applied sciences

Article

Revolutionizing Time Series Data Preprocessing with a Novel Cycling Layer in Self-Attention Mechanisms [†]

Jiyan Chen and Zijiang Yang *

School of Information Technology, York University, North York, ON M3J 1P3, Canada; jiyanche@my.yorku.ca
* Correspondence: zyang@yorku.ca

† This work is an extend version of a paper published in the proceeding of the 2024 16th International Conference on Intelligent Human Machine Systems and Cybernetics (IHMSC 2024), held in Hangzhou, China, 24–25 August 2024.

Abstract: This paper introduces an innovative method for enhancing time series data preprocessing by integrating a cycling layer into a self-attention mechanism. Traditional approaches often fail to capture the cyclical patterns inherent to time series data, which affects the predictive model accuracy. The proposed method aims to improve models' ability to identify and leverage these cyclical patterns, as demonstrated using the Jena Climate dataset from the Max Planck Institute for Biogeochemistry. Empirical results show that the proposed method enhances forecast accuracy and speeds up model fitting compared to the conventional techniques. This paper contributes to the field of time series analysis by providing a more effective preprocessing approach.

Keywords: information technology; machine learning; autoencoder; data science; data preprocessing; attention model; unsupervised learning; weather prediction; regression model

1. Introduction

Time series analysis is a fundamental aspect of numerous scientific disciplines, including economics and engineering. At its core, a time series is a sequence of data points, typically comprising successive measurements made over a specific time interval. The importance of time series data lies in its ability to predict future phenomena based on the correlation between previous observations and future values. This forecasting process is not merely an extrapolation; it involves a comprehensive understanding of intricate patterns, trends, and seasonality within the data. For instance, in economics, time series analysis facilitates distinguishing long-term trends from short-term variations, ensuring that economic policies are not misguided by transient fluctuations [1].

Nevertheless, time series analysis presents several challenges. The data can exhibit diverse patterns, including seasonal, cyclical, and chaotic behavior patterns. Furthermore, the data's non-stationarity can lead to unreliable forecasting outcomes if not properly addressed. In addition to these analytical challenges, external factors can introduce noise into data patterns, complicating the prediction process. Thus, the rigorous preprocessing of time series data is imperative for effective analysis.

In our previous work [2], we explored the use of positional self-attention mechanisms integrated into autoencoders for preprocessing time series data. This study demonstrated that positional encodings could effectively capture short-term temporal dependencies, leading to significant improvements in model performance over traditional statistical preprocessing methods. The key findings highlighted a reduction in the Mean Squared Error (MSE), Mean Absolute Error (MAE), and other performance metrics across various regression models, including Linear Regression, Support Vector Regression (SVR), and K-Nearest Neighbors (KNN). However, the previous study also identified several limitations. The positional self-attention mechanism, while effective for short-term dependencies,

check for updates

Citation: Chen, J.; Yang, Z. Revolutionizing Time Series Data Preprocessing with a Novel Cycling Layer in Self-Attention Mechanisms. *Appl. Sci.* **2024**, *14*, 8922. https://doi.org/10.3390/app14198922

Academic Editors: Yujin Lim and Hideyuki Takahashi

Received: 31 August 2024
Revised: 28 September 2024
Accepted: 30 September 2024
Published: 3 October 2024

struggled to adequately capture longer-term cyclical patterns in the data [3]. Additionally, the method exhibited sensitivity to the choice of hyperparameters and showed limited improvements in datasets with pronounced cyclical behaviors, such as seasonal or periodic time series.

Our paper underscores the importance of addressing and mitigating various issues that arise from improper data preprocessing methods. Capturing intricate temporal dependencies within data that exhibit periodic features is a significant challenge in data analysis, particularly in the context of time series and spatiotemporal data. A primary difficulty lies in the tendency of most existing approaches to overlook the long-term dependencies inherent to the data. These approaches often fail to consider the comprehensive mechanisms required to model these long-term dependency issues [4]. Alternatively, they employ batch techniques that fragment long sequences into numerous short ones, which inadequately capture the dependencies between sequences, a problem known as sequence fragmentation [5]. Additionally, the shifting of long-term periodic dependencies needs to be addressed in current studies [6].

The objective of this paper is to capture periodic features effectively using weather coefficient data. Addressing these challenges necessitates innovative techniques specifically designed for time series data. This paper proposes a pioneering approach to tackle the complexities of preprocessing time series data. The central aspect of this approach involves replacing traditional positional encoding techniques with a personalized cycling technique integrated into self-attention mechanisms. The significance of this issue is further highlighted by existing research that emphasizes the limitations of positional models in accurately capturing these dependencies. Conventional approaches often neglect long-term dependencies, resulting in fragmented data sequences that fail to accurately reflect the underlying temporal dynamics.

