, can be seen at Figure 5.2.

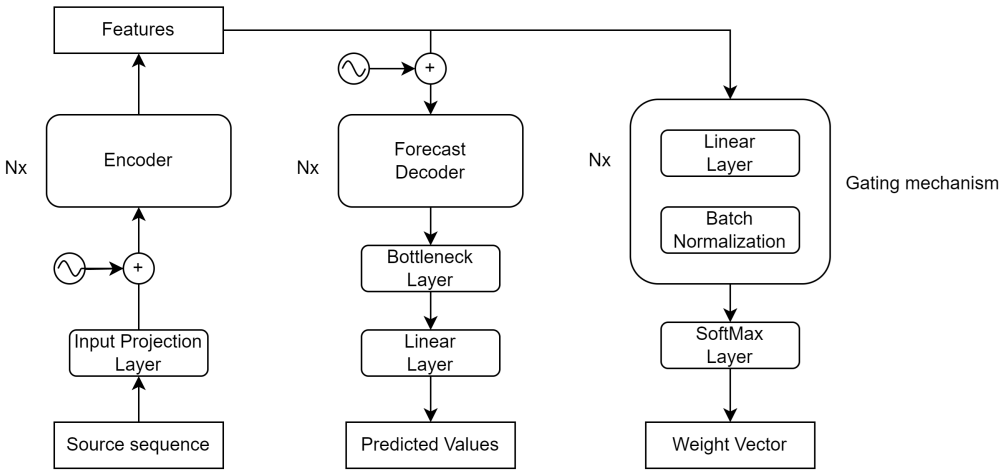


FIGURE 5.2: Overview of the architecture

### 5.2.2 Training

The training process of the AdaTransformer model represents the phase where the theoretical design and architectural innovations are put into practice. It is where the model learns to navigate the complexities of time series data, adapting its parameters to capture the underlying patterns and dynamics effectively. The training methodology employed for AdaTransformer is designed to leverage the model's unique features, such as its dual-decoder architecture for forecasting and backcasting, and the gating mechanism designed for temporal distribution matching.

In this section, we will explore the sequence of steps involved in training and how the model's distinctive components—such as the backcast decoder stack and the gating mechanism, are used in order to enhance its predictive performance. Additionally, we will discuss the strategies employed to ensure that the model not only excels in fitting the training data but also exhibits strong generalization capabilities when faced with unseen data.

We will cover the initial pre-training phase, the iterative refinement of model parameters, and the application of specialized techniques designed to optimize the model's adaptability and forecasting accuracy.

As usual, the model's training is structured around epochs. However, instead of just a standard pass across all dataset, at each epoch we perform a pass through every possible pair of segments that the TDC module produced. During each epoch, the model evaluates the forecasting loss, "transfer" loss, referring to the discrepancies between the different segments, and the backcasting loss.

The AdaTransformer model, along with its two optimizers (backcast and forecast), configured to use the Adam optimization algorithm, is initialized based on different settings. These settings include various hyperparameters such as learning rate and model architecture specifics, which are crucial for obtaining the best performance out of the model.

As said already multiple times, the model is designed to work with time series split into different segments, enabling it to learn from a diverse range of patterns and dynamics.

Throughout the training process, the model's performance is continuously monitored through the calculation of loss on the training dataset. This loss measurement guides the optimization of the model's parameters, aiming to reduce the discrepancy between the model's predictions and the actual observations. Additionally, the model is evaluated on a separate validation dataset, providing an unbiased assessment of its predictive accuracy. This evaluation step is critical for ensuring that the model generalizes well to new, unseen data.

Another important aspect of the AdaTransformer's training loop is the implementation of model checkpointing and early stopping mechanisms. Model checkpointing involves saving the model parameters whenever an improvement in validation loss is observed, ensuring that the best-performing version of the model is retained. Early stopping, on the other hand, aims to prevent overfitting by terminating the training process if the validation loss does not improve for a specified

number of consecutive epochs.

Upon completion of the training process, the model undergoes a final evaluation on a test dataset. This step is essential for assessing the model's overall performance and its ability to forecast future values accurately. The test results, along with detailed performance metrics such as L1 loss, and RMSE, provide a comprehensive overview of the model's forecasting capabilities.

In summary, the training process of the AdaTransformer model is designed to optimize its performance for time series forecasting tasks. Through careful management of model parameters, continuous performance evaluation, and strategic use of computational resources, the AdaTransformer aims to achieve high accuracy and generalization across various time series datasets.

This process can be divided into a two-phase approach: pre-training and forward boosting. This methodology ensures that the model not only learns the underlying patterns in the time series data but also adapts to the temporal dynamics and distributional characteristics across different segments or domains.

During these two parts, that modify the behavior of the temporal distribution matching, the backcasting part remains the same for both. As explained in the previous section about the architecture, the model updates the encoder and backcast decoder at each epoch.

The data is processed through the encoder, then some values are masked, and the backcast decoder tries to predict those hidden values, with that prediction, both components are updated, and then the TDM module comes into play.

Let's delve into each phase to understand their significance and execution.

**Pre-training**

The pre-training phase of the AdaTransformer model is designed with two primary objectives in mind: to learn robust network parameters capable of capturing the complex temporal dynamics within the dataset, and to fine-tune the model's sensitivity to these dynamics.

Central to this phase is the gating mechanism, the network that adjusts the model's focus and attention across different segments of the time series data. This mechanism is about ensuring that the AdaTransformer's internal representations are in harmony with the dataset's temporal structures and dynamics. By modulating the encoder outputs, the gating mechanism enables the model to tailor its processing to the unique characteristics of each data segment, whether influenced by seasonal trends, cyclic behaviors, or irregular events.

With the gating mechanism in action, the pre-training phase introduces two key loss functions: the standard forecasting loss and the transfer loss. The latter is computed as follows.

The model employs a layered approach to capture the complexity of time series data, computing the transfer loss across each layer in its stack of encoders. This involves an iterative evaluation where the model goes through each time step within

the sequence, assessing the discrepancy between different segments. The concept of "discrepancy" here refers to the variations or differences between segments of the time series data. These segments, as provided by the TDC, represent various temporal slices of the dataset. The model's task is to measure how these segments differ or resemble each other in terms of their temporal features, which is essential for learning the inherent temporal structures and dynamics within the dataset.