The significance of this paper lies in its innovative approach to preprocessing strongly time-correlated data, such as the weather data utilized in experiments. Firstly, this study contributes to the time efficiency of data processing and the speed of regression in a time series analysis. This advantage is evident in the faster regression capabilities, allowing the model to achieve desired loss scores in significantly less time compared to traditional methods. Secondly, this paper enhances the accuracy and reduces the error of data predictions in time series analysis. This improvement is demonstrated by lower scores in squared error loss functions when employing regression model forecasting tasks, a standard metric for assessing performance. The Cycling technique achieves lower function scores, indicating a more accurate representation of the data. Finally, this study highlights the adaptability and customizability of time series analysis. Traditional preprocessing techniques often provide a one-size-fits-all solution, which may not be optimal for specific analytical tasks. In contrast, the Cycling Layers can be tailored to capture the regular features of different time series data by adjusting the cycling period. Furthermore, the architecture of the Cycling Self-Attention Layer allows for customization, enabling users to modify and train models specific to their objectives.

This paper aims to (1) evaluate the effectiveness of different preprocessing techniques, including positional self-attention mechanisms and traditional statistical methods, in enhancing the performance of deep learning models for time series data, and (2) compare the performance of these methods, particularly in tasks related to anomaly detection and forecasting. Section 2 reviews the related work about the dataset and presents the theoretical framework for integrating positional self-attention mechanisms into autoencoders. Section 3 describes the methodologies employed, including the specific models and evaluation metrics used. Section 4 discusses the dataset used and the results of the research, comparing the performance of the positional self-attention method with traditional preprocessing techniques. Section 5 concludes the paper by summarizing the findings and discussing the implications for future research and potential applications.

2. Literature Review

Traditional methods for time series data processing have relied heavily on statistical and digital signal-processing techniques [7]. Statistical approaches, including autoregressive models [8] and integrated moving average models [9], have provided a foundational framework for analyzing linear dependencies within data. Concurrently, digital signal-processing techniques, such as the Fourier transform, have facilitated trend decomposition and offered insights into the underlying patterns of time series data [10–12]. As technology advanced, the focus shifted towards more nuanced analyses, incorporating non-linear dynamics [13] and time-frequency analysis [14]. Bayesian methods [15] have emerged as powerful tools, incorporating prior knowledge and uncertainty into models, thereby providing robust analyses capable of handling non-Gaussian distributions.

In parallel, researchers have addressed challenges in data preprocessing, focusing on mitigating issues, such as noise reduction and feature selection arising from low-quality data [16,17]. Novel methodologies, such as those proposed by Cortés-Ibáñez et al. [18], and the use of positional self-attention autoencoders by Chen and Yang [2], have underscored the critical importance of preprocessing in enhancing the performance of time series analyses.

The limitations of traditional approaches have catalyzed the development of advanced techniques, especially within the domain of machine learning. Recurrent Neural Networks (RNNs) marked a significant breakthrough in time series analysis due to their capacity to process sequences and retain past information [19]. To further enhance the performance of RNNs, Long Short-Term Memory (LSTM) units [20] and Gated Recurrent Units (GRUs) [21] were introduced. These advancements addressed issues, such as vanishing and exploding gradients, thereby enhancing the networks' efficiency. Additionally, Convolutional Neural Networks (CNNs) have proven to be powerful tools in time series analysis, excelling in recognizing local and global patterns within data [22]. The integration of CNNs with other methods, such as exponential smoothing, has led to hybrid models that offer improved forecasting accuracy.

Recent research has focused on leveraging neural network technologies for preprocessing time series data. Pre-trained LSTM-based stacked autoencoders [23] and ensembles of recurrent autoencoders [24] have shown promise in outlier detection and data denoising, highlighting the potential of neural networks for improving data quality for subsequent analyses.

Overall, while statistical approaches, like ARIMA, have provided foundational insights, their limitations have driven the adoption of neural networks, such as RNNs, LSTMs, and CNNs, to better capture complex temporal patterns. Recent studies emphasize the complementary strengths of combining statistical and machine learning methods, as seen in the application of these techniques to the Jena Climate dataset. This ongoing evolution underscores the need for continued innovation in time series analysis.

The analysis of the Jena Climate dataset by Li [25] involved a detailed Exploratory Data Analysis (EDA) to understand the dataset's characteristics, followed by the application of an ARIMA model for time series forecasting. This approach predicts future climate conditions based on historical data, which helps to reveal patterns, trends, and correlations in climate variables, such as temperature and humidity.