For each layer of the encoder and each time step, the model adjusts the window for calculating the transfer loss, focusing on the relevant segments of the time series. The measurement of discrepancy between these segments is conducted using a specialized criterion that assesses how well the model's internal representations of these segments align. This computation is then weighted by the output of the gating mechanism network for that particular layer and time step, which reflects the model's learned importance of different temporal segments.

The completion of pre-training marks the transition in the model's training lifecycle, setting the stage for the next phase, known as forward boosting.

**Forward boosting**

Forward boosting represents a shift in focus from the broad foundational learning accomplished during pre-training to a more targeted and iterative refinement process. In this phase, the model leverages the representations and insights gained from pre-training to optimize its predictions further.

A critical aspect of this transition is the utilization of the last set of weights from the gating mechanism obtained during pre-training. These weights, which have been fine-tuned to align the model's attention with the temporal segments of the data, serve as the starting point for the forward boosting phase. This ensures that the iterative refinement during forward boosting is grounded in the understanding of temporal dynamics developed during pre-training.

Unlike pre-training, where the gating mechanism plays a crucial role in adjusting the model's focus and enhancing its sensitivity to temporal dynamics, forward boosting relies on a different strategy to refine predictions and reduce errors.

During this phase, the AdaTransformer model no longer utilizes the gating mechanism to directly influence the learning process. Instead, it leverages a weight matrix that was initialized based on the outcomes of the gating mechanism during pre-training. The weight matrix serves as a representation of the insights gained from the gating mechanism, encapsulating the relative importance of different segments of the time series data as determined through the pre-training phase.

The weight matrix guides the optimization process by adjusting the emphasis placed on different parts of the data. Each element of the weight matrix corresponds to a specific layer within the stack of encoders and a particular position of the sequence being processed, reflecting the learned importance of that position for making accurate predictions.

The process of forward boosting involves, just as in pre-training, iteratively adjusting the model's parameters to minimize the transfer loss and the forecasting loss. The weight matrix plays a critical role in this process by influencing how the transfer loss is computed and applied. By adjusting the weights, the model can

focus its refinement efforts on segments of the data where the potential for improvement is greatest, thereby enhancing its overall predictive accuracy.

The shift away from the gating mechanism during forward boosting is strategic. While the gating mechanism is invaluable for establishing a good understanding of the data's temporal dynamics, forward boosting requires a more targeted approach. The weight matrix, informed by the initial insights from the gating mechanism, provides a means to apply those insights in a focused manner, optimizing the model's predictions based on a refined understanding of the data's structure.

The computation of transfer loss between each pair of segments is weighted by elements of the weight matrix, which encapsulates the relative importance of the different layers and positions as determined during the pre-training phase.

The weight matrix is dynamically adjusted based on the outcomes of the transfer loss computation. This process aims to iteratively refine the model's focus, enhancing its sensitivity to the most relevant temporal dynamics.

The model identifies positions where the discrepancy between epochs exceeds a predefined threshold, indicating potential for optimization. For these positions, the corresponding weights in the matrix are adjusted using a sigmoid function, increasing their magnitude to amplify the model's focus on these positions in subsequent iterations. To ensure stability and keep the weights within a manageable range, the updated weight matrix is normalized. This step prevents the weights from becoming excessively large, maintaining a balanced approach to model refinement.

# Chapter 6

# Methodology

In the exploration of advanced time series forecasting methodologies, particularly those involving the AdaTransformer model, it is crucial to validate and compare the proposed approaches across diverse datasets. This section delves into the experimental setup, focusing on 4 distinct datasets: the EUR/USD daily and TSLA daily exchange rate, electric power consumption and air quality. These datasets not only offer a broad spectrum of real-world applications for time series forecasting but also serve as benchmarks for assessing the effectiveness and adaptability of forecasting models.

## 6.1 Datasets

### 6.1.1 EUR/USD Daily Exchange Rate

The EUR/USD daily exchange rate dataset offers a granular view into the flux of the foreign exchange market, detailing daily open, high, low, and close (OHLC) values of the Euro against the US dollar. This dataset spans an extensive period from March 1, 2000, to November 9, 2023, providing a comprehensive historical record that captures a wide range of economic cycles and market conditions.

In preparing our dataset for analysis, we distill the OHLC data to focus on the closing prices, which reflect the market's daily valuation consensus. We further refine the data by computing returns—the percentage changes in consecutive closing prices—to address the non-stationarity typically observed in raw price levels. This conversion to returns is a foundational step that ensures the data is suitably stationary and normalized for subsequent modeling, as recommended by previous studies and best practices in financial time series forecasting.

Building upon these insights, we introduce lagged returns as features in our analysis. Specifically, we utilize a two-lagged-feature approach, a configuration suggested by the literature [2], to capture the temporal dependencies that are most relevant for prediction in financial markets. This method, underpinned by empirical evidence, holds that the previous two days' returns contain pivotal information that can be used to forecast the next day price movements.

Consequently, the AdaTransformer model is designed to leverage these two lagged returns as features. This design choice facilitates a direct comparison with an AR(2) model, which similarly utilizes the two most recent periods of data to predict the next value in the sequence.

### 6.1.2 Electric Power Consumption

The electric power consumption dataset, as referenced in the adaRNN study, comprises 2,075,260 measurements collected from 16 December 2006 to 26 November 2010, with a one-minute sampling rate. This dataset includes various attributes such as global active power, global reactive power, voltage, global intensity, and three sub-metering readings, offering a comprehensive view of household electric power consumption. After preprocessing to remove null data, the dataset consists of 2,062,346 measurements, providing a substantial volume of data for in-depth time series analysis. Additionally, the data is normalized to ensure uniformity in scale among the different variables, which is crucial for applying machine learning algorithms effectively. Normalization helps to speed up learning and leads to faster convergence during training, as well as reducing the dependency on the specific units used in measurements, thus enabling the models to treat all features with equal importance during analysis.

### 6.1.3 Tesla Stock

The Tesla stock dataset offers exploration of the company's stock performance over a decade, presenting a rich compilation of technical indicators to analyze its market behavior. Recorded daily from January 2, 2014 to December 29, 2023, the dataset encapsulates essential trading metrics such as opening, highest, lowest, and closing prices, alongside the volume of shares traded. It is augmented with momentum indicators like the 7-day and 14-day Relative Strength Index (RSI), providing insights into potential overbought or oversold conditions of the stock.

To further enhance the analysis, the dataset includes the 7-day and 14-day Commodity Channel Index (CCI), which compares the current price to an average price over a specific period to identify new trends. The inclusion of moving averages, specifically the 50-day and 100-day Simple Moving Average (SMA) and Exponential Moving Average (EMA), helps in understanding the stock's trajectory by smoothing out price data over specified time frames.

Additional technical indicators such as the Moving Average Convergence Divergence (MACD) and Bollinger Bands offer perspectives on the stock's momentum and price volatility, respectively. The dataset also features the True Range and the 7-day and 14-day Average True Range (ATR), quantifying market volatility and the degree of price movement.

### 6.1.4 Air Quality Prediction

The air-quality dataset, encompassing hourly air quality measurements from 12 monitoring stations in Beijing from March 2013 to February 2017, serves as a critical benchmark for evaluating the AdaTransformer model. This study specifically focuses on data from four stations—Dongsi, Tiantan, Nongzhanguan, and Dingling—highlighting six essential pollutants: PM2.5, PM10, SO2, NO2, CO, and O3. The selection of this dataset is strategic, aimed at providing a direct comparison with the adaRNN paper, which utilized the same dataset to demonstrate its forecasting capabilities.

To address missing data, a preprocessing step involves filling gaps with average values, ensuring a complete dataset for analysis. Further, max-min normalization standardizes the range of all features to between 0 and 1, facilitating effective model training by equalizing the influence of each feature.

## 6.2 TDC Comparison

In this section, we delve into a comparative analysis of three distinct methodologies applied in Temporal distribution characterization (TDC): the Greedy Approach, Dynamic Programming, and Simulated Annealing. The primary objective of this analysis is to determine which method most effectively maximizes dissimilarity across all segments in the time series. This comparison is essential, as having a better segmentation can potentially provide better result during inference.

Three distance metrics will be utilized to assess the quality of segmentation and the underlying temporal dependencies: Pairwise Distance, Kullback-Leibler (KL) Divergence and Cosine Distance. These metrics offer complementary perspectives on the similarity or divergence between segments, enabling a nuanced analysis of the TDC's effectiveness.

### 6.2.1 Pairwise Distance

Pairwise Distance measures the direct similarity between two segments of the time series data. It calculates the Euclidean distance between points in each segment, providing a straightforward metric of dissimilarity. A lower pairwise distance within segments indicates a high degree of similarity. Conversely, a higher pairwise distance between segments signifies distinct temporal patterns, validating the segmentation's ability to capture meaningful temporal dependencies. The simplicity and interpretability of Pairwise Distance make it an accessible metric for initial evaluations of segmentation quality. It offers a clear measure of how closely grouped the data points are within segments and how well-separated the segments are from each other.

### 6.2.2 Kullback-Leibler (KL) Divergence

KL Divergence is a measure of how one probability distribution diverges from a second, reference probability distribution. Unlike Pairwise Distance, KL Divergence is not symmetric and is particularly useful in measuring the difference in the information content between distributions. A higher KL Divergence between segments suggests that the segmentation effectively distinguishes between periods with different underlying dynamics. KL Divergence provides a more sophisticated analysis of segmentation quality by focusing on the distributional characteristics of the segments. It is particularly useful in time series data where the temporal patterns may not be immediately apparent through direct comparisons but are evident in the statistical properties of the segments.

### 6.2.3 Cosine Distance

Cosine Distance serves as a metric for evaluating the similarity between two segments of time series data, focusing on the angle between vectors representing each segment rather than their magnitude. This approach calculates the cosine of the angle between two vectors in a multidimensional space, offering insights into their directional alignment rather than their Euclidean proximity. A cosine distance close to 0 indicates a high degree of similarity, with the vectors pointing in roughly the same direction, suggesting that the segments share underlying temporal patterns. Conversely, a cosine distance closer to 1 implies a greater divergence in direction, highlighting distinct temporal behaviors between segments.

This metric is particularly valuable in time series analysis for its ability to capture the intrinsic shape and trend of data segments, independent of their scale. It emphasizes the pattern and sequence within the data, making it an effective tool for identifying segments that follow similar trends over time, even if their absolute values differ. The use of cosine distance in evaluating segmentation quality provides a deeper understanding of the temporal relationships and dynamics within the data, complementing more traditional distance measures by offering a perspective focused on pattern alignment and trend similarity.

### 6.2.4 Evaluation of TDC

The effectiveness of each TDC approach is assessed based on several criteria:

- **Forecasting Accuracy:** The first evaluation metric is the improvement in forecasting accuracy achieved by models utilizing the different approaches.
- **Computational Efficiency**: The time and resources required to perform the segmentation are considered, especially in the context of large or complex time series datasets.
- **Segmentation Quality**: The quality of the segmentation is evaluated in terms of which method can achieve the highest value for the optimization problem.

## 6.3 Forecasting Evaluation Methodology

The forecasting performance of the AdaTransformer model, equipped with the optimized Temporal Dependency Characterization (TDC) method, will be rigorously tested. The process will involve experimenting with a range of parameters to fine-tune the model for each dataset, aiming to achieve the best possible forecasting accuracy.

### 6.3.1 Testing Strategy

The testing strategy for forecasting involves several key steps, tailored to extract maximum performance from the AdaTransformer model across both datasets.

1. **Parameter Exploration**: A systematic exploration of model parameters, including the number of layers, head counts, dimensionality of feed-forward networks, dropout rates, and learning rates, will be conducted. This exploration aims to identify the optimal configuration that balances model complexity with forecasting accuracy.

2. **Segmentation Quality Integration**: The impact of segmentation quality, as determined by the chosen TDC method, on forecasting performance will be closely examined. This involves integrating the segmentation results into the model's training process, ensuring that the temporal dependencies identified during segmentation are effectively leveraged.

3. **Performance Metrics**: The model's forecasting performance will be assessed using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). These metrics provide a multi-faceted view of the model's accuracy, sensitivity to outliers, and overall error magnitude, and are the metrics used in the adaRNN study.

4. **Comparative Analysis**: The AdaTransformer model's performance will be benchmarked against the original adaRNN approach. This comparative analysis will highlight the AdaTransformer model's strengths and areas for improvement, providing valuable insights into its relative performance. The two new datasets introduced in this work-EUR/USD and TSLA-will also be compared against traditional and well-known methods.

5. **Ablation Study**: an ablation study will be conducted to meticulously assess the individual contributions of the Self-Adaptive Forecasting (SAF) component and the adaRNN component within the AdaTransformer model. This study aims to isolate and evaluate the impact of each component by systematically removing them from the model and analyzing the effects on forecasting performance. The purpose is to quantify how much each component enhances the model's accuracy and to understand their roles in improving the overall prediction capabilities of the AdaTransformer. This approach will provide clear insights into the functionality and value of each component, helping to determine their necessity and effectiveness in the context of time series forecasting.