Moreover, the ongoing debate between statistical and machine-learning approaches highlights the dynamic nature of this field, underscoring the necessity for continued research and innovation. The "Jena Climate Prediction with LSTM" analyzed by Qin [26] and the "Daily Forecasting LSTM-FB Prophet" analyzed by Yacoub [27] both utilized advanced machine learning techniques for climate data forecasting. Qin's approach focused on employing Long Short-Term Memory (LSTM) networks. On the other hand, Yacoub's method combined the deep learning capabilities of LSTMs with the robust, intuitive features of Facebook's Prophet tool. Aiming to enhance the precision of daily climate predictions. This dual-model methodology in Yacoub's analysis was particularly adept at capturing complex temporal patterns and adapting to seasonal variations and irregularities.

Meanwhile, the "Tensorflow 3-RNN" analyzed by Shen [28] employed advanced Recurrent Neural Network (RNN) techniques within TensorFlow to analyze the Jena Climate dataset. This approach focused on leveraging RNNs to capture complex temporal relationships in climate data, such as temperature and humidity. The goal is to forecast future climate conditions or to extract meaningful insights from the data's temporal patterns accurately. It prioritizes model optimization, training, and evaluation to ensure accurate and reliable predictions.

Furthermore, Muhammad [29] focused on the meticulous preparation and exploration of the Jena Climate dataset, emphasizing data wrangling techniques crucial for time series analysis. This process addressed challenges, such as missing values, data normalization, and outlier handling. Exploratory data analysis was then conducted to reveal trends, seasonality, and patterns in climate variables. By leveraging Python libraries for data manipulation and visualization, the study aimed to establish a solid foundation for more complex time series forecasting, thereby providing insightful and reliable interpretations of climatic trends and behaviors.

Despite these advancements, challenges persist in processing time series data, particularly in these data-type-related real-time scenarios. Robust and scalable streaming frameworks are essential to handle the continuous influx of data efficiently.

3. Methodology and Materials

3.1. Autoencoder Methodology

Auto-encoders (AEs), as outlined in Goodfellow et al. [30], are a fundamental technique in unsupervised learning, utilizing neural networks to learn data representations. These tools are particularly adept at managing high-dimensional data, focusing on dimensionality reduction to effectively represent datasets. Auto-encoders also have extensive applications in unsupervised learning tasks, including dimensionality reduction, feature extraction, efficient coding, generative modeling, de-noising, and anomaly detection, as discussed by Zhang et al. [31]. They are fundamentally similar to Principal Component Analysis (PCA), particularly in terms of reducing the dimensionality of large datasets, a similarity most apparent when comparing PCA to a single-layered auto-encoder with a linear activation function, as noted by Sarker et al. [32].

For our research experiment, the cycling layer is incorporated within an autoencoder, an artificial neural network designed to learn efficient representations of input data. This autoencoder compresses the input data into a compact representation, which is then decoded back to its original form. By integrating the cycling layer and self-attention mechanism into the encoding process, the model is equipped to effectively capture and represent the cyclical patterns present in the time series data.

Autoencoders play a crucial role in time series data preprocessing by performing noise reduction and dimensionality reduction. They effectively filter out noise while preserving essential signals, enhancing data quality for analysis. Additionally, autoencoders address the challenge of high-dimensional time series data by reducing dimensionality, which improves computational efficiency and mitigates the risk of overfitting. This is particularly valuable in multivariate time series data, where complex variable interactions are present. By encoding these interactions, autoencoders reveal the underlying structure, providing a more refined input for further analysis.

However, they exhibit several limitations when applied to time series data. Single autoencoders only compress all input data points into a fixed-size latent representation, which often leads to the loss of crucial long-term dependencies. Moreover, they are not designed to handle cyclical patterns effectively, treating sequences as linear inputs without regard to periodicity. This limitation makes them less suitable for datasets where capturing cycles, such as daily or seasonal variations, is essential. Additionally, autoencoders do not distinguish between important and less relevant data points within a sequence, leading to a loss of focus on critical time steps. These drawbacks motivate the need for additional mechanisms to enhance their capabilities in time series analysis.

3.2. Applying with Attention Mechanisms

In 2014, Bahdanau et al. [33] introduced the attention mechanism to overcome the limitations of traditional sequence-to-sequence models, particularly for long sequences. This mechanism enabled the model to dynamically focus on different parts of the input sequence while generating the output. However, due to their sequential nature, recurrent models inherently limit parallel processing within a single training instance [34]. This limitation is especially problematic for longer sequences, where memory constraints hinder the batching of multiple examples [35]. The introduction of attention mechanisms [36] revolutionized sequence modeling by enabling the capture of dependencies regardless of the distance between sequence elements. However, it is noteworthy that, with few exceptions [37], these attention mechanisms were predominantly integrated with recurrent networks. Below will explore how self-attention, when combined with a positional layer and novel cycling layer, can enhance the model's ability to process and analyze sequential data.

3.2.1. Applying Self-Attention with a Positional Layer

In models such as Transformers, the self-attention mechanism includes multiple attention heads within each self-attention sublayer. For an input sequence, each attention head processes an input sequence, $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, consisting of n elements, where $x_i \in \mathbb{R}^{d_x}$, and computes a new sequence $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n)$ of the same length, where $z_i \in \mathbb{R}^{d_z}$. Each output element, z_i, is determined by a weighted sum of the input elements that have been linearly transformed (Equation (1)).

$$z_i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j W^V \right) \tag{1}$$

The weight coefficient α_{ij} for the contribution of input element x_i to output element z_i is computed using a SoftMax function (Equation (2)).

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}} \tag{2}$$

The SoftMax function (Equation (2)) is applied to a set of scores that measure the compatibility or similarity between x_i and x_j. These scores are typically computed using a trainable linear transformation of the input elements, involving dot products among the query, key, and value vectors derived from the input elements. Finally, the output of each attention head, represented by different z sequences, is concatenated and linearly transformed through a fully connected layer.

Algorithm 1 outlines the function Self_Attention, which takes three input matrices: Q (*queries*), K (*keys*), and V (*values*). The function initializes three empty lists: *attention_scores*, *attention_weights*, and *output*. For each query in Q, the dot product is computed with each key in K to generate attention scores. These scores are normalized using the SoftMax function to obtain the attention weights. Each value in V is then multiplied by its corresponding attention weight, and the results are aggregated into the output list.

In an additional approach, Vaswani et al. [38] introduced positional encodings before self-attention, allowing the model to capture the order of the sequence.

Incorporating positional layer self-attention into data preprocessing primarily aims to leverage its capability to effectively understand and encode the positional relationships within sequential data. Traditional methods of sequence data processing often depend on manual feature engineering or simplistic assumptions about sequences, which can result in a significant loss of context and information. In contrast, positional layer self-attention automates the recognition of complex dependencies, thus avoiding the pitfalls of traditional approaches. In positional layer self-attention, data preprocessing enhances the performance of self-attention mechanisms by ensuring that the temporal or sequential relationships inherent in the data are not only preserved but are also highlighted and actively utilized for analysis or predictions.

Algorithm 1: Self_Attention

```
1   Input: Q, K, V;
2   Output: output list S
3   Initialized empty lists: attention_scores, attention_weights, output
4   for each query in Q do
5     for each key in K do
6       Compute attention score as dot product of query and key
7       Add attention score to attention_scores list
8     end for
9     for each score in attention_scores do
10      Compute attention weight as SoftMax of score
11      Add attention weight to attention_weights list
12    end for
13    for each value in V do
14      Multiply value by corresponding attention weight
15      Add result to output list
16    end for
17  end for
```

Moving to the method by which a positional self-attention system addresses experimental challenges by incorporating positional information, consider that the input representation $x_i \in \mathbb{R}^{n\times d}$ contains the d-dimensional embeddings for n tokens of a sequence. The positional encoding outputs $X + P$ using a positional embedding matrix $P \in \mathbb{R}^{n\times d}$ of the same shape. The element in the i th row and the $(2j)^{th}$ (Equation (3)) or the $(2j+1)^{th}$ (Equation (4)) column is as follows:

$$p_{i,2j} = \sin\left(\frac{i}{10,000^{2j/d}}\right) \tag{3}$$

$$p_{i,2j+1} = cos\left(\frac{i}{10,000^{2j/d}}\right) \tag{4}$$

This process ensures that the model understands the content of each element and its position within the sequence.

Algorithm 2 outlines a function that takes the *sequence_length* and the number of dimensions as input. It initializes a matrix *pos_enc* of size *sequence_length* × *dimensions* filled with zeros. The function then iterates over each position in the sequence and each dimension in the embedding space. For each pair of positions and dimensions, a value is computed using either the sine or cosine function, and these values are assigned to the corresponding elements in the *pos_enc* matrix. Finally, the function returns the *pos_enc* matrix, which represents the positional encodings.

However, it is crucial to recognize the potential pitfalls associated with the intricate dynamics of positional self-attention. Its comprehensive consideration of all sequence attributes can present challenges, mainly when dealing with time series data. Many researchers have focused on addressing these issues.

By incorporating self-attention mechanisms, we address several limitations of traditional autoencoders in a time series analysis. Self-attention improves the model's ability to capture long-term dependencies by dynamically computing the relationships between all elements in a sequence, regardless of their distance from one another. This allows the model to focus on the most relevant time steps, enhancing interpretability by highlighting which data points contribute most to the prediction. Additionally, self-attention provides greater flexibility in handling variable-length sequences, avoiding the need for fixed-size inputs and allowing for better generalization across different datasets.