# Chapter 7

# Results

This chapter presents the outcomes of the comparative analysis described in chapter 6. Through meticulous experimentation, this section aims to shed light on the performance and efficiency of the different parts described in the work, providing a comprehensive understanding of their implications for time series forecasting.

The results are presented through detailed tables and discussions.

## 7.1 TDC Comparison

The comparative analysis of different Temporal Dependency Characterization (TDC) methods—namely, the dynamic programming approach, the greedy approach from the adaRNN paper, and the simulated annealing—reveals insightful outcomes regarding their performance, efficiency, and segmentation quality. The dynamic programming approach, introduced in this study, stands out for its superior ability to segment time series data accurately in nearly every case, capturing meaningful temporal dependencies that significantly enhance forecasting models. However, this method's meticulous search for the near optimal segmentation comes at the cost of computational speed, making it the slowest among the tested approaches.

The simulated annealing approach emerges as a compelling alternative, striking a balance between segmentation quality and computational efficiency. It outperforms the greedy approach in terms of segmentation effectiveness, suggesting that its search strategy is more adept at uncovering useful temporal patterns within the data. Moreover, it demonstrates considerable speed, positioning it as a viable option for scenarios where both accuracy and efficiency are priorities.

The greedy approach, while being the fastest, falls short in segmentation quality compared to the other two methods. Its reliance on local optimizations limits its ability to identify the most meaningful temporal segments.

The findings from this experiment have several important implications for the development and application of time series forecasting models:

There is a trade-off between the quality of temporal segmentation and the computational resources required. The dynamic programming approach, despite its computational demands, provides high-quality segmentations. This suggests its suitability for applications where precision is mandatory, and computational resources are less of a constraint.

The simulated annealing algorithm's performance highlights its adaptability and efficiency in finding effective segmentations. Its ability to balance segmentation quality with computational speed makes it an attractive option for a wide

range of applications, especially when dealing with large datasets or requiring faster processing times.

While the greedy approach may not yield the highest quality segmentations, its speed and simplicity could still make it useful for preliminary analyses or applications where computational efficiency is the primary concern.

These results can be seen at Table 7.1, Table 7.2, Table 7.3, Table 7.4.

Building on these findings, the subsequent experiments within this study will utilize the dynamic programming approach for segmentation, with the cosine distance. Despite its slower computational speed, this method has consistently delivered superior segmentation quality, proving indispensable for capturing the most meaningful temporal dependencies in our data. This strategic choice aligns with our aim to achieve the highest possible accuracy in our time series analysis, prioritizing quality of results over computational speed.

Furthermore, it is recommended for practical applications to consider employing the simulated annealing method. Its impressive ability to strike a balance between high-quality segmentation and computational efficiency makes it a highly practical option for broader usage. This approach is especially suitable for large-scale datasets or scenarios where rapid processing is crucial, ensuring a robust performance without the extensive computational overhead associated with the dynamic programming method

TABLE 7.1: Comparison of TDC Methods Using Pairwise Distance

| **Data Set** | **Greedy Approach** | **Dynamic Programming** | **Simulated Annealing** |
|---|---|---|---|
| EUR USD Daily | 0.0063 | **0.0072** | 0.0071 |
| TSLA | 0.2562 | **0.2800** | 0.2744 |
| Electric Power | 0.0054 | **0.0058** | 0.0058 |
| Dongsi (AQ) | 0.0474 | **0.0512** | 0.0496 |
| Tiantan (AQ) | 0.0469 | 0.0496 | **0.5000** |
| Nongzhanguan (AQ) | 0.0505 | **0.0568** | 0.0532 |
| Dingling (AQ) | 0.0546 | **0.0582** | 0.058 |

TABLE 7.2: Comparison of TDC Methods Using KL-Divergence

| **Data Set** | **Greedy Approach** | **Dynamic Programming** | **Simulated Annealing** |
|---|---|---|---|
| EUR USD Daily | 0.0090 | **0.0113** | 0.1062 |
| TSLA | 1.0734 | **1.1365** | 1.1315 |
| Electric Power | 0.8330 | **0.8901** | 0.8885 |
| Dongsi (AQ) | 4.2259 | 4.5055 | **4.4620** |
| Tiantan (AQ) | 3.5643 | **3.6901** | 3.6666 |
| Nongzhanguan (AQ) | 3.9733 | **4.1046** | 4.0380 |
| Dingling (AQ) | 4.9294 | **5.1414** | 5.1400 |

TABLE 7.3: Comparison of TDC Methods Using Cosine Distance

| Data Set | Greedy Approach | Dynamic Programming | Simulated Annealing |
|---|---|---|---|
| EUR USD Daily | $2.87 \times 10^{-6}$ | $3.28 \times 10^{-6}$ | $3.08 \times 10^{-6}$ |
| TSLA | $1.47 \times 10^{-12}$ | $2.3 \times 10^{-12}$ | $1.74 \times 10^{-12}$ |
| Electric Power | $1.83 \times 10^{-4}$ | $2.48 \times 10^{-4}$ | $2.23 \times 10^{-4}$ |
| Dongsi (AQ) | 0.2120 | **0.2390** | 0.2316 |
| Tiantan (AQ) | 0.2255 | **0.2444** | 0.2350 |
| Nongzhanguan (AQ) | 0.4447 | **0.4736** | 0.4603 |
| Dingling (AQ) | 0.5511 | **0.5818** | 0.5665 |

TABLE 7.4: Comparison of TDC Efficiency in seconds(average)

| Data Set | Greedy Approach | Dynamic Programming | Simulated Annealing |
|---|---|---|---|
| EUR USD Daily | **0.01** | 188 | 9.75 |
| TSLA | **0.01** | 121 | 11.99 |
| Electric Power | **0.04** | 14400 | 28.11 |
| Dongsi (AQ) | **0.05** | 6700 | 25.01 |
| Tiantan (AQ) | **0.05** | 6655 | 25.10 |
| Nongzhanguan (AQ) | **0.05** | 6889 | 24.93 |
| Dingling (AQ) | **0.05** | 6669 | 25.52 |

## 7.2 Forecasting comparison

In this section, we present and systematically analyze the forecasting accuracy results derived from implementing the AdaTransformer model on our carefully chosen datasets: the EUR/USD daily exchange rate, electric power consumption, Tesla stock performance, and air quality indices. This analysis aims to evaluate the precision of the AdaTransformer in forecasting future values and to assess its performance in comparison with other forecasting approaches.