Algorithm 2: Positional_Encoding

```
1   **Input:** sequence_length, dimensions;
2   **Output:** pos_enc
3   Initialize pos_enc as a matrix of size sequence_length $\times$ dimensions with all zeros
4   for pos from 0 to sequence_length $-1$ do
5     for I from 0 to dimensions $-1$ in steps of 2 do
6       Pos_enc[pos,i] = $\sin\left(\frac{pos}{10000^{\frac{2i}{dimensions}}}\right)$
7       if i + 1 < dimensions then
8         Pos_enc[pos,i+1] = $cos\left(\frac{pos}{10000^{\frac{2i}{dimensions}}}\right)$
9       end if
10    end for
11  end for
```

3.2.2. Applying Self-Attention with Novel Cycling Layer

However, while self-attention mechanisms offer significant advantages, they still rely on positional encodings to maintain the order of the elements in a sequence. These positional encodings are designed to capture the linear order of data points but do not inherently account for cyclical patterns, such as daily, weekly, or seasonal cycles, which are crucial in many real-world time series datasets. Moreover, the quadratic complexity of self-attention can make it computationally expensive for very long sequences, limiting its efficiency in practical applications.

In light of this, we advocate for adopting cycling encoding as a substitute for positional self-attention. This method refines the attention's focus based on the temporal characteristics of the target data, ensuring a more nuanced and contextually relevant representation.

Based on our analysis of weather data with seasonal features incorporated into the positional layer Figure 1, we observed inconsistencies in the step length of the positional layer. While a straightforward numerical step relationship may be effective for dynamic features, such as language, it is less suitable for capturing the temporal characteristics of time series data. To address this issue, we have developed a cycling layer. We anticipate that our cycling layer, with its consistent step length, will more accurately capture the seasonal features inherent to time series data.

The advent of the cycling layer self-attention signifies a pivotal advancement in unsupervised machine learning, particularly in preprocessing time series data. This innovative architectural layer is motivated by the intrinsic need to capture the cyclical nature of temporal data more effectively. Traditional layers, such as positional encodings, handle sequential information but often need more capability to grasp cyclical patterns critical in datasets, such as weather patterns, where features fluctuate periodically.

The cycling layer is specifically designed to address this gap. Unlike the one-size-fits-all approach of traditional positional encoding, it provides a framework to isolate and understand the seasonal and daily cycles inherent in time series datasets. The flexibility of the cycling layer allows for the manual adjustment of periodic time features, enabling a more tailored and precise capture of cyclical trends. This ability to adjust and focus on specific cycle lengths with a reduced risk of overfitting irrelevant patterns represents a significant improvement over the traditional method. Furthermore, the simpler construction of the cycling layer compared to more complex positional encoding mechanisms can potentially reduce computational overhead and streamline the preprocessing phase, leading to cost efficiencies in both time and resources.