The presented results utilize the architecture that delivered the best overall performance. To ensure robustness, this architecture was subjected to multiple runs using different seeds to generate average outcomes. The results presented in the tables are the averages of the multiple runs.

Given the diversity of the datasets, our comparative analysis strategy is tailored to each dataset's previous examinations. For the electric power consumption and air quality datasets, we focus on a direct comparison with the adaRNN model, as these datasets have been extensively analyzed against classical forecasting methods in the adaRNN study. This allows us to specifically highlight the advancements and improvements provided by the AdaTransformer over adaRNN.

Conversely, for the EUR/USD daily exchange rate and Tesla stock datasets, which have not been previously explored in the adaRNN paper, we will compare the AdaTransformer's performance against a baseline mode along with adaRNN. These comparisons aim to establish a benchmark for the AdaTransformer's effectiveness and to offer a clear perspective on its forecasting capabilities relative to traditional methods.

Moreover, it is important to note that our objective with the AdaTransformer is to predict the next point in a sequence based on the information provided by prior data points. This predictive focus underscores the practical application of the

AdaTransformer in real-world scenarios where timely and accurate predictions are crucial.

### 7.2.1 Air Quality and Power Consumption

For the air quality forecasting at the monitoring stations (Nongzhanguan, Dongsi, Tiantan and Dingling) the AdaTransformer has consistently outperformed the adaRNN. At the Dongsi station, the AdaTransformer achieved an RMSE of 0.0239 and an MAE of 0.0173, which shows an improvement over the adaRNN's RMSE of 0.0295 and MAE of 0.0185. Similar improvements were observed at the Tiantan station, where the AdaTransformer recorded an RMSE of 0.0131 and an MAE of 0.0100, better than the adaRNN's 0.0164 RMSE and 0.0112 MAE.

The trend continues at Nongzhanguan, with the AdaTransformer achieving an RMSE of 0.0175 and an MAE of 0.0118, improving upon the adaRNN's 0.0196 and 0.0122. At Dingling, the AdaTransformer's performance surpassed adaRNN's again, evidenced by an RMSE of 0.0216 and a lower MAE of 0.0141, compared to adaRNN's RMSE of 0.0233 and MAE of 0.0150.

For electric power consumption, the AdaTransformer demonstrated another enhancement in RMSE, achieving 0.0071 compared to adaRNN's 0.077. This represents an increase in accuracy for predicting power consumption metrics. The MAE results were comparably stable, with the AdaTransformer recording a 0.0640, only marginally lower than adaRNN's 0.0644.

The results from this comparative analysis clearly demonstrate that the AdaTransformer model consistently outperforms the adaRNN model, not only in terms of RMSE but also in MAE across various datasets.

| **Dataset** | **adaRNN** | | **adaTransformer** | |
|---|---|---|---|---|
| | RMSE | MAE | RMSE | MAE |
| Dongsi | 0.0295 | 0.0185 | 0.0239 | 0.0173 |
| Tiantan | 0.0164 | 0.0112 | 0.0131 | 0.0100 |
| Nongzhanguan | 0.0196 | 0.0122 | 0.0175 | 0.0118 |
| Dingling | 0.0233 | 0.0150 | 0.0216 | 0.0141 |
| Electric Power | 0.0770 | 0.0644 | 0.0710 | 0.0633 |

TABLE 7.5: RMSE and MAE comparison for air quality and electric power consumption datasets using adaRNN and adaTransformer methods.

Building on the comparative analysis of performance metrics such as RMSE and MAE, the training time data for both the adaRNN and AdaTransformer models offers additional insights into the computational demands of these methods. As observed in the Table 7.6, the AdaTransformer consistently requires more time for training across all datasets, which highlights its greater computational complexity.

At the Dongsi station, the AdaTransformer required 113.69 seconds for training, which is significantly more than the 49.31 seconds needed by the adaRNN. This pattern is evident at other stations as well; for instance, at the Tiantan and Nongzhanguan stations, the AdaTransformer took 107.91 and 115.69 seconds respectively, compared to much shorter times by adaRNN. Even at Dingling, where

the time difference is somewhat narrower, the AdaTransformer still took nearly double the time of the adaRNN. The Electric Power dataset shows the largest absolute increase in training time, with the AdaTransformer taking 1800.11 seconds, noticeably longer than adaRNN's 1499.65 seconds.

| **Dataset** | **adaRNN** (s) | **adaTransformer** (s) |
|---|---|---|
| Dongsi | 49.31 | 113.69 |
| Tiantan | 42.80 | 107.91 |
| Nongzhanguan | 44.72 | 115.69 |
| Dingling | 39.51 | 74.44 |
| Electric Power | 1499.65 | 1800.11 |

TABLE 7.6: Training time comparison for air quality and electric power consumption datasets using adaRNN and adaTransformer methods

### 7.2.2 EurUsd and TSLA

For the two datasets EurUsd and TSLA, the AdaTransformer model consistently outperforms both the ARIMA and adaRNN models in terms of both RMSE and MAE, indicating its superior accuracy for these forecasting tasks.

For the EurUsd currency pair, the AdaTransformer achieved the lowest RMSE of 0.0045 and MAE of 0.0032. This is an improvement over the adaRNN's RMSE of 0.0049 and MAE of 0.0035, and even more significantly better than the ARIMA model's RMSE of 0.0055 and MAE of 0.0041. This showcases the AdaTransformer's enhanced accuracy in predicting currency exchange rates.

In the case of TSLA stock prices, the AdaTransformer again demonstrated superior performance. It recorded an RMSE of 0.0107 and an MAE of 0.0077, both lower than those of the adaRNN, which posted an RMSE of 0.01388 and an MAE of 0.00869. The AdaTransformer's metrics also notably outperformed the ARIMA model, which showed an RMSE of 0.02280 and an MAE of 0.01978. This considerable difference highlights the AdaTransformer's effectiveness in forecasting stock price movements.

Overall, the results from this comparative analysis clearly demonstrate that the AdaTransformer model consistently provides more accurate forecasts than the adaRNN and ARIMA models across the examined financial datasets, both in terms of RMSE and MAE.

| **Dataset** | **ARIMA** | | **adaRNN** | | **adaTransformer** | |
|---|---|---|---|---|---|---|
| | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| EurUsd | 0.0055 | 0.0041 | 0.0049 | 0.0035 | 0.0045 | 0.0032 |
| TSLA | 0.02280 | 0.01978 | 0.01388 | 0.00869 | 0.0107 | 0.0077 |

TABLE 7.7: RMSE and MAE comparison for EurUsd and TSLA datasets using ARIMA, adaRNN, and adaTransformer methods.