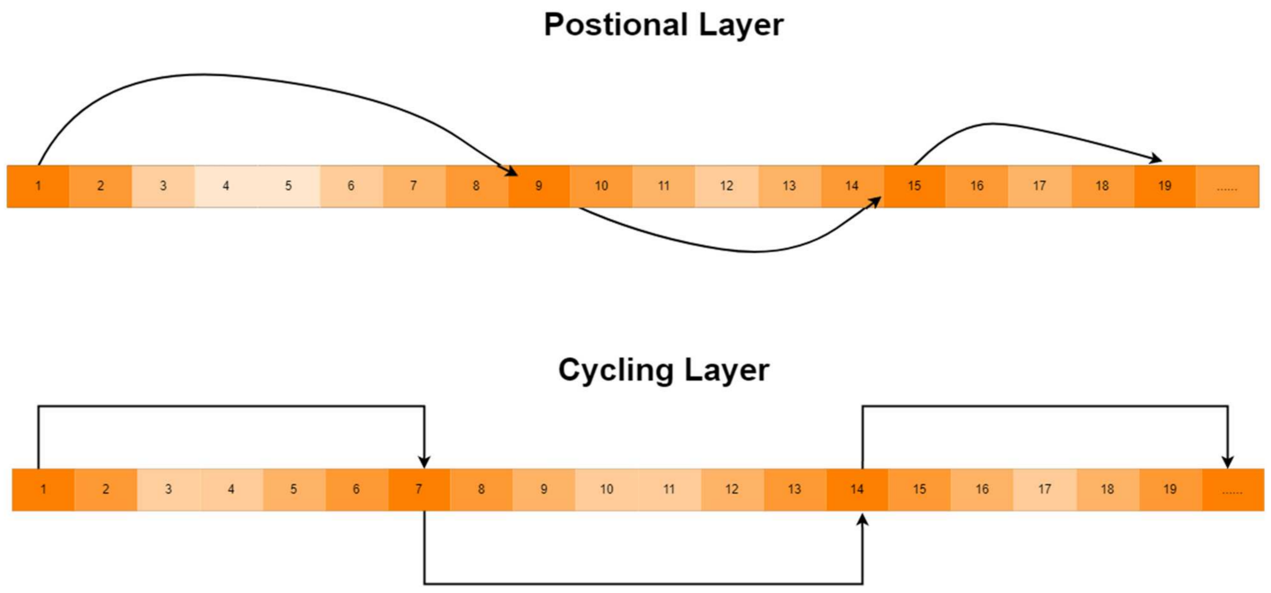

Figure 1. Difference between the positional layer and cycling layer: in the positional layer, the arrows show that the attention trend is based on mathematical properties and focuses on subsequent directions that are not fixed; in the cycling layer, the arrows reveal a periodic pattern in finding the next direction of focus. The color intensity reflects the weight of attention given to each sequence.

Suppose that the input representation $x_i \in \mathbb{R}^{n \times d}$, each value x in X is a timestamp t. Here, the cycling period represents the total time for one full cycle, such as 24 h for daily cycles. The result $t\prime$ will be a fraction between 0 and 1 representing the position of the timestamp within the cycle. Then, convert the fraction of the cycle $t\prime$ to radians, representing the position within the cycle in radians, covering a full circle from 0 to 2π. This equation (Equation (5)) transforms the timestamp t into two values, sin_t (Equation (6)) and cos_t (Equation (7)), which together represent the cyclic nature of the timestamp.

$$\theta = \left(\left(\frac{t}{cycling\ period\ time}\right) mod\ 1\right) \times 2\pi \tag{5}$$

$$sin_t = sin(\theta) \tag{6}$$

$$cos_t = \cos(\theta) \tag{7}$$

The cycling layer is a novel construct in the architecture of machine learning models designed for processing time series data. Algorithm 3 begins with the input tensor x, which is assumed to have a shape of (batch_size, sequence_length, embed_dim), where each value in x represents a timestamp. The design goal of the cycling layer is to encode these timestamps to make the cyclical information explicit and usable by the model. The cycling position features are the cornerstone of the cycling layer's functionality. They are derived from the radian-converted timestamps using sine and cosine functions, creating a pair of values for each point in time. This duality allows the model to capture the cyclical nature of time, ensuring that it recognizes the continuity at the cycle's boundaries—such as the transition from the end of one day to the beginning of the next. The sine and cosine values are concatenated along the tensor's last dimension, effectively doubling this dimension's size. This concatenation preserves the original timestamp information while augmenting it with cyclical encoding, providing a dual representation of linear time and its inherent cycles.

Algorithm 3: Cycling_Encoding

```
1 Input: batch_size, sequence_length, embed_dim
2 Output: combined feature x
3 x is assumed to be a tensor of shape (batch_size, sequence_length, embed_dim) where
  each value in x is a timestamp. The goal is to encode these timestamp
4   for pos from 0 to sequence_length $-1$ do
5     Convert x to fraction of the day $\left(\frac{x}{cycling\ period\ time}\right) mod\ 1$
6     Convert to radians $x \times 2\pi$
7     Concatenate$[\sin(x), \cos(x)]$, along last dimension
8   end for
9 Concatenate original feature with cyclically encoded feature
```

The final step in the cycling layer's process is concatenating the original features with these newly created cyclical position features. The result is a combined feature vector that retains all information from the original data enriched with a clear, cyclical context. This enriched representation is then fed into the self-attention mechanism, which can now take advantage of the cyclical patterns alongside the raw data features.

3.2.3. Autoencoder Preprocess with Cycling Layer

Within the autoencoder's encoding phase, we have innovatively integrated the cycling layer and self-attention layer. This integration is critical as it allows the encoder to reduce the data dimensionality and recognize and encode the cyclical patterns inherent to time series datasets.

In Figure 2, the data after cleaning and sequence creation will be moved to the autoencoder process. The dataset with 14×16 (original attributes with timestamp and order) dimensions allows these models to efficiently learn and compress data into lower-dimensional spaces, preserving the crucial sequential information that would be lost in flattened 2D representations. The encoding process commences may encompass various time series variables recorded over time. Initially, the data are routed through the cycling layer, which incorporates cyclical temporal information into the data representation. Subsequently, the enriched data are processed by the self-attention layer, which enables the network to emphasize information based on the cyclical context introduced by the preceding layer. As the data advance through the subsequent layers of the encoder, they are progressively compressed into a latent space—the core of the autoencoder. This latent space embodies the most salient features of the input data in a condensed format. At this bottleneck, the data, now represented as a compressed version of the original dataset, encapsulate the most crucial patterns and trends discerned by the network.