In continuation of the analysis between the AdaTransformer and adaRNN models, an examination of the training times for both the EurUsd and TSLA datasets

presents an interesting deviation from earlier observations. Contrary to what was noted with the air quality and electric power datasets, the training times for financial datasets reveal a slight advantage for the AdaTransformer in terms of efficiency, as Table 7.8 shows.

For the EurUsd currency pair, the AdaTransformer not only outperforms in accuracy but also in efficiency, requiring only 429.81 seconds for training, compared to 465.11 seconds with the adaRNN. This enhancement is reflective of the AdaTransformer's ability to handle complex financial data more effectively, even when reducing computational load. Similarly, in the case of the TSLA stock prices, the AdaTransformer shows a marginal improvement in training time, completing its run in 174.90 seconds, slightly less than the 178.58 seconds taken by the adaRNN.

These results are particularly significant because they highlight the AdaTransformer's dual advantage of providing more accurate forecasts and doing so in a marginally shorter time frame for financial markets. This contrasts with the earlier noted increase in computational time required for other types of datasets, suggesting that the AdaTransformer might be particularly well-optimized for financial data.

| **Dataset** | **adaRNN (s)** | **adaTransformer (s)** |
|---|---|---|
| EurUsd | 465.11 | 429.81 |
| TSLA | 178.58 | 174.90 |

TABLE 7.8: Training time comparison for EurUsd and TSLA datasets using adaRNN and adaTransformer methods.

## 7.3 Loss Analysis for Model Training

In this section, we analyze the convergence behavior of the AdaTransformer model, focusing on its performance across different stages of training. To effectively demonstrate the model's learning dynamics, we present the training and validation loss curves for two of the primary datasets previously discussed: EUR/USD daily exchange rate and one of the air quality prediction stations - Dongsi. These datasets were selected due to their contrasting natures—financial time series and environmental data—providing a comprehensive view of the model's adaptability and effectiveness in diverse settings.

For the Dongsi dataset, as we can see in Figure 7.1, the training begins with a rapid reduction in both the training and validation losses, demonstrating quick initial convergence toward the primary trends within the environmental data. This fast convergence early in the training phase underscores the model's ability to quickly adapt to complex environmental patterns. As the training continues, both loss curves exhibit a trend toward stability and closely converge, which is indicative of the model's consistent learning and effective generalization from training to validation data. The later stages show sustained convergence, confirming the model's robustness and its capability to maintain stable performance despite the variability in the data.

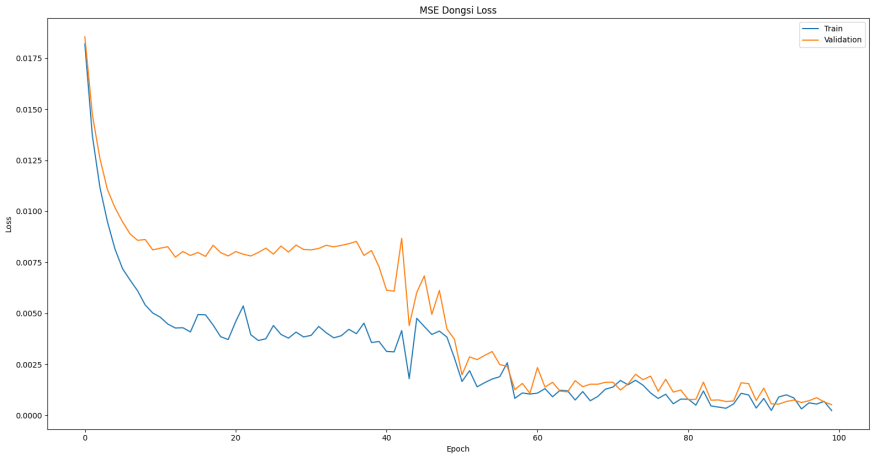


FIGURE 7.1: Dongsi Dataset Loss

Similarly, the EUR/USD dataset (Figure 7.2) shows a remarkably smooth and steady decline in both the training and validation losses from the beginning. The tight convergence of these curves throughout the training process is indicative of the model's excellent generalization capabilities, essential in the volatile environment of financial markets. This consistent convergence from an early stage highlights the model's efficiency in handling the inherent noise and dynamic changes typical of financial time series data.

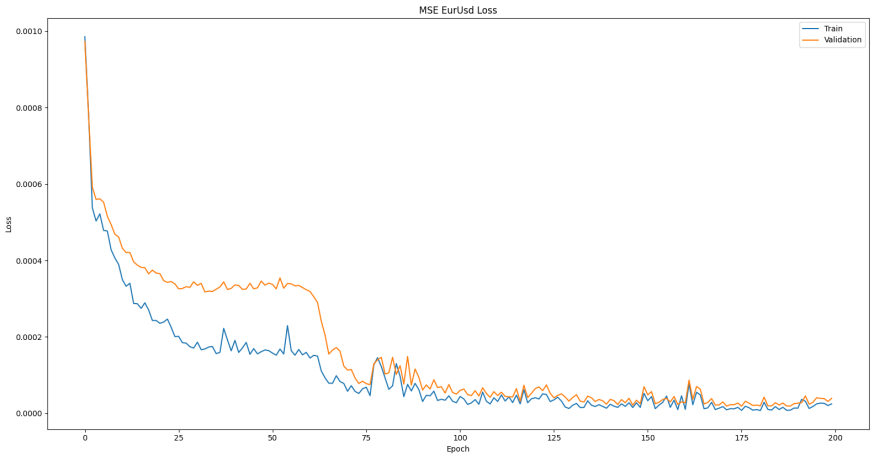


FIGURE 7.2: EurUsd Dataset Loss

Comparing the convergence behaviors across these datasets, both models demonstrate robust learning capabilities, as shown by the significant decrease in loss values. The Dongsi dataset, after adjusting from initial fluctuations, shows notable

convergence stability, effectively capturing the complexities of environmental factors. In contrast, the EUR/USD model illustrates superior and consistent convergence right from the beginning, making it particularly suitable for financial forecasting where adaptability to rapid market changes is critical.