The decoding process mirrors the encoding process. The compressed representation obtained from the latent space is fed into the decoder. The decoder aims to reconstruct the original high-dimensional data from this compressed representation. This reconstruction occurs through a sequence of layers that progressively increase in dimensionality. In the initial stages, the decoder aims to re-expand the encoded features toward a format that approximates the original input data. Subsequent layers further refine this reconstruction by adjusting the output according to a loss function, which quantifies the discrepancy between the original data and the reconstructed data. The final output layer of the decoder is designed to align with the dimensions of the original input data, yielding the reconstructed data. These reconstructed data are then compared against the original dataset to evaluate the accuracy of the reconstruction and the efficacy of the encoding process.

In our experiment, the entire autoencoder process, encompassing both encoding and decoding, undergoes optimization and training. A loss function, such as the Mean Squared Error for continuous data, is employed to guide the adjustment of network weights to minimize reconstruction error. This process ensures that the autoencoder learns the most effective input data representation. The autoencoder refines its parameters through iterative training until the reconstruction loss is minimized, indicating that the network has acquired a robust representation of the time series data.

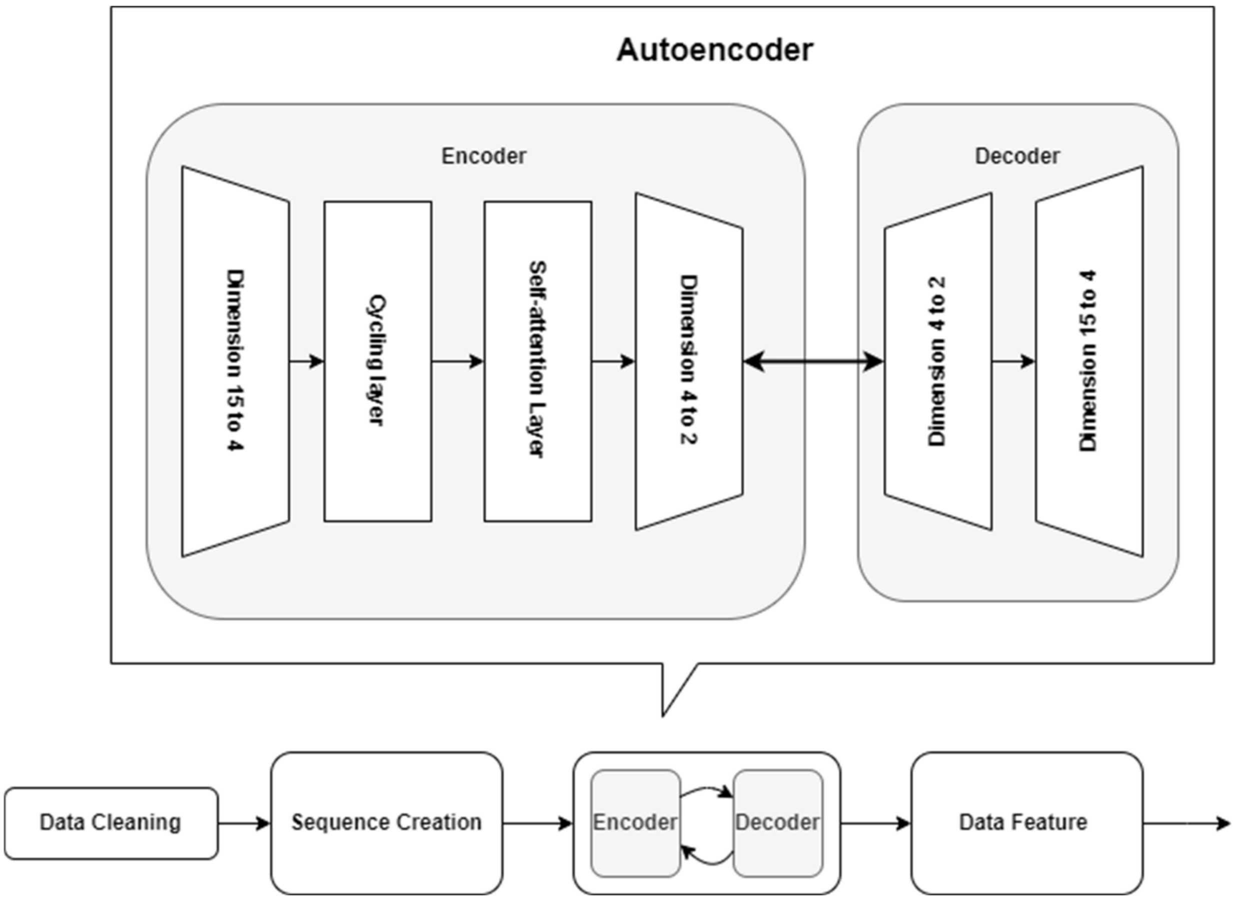

Figure 2. Preprocess plow chart about the autoencoder with a cycling layer.