These findings validate the AdaTransformer's capacity for achieving stable and effective learning across diverse application domains, supporting its use in tasks that require high accuracy and reliable prediction capabilities in both environmental monitoring and financial forecasting.

## 7.4 Impact of TDC on forecasting accuracy

In this section, we explore the influence of Temporal Dependency Characterization (TDC) quality on the forecasting accuracy of the AdaTransformer model. Specifically, we compare the performance of the AdaTransformer equipped with an advanced TDC algorithm against the same model using the Greedy TDC method from the adaRNN study. This analysis aims to determine whether a more sophisticated segmentation approach enhances the model's predictive capabilities. By examining the differences in forecasting outcomes between the two TDC methods, we seek to establish a clear link between improved segmentation techniques and their impact on the effectiveness of time series forecasting. This comparative approach provides valuable insights into the potential benefits of refining TDC methodologies within advanced forecasting models.

The analysis of the Dynamic Programming (DP) TDC versus the Greedy TDC method provides critical insights into the AdaTransformer's forecasting capabilities. While the advanced TDC method does exhibit improvements in RMSE and MAE across various datasets, these enhancements are notably modest. This finding suggests that while the sophisticated segmentation techniques embedded within the DP method can refine the AdaTransformer's predictive accuracy, the extent of improvement is less pronounced than one might anticipate.

These incremental gains, though modest, are consistent across different datasets, indicating that the advanced TDC method does contribute positively to the model's forecasting precision. However, the slight nature of these improvements points to the need for further refinement of the TDC techniques. It appears that the advanced method, despite its theoretical advantages, does not drastically outperform the simpler Greedy TDC in practical applications. This outcome underscores the complexity of forecasting tasks and the challenges inherent in significantly boosting performance through segmentation techniques alone.

Furthermore, this observation invites a critical examination of the cost-benefit ratio of employing more complex TDC methods. Given the limited improvement, the additional computational complexity and resource demands of the DP TDC might not always be justified. This realization highlights the importance of continuing to refine and perhaps simplify these methods to enhance their practical utility without overly complicating the model architecture.

| **Dataset** | **Greedy TDC** | | **DP TDC** | |
|---|---|---|---|---|
| | RMSE | MAE | RMSE | MAE |
| Dongsi | 0.0240 | 0.0172 | 0.0239 | 0.0173 |
| Tiantan | 0.0135 | 0.0105 | 0.0131 | 0.0100 |
| Nongzhanguan | 0.0180 | 0.0120 | 0.0175 | 0.0118 |
| Dingling | 0.0220 | 0.0149 | 0.0216 | 0.0141 |
| Electric Power | 0.072 | 0.0639 | 0.0710 | 0.0633 |
| EUR/USD | 0.0049 | 0.0035 | 0.0045 | 0.0032 |
| TSLA | 0.0111 | 0.0080 | 0.0107 | 0.0077 |

TABLE 7.9: Comparison of RMSE and MAE between Greedy TDC and DP TDC for air quality, electric power consumption, EUR/USD, and TSLA datasets.

## 7.5 Ablation Study

This section presents an ablation study designed to isolate and evaluate the individual contributions of the SAF module and the adaRNN component to the overall forecasting accuracy of our model. By systematically removing each component while keeping the rest of the model architecture intact, we aim to discern the specific impact and utility of SAF and adaRNN within the AdaTransformer framework. This study will enable us to understand how each component influences the model's ability to capture and process temporal dependencies, thereby shedding light on their respective roles in enhancing forecasting performance.

| **Dataset** | **adaTransformer** | | **SAF Removed** | | **adaRNN Removed** | |
|---|---|---|---|---|---|---|
| | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| Dongsi | 0.0239 | 0.0173 | 0.02592 | 0.01838 | 0.0339 | 0.0220 |
| Tiantan | 0.0131 | 0.0100 | 0.0149 | 0.0196 | 0.0233 | 0.0164 |
| Nongzhanguan | 0.0175 | 0.0118 | 0.01802 | 0.01267 | 0.0226 | 0.0181 |
| Dingling | 0.0216 | 0.0141 | 0.02201 | 0.01441 | 0.0263 | 0.0163 |
| Electric Power | 0.0710 | 0.0633 | 0.0733 | 0.0648 | 0.0790 | 0.0666 |
| EUR/USD | 0.0045 | 0.0032 | 0.0049 | 0.0034 | 0.0055 | 0.0038 |
| TSLA | 0.0107 | 0.0077 | 0.0115 | 0.0091 | 0.01366 | 0.0100 |

TABLE 7.10: Ablation study comparing RMSE and MAE for adaTransformer with SAF and adaRNN components removed, across air quality, electric power consumption, EUR/USD, and TSLA datasets.

The results from the ablation study of the AdaTransformer model across various datasets show a consistent pattern: the complete AdaTransformer model, with all features enabled, consistently achieves the lowest RMSE and MAE values. This indicates that the full model—with the SAF module and adaRNN components—has superior forecasting accuracy compared to the versions with either the SAF module or adaRNN components removed.

The increase in RMSE and MAE when the SAF is removed suggests that the self-adaptive mechanisms are playing a significant role in capturing the temporal

dependencies within the data. It appears that the SAF module is effective at enhancing the model's ability to understand and process the input sequences, leading to more accurate forecasts.

Further removal of the adaRNN components results in an even more pronounced increase in RMSE and MAE. This indicates that the adaptive aspects of the model, including its ability to adjust to varying distributions, are crucial for optimal performance. Without these components, the model's predictive power diminishes notably, which underscores their importance in the AdaTransformer architecture.

The results highlight the effectiveness of the full AdaTransformer model's architecture. The ablation study confirms that both the SAF module and the adaRNN components contribute significantly to the model's overall performance. Removing either element leads to a clear degradation in forecasting accuracy, emphasizing the value of the complete AdaTransformer model for complex time series prediction tasks.

## Chapter 8

# Conclusions and Future Work

This thesis examined the forecasting performance of the AdaTransformer model in contrast to the adaRNN model across multiple datasets, including air quality indices, electric power consumption, and financial markets represented by EUR/USD and TSLA datasets. The AdaTransformer demonstrated superior forecasting accuracy, evidenced by lower RMSE and MAE scores, suggesting its enhanced ability to model complex temporal dependencies.

An important observation from this study is the variability in training time between the AdaTransformer and adaRNN models, depending on the nature of the dataset. In some cases, such as financial markets, the AdaTransformer not only provided better accuracy but also required less training time than the adaRNN, highlighting its potential for efficient real-time forecasting. However, for other datasets, particularly those involving environmental and energy consumption metrics, the AdaTransformer required longer training times. This underscores a trade-off between computational demand and accuracy, where the AdaTransformer's increased complexity, which facilitates its superior performance, also leads to longer training durations.

The findings from this study underscore the potential of advanced transformer-based models in time series analysis, particularly in settings traditionally dominated by RNNs. By integrating sophisticated techniques from adaRNN and Self-Adaptive Forecasting (SAF) into the AdaTransformer, this research illustrates the importance of employing adaptive mechanisms that enhance the model's ability to accurately forecast across periods with varying underlying distributions. This approach not only improves forecasting accuracy but also ensures the model remains robust under different economic and environmental conditions, showcasing its adaptability in both training and real-time prediction scenarios.

The ablation study conducted as part of this research further solidifies the importance of both the adaRNN and Self-Adaptive Forecasting (SAF) components within the AdaTransformer model. The study's results clearly demonstrate that each component contributes uniquely to enhancing the model's performance. Removing either component leads to a noticeable decline in forecasting accuracy, underscoring their individual usefulness. This finding confirms that both the adaRNN and SAF components are integral to the model's ability to handle diverse and dynamic data environments effectively.

Despite the promising results achieved by the AdaTransformer model, the expected significant enhancements from the advanced Temporal Dependency Characterization (TDC) techniques, such as the Dynamic Programming approach, did not materialize as anticipated. The modest improvements observed highlight a key challenge: maintaining a balance between the sophistication of advanced modeling techniques and their practical utility in forecasting. It is crucial to ensure that the complexity added by new techniques leads to meaningful improvements in real-world forecasting accuracy, justifying the additional computational effort involved.

Building upon the comprehensive insights obtained from this thesis, future research can strategically address the identified gaps and further capitalize on the potential of advanced transformer-based models for time series analysis. The modest incremental gains realized through Temporal Dependency Characterization (TDC) techniques like the Dynamic Programming approach have illuminated a critical challenge: optimizing the balance between model complexity and practical utility in forecasting.

To move forward effectively, future investigations could delve into developing more computationally efficient TDC algorithms. This development should aim not only to retain the accuracy benefits of complex methods but also to reduce computational demands, making these models more feasible for larger datasets or real-time applications. Exploring approximations of current algorithms or entirely new adaptive methods could offer promising results, enhancing the model's applicability without sacrificing performance.

Additionally, there is an opportunity to refine the AdaTransformer's architecture further. Experimenting with different configurations and integrating novel neural network components such as convolutional or additional layers might uncover new ways to capture spatial-temporal patterns and long-term dependencies more effectively. Such architectural innovations could boost the model's forecasting capabilities across varied economic and environmental scenarios.

In conclusion, this thesis not only advances our understanding of transformer-based models in time series forecasting, but also sets the stage for future innovations in the field. The AdaTransformer model, with its robust performance across diverse datasets, represents a significant step forward in our pursuit of more accurate and reliable forecasting methods.

# Bibliography


[1] Sercan O Arik, Nathanael C Yoder, and Tomas Pfister. Self-adaptive forecasting for improved deep learning on non-stationary time-series. *arXiv preprint arXiv:2202.02403*, 2022.

[2] Argimiro. Arratia. *Computational finance : an introductory course with R*. Atlantis studies in computational finance and financial engineering ; 1. Atlantis Press, Paris, 2014.

[3] Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29, 06 2014.

[4] Alexander Bartler, Andre Bühler, Felix Wiewel, Mario Döbler, and Bin Yang. Mt3: Meta test-time training for self-supervised test-time adaption, 2022.

[5] Sofiane Brahim-Belhouari and Amine Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics Data Analysis*, 47(4):705–712, 2004.

[6] Lijuan Cao and Qingming Gu. Dynamic support vector machines for nonstationary time series forecasting. *Intell. Data Anal.*, 6:67–83, 04 2002.

[7] Mayee Chen, Karan Goel, Nimit S. Sohoni, Fait Poms, Kayvon Fatahalian, and Christopher Ré. Mandoline: Model evaluation under distribution shift, 2022.

[8] R. Dahlhaus. Fitting time series models to nonstationary processes. *The Annals of Statistics*, 25(1):1 – 37, 1997.

[9] Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, Renjun Xu, and Chongjun Wang. Adarnn: Adaptive learning and forecasting of time series. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 402–411, 2021.

[10] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *CoRR*, abs/1704.04110, 2017.

[11] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks, 2016.

[12] Vincent Le Guen and Nicolas Thome. Probabilistic time series forecasting with structured shape and temporal diversity, 2021.

[13] Tomasz Górecki and Maciej Łuczak. Multivariate time series classification with parametric derivative dynamic time warping. *Expert Systems with Applications*, 42(5):2305–2312, 2015.

[14] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment, 2021.

[15] Eamonn Keogh and Chotirat Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7:358–386, 01 2005.

[16] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts, 2021.

[17] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. *CoRR*, abs/1703.07015, 2017.

[18] Vincent LE GUEN and Nicolas THOME. Shape and time distortion loss for training deep time series forecasting models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[19] C. N. Ng and P. C. Young. Recursive estimation and forecasting of non-stationary time series. *Journal of Forecasting*, 9(2):173–204, 1990.

[20] Carlotta Orsenigo and Carlo Vercellis. Combining discrete svm and fixed cardinality warping distances for multivariate time series classification. *Pattern Recognition*, 43:3787–3794, 11 2010.

[21] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D Sculley, Sebastian Nowozin, Joshua V. Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift, 2019.

[22] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *CoRR*, abs/1704.02971, 2017.

[23] Patrick Schäfer. Scalable time series classification. *Data Min. Knowl. Discov.*, 30(5):1273–1298, sep 2016.

[24] Rajat Sen, Hsiang-Fu Yu, and Inderjit Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting, 2019.

[25] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

[26] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020.

[27] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[28] 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.

[29] Jindong Wang, Yiqiang Chen, Wenjie Feng, Han Yu, Meiyu Huang, and Qiang Yang. Transfer learning with dynamic distribution adaptation. *ACM Transactions on Intelligent Systems and Technology*, 11(1):1–25, February 2020.

[30] Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S. Yu. Visual domain adaptation with manifold embedded distribution alignment, 2018.

[31] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip S. Yu. Generalizing to unseen domains: A survey on domain generalization, 2022.

[32] Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep subdomain adaptation network for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 32(4):1713–1722, 2021.