The proposed autoencoder preprocessing method excels at extracting essential features from complex datasets, allowing models to concentrate on the most significant elements. Through Table 1, the cycling layer demonstrates its value when compared to traditional self-attention in an autoencoder. The comparison in the table is grounded in the inherent design strengths and weaknesses of positional and cycling self-attention layers based on the analysis and summary in the methodology section.

Table 1. Autoencoder challenge comparison between the design of positional and cycling self-attention.

Challenge of Autoencoder	With Positional Self-Attention	With Cycling Self-Attention
Capturing Long-Term Dependencies	Normal	Better
Manual Focusing on Relevant Time Steps	Bad	Better
Reducing Overfitting and Improving Generalization	Same	Same
Computational Efficiency	Normal	Better

Note: focusing on time series data and current experiment environment.

Based on the methodology analysis, positional self-attention captures short-term dependencies well but struggles with long-term cyclical patterns, while cycling self-attention is designed to handle periodic structures, excelling at capturing long-term dependencies. Positional self-attention treats each time step rigidly, which can make it less efficient at focusing on key cyclical moments, whereas cycling self-attention prioritizes relevant cyclical events. Positional self-attention may lead to overfitting and a larger generalization gap, while cycling self-attention reflects the same issue. Lastly, positional self-attention is computationally more complex, leading to slower convergence, whereas cycling self-attention improves efficiency by focusing on cyclical patterns and speeding up convergence.

3.3. Data Split

Partitioning the dataset into training, validation, and test sets is essential for developing predictive models. The training set, which constitutes the majority of the data, allows the model to learn and internalize patterns comprehensively. The test set, distinct from the training phase, provides an unbiased evaluation of the model's generalization capabilities. Additionally, validation subsets derived from the training data function as checkpoints for model refinement, helping to prevent overfitting and ensuring the robustness and reliability of the model's performance.

As illustrated in Figure 3, the data-splitting strategy begins with dividing the entire dataset into an 80–20 ratio. Eighty percent of the data is designated for the training set, which forms the foundation for autoencoder preprocessing. This training subset is processed through the autoencoder to generate a compressed representation of the input data, a crucial step for effective feature extraction.

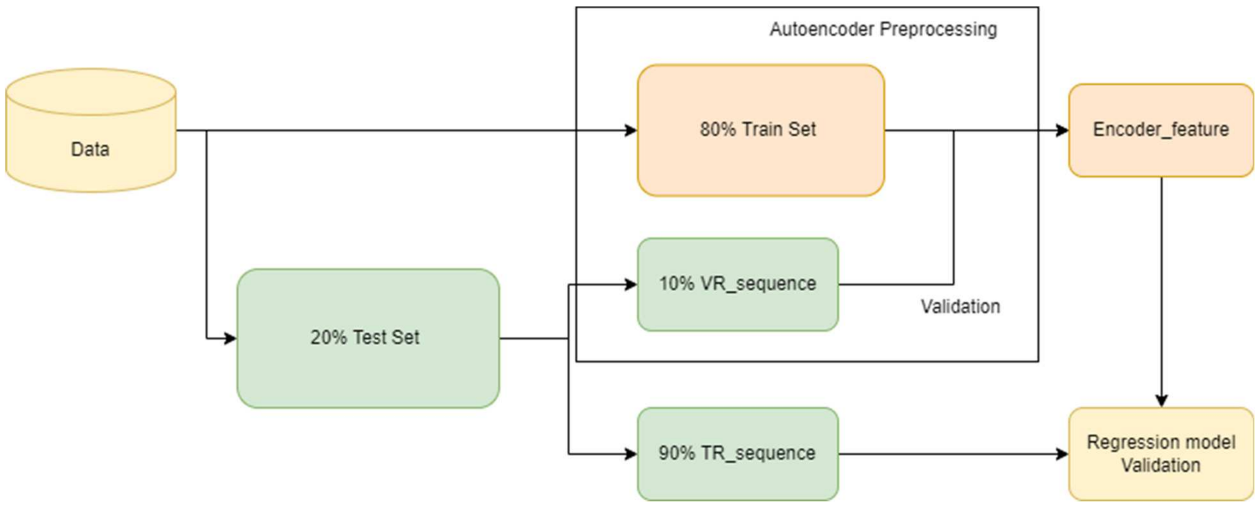

Figure 3. Data split flow graph.

The remaining 20% of the dataset is allocated to the test set, serving as the final assessment of the model's performance. This test set is further divided into validation *VR_sequence* and *Test TR_sequence* partitions, allowing for additional refinement and an evaluation of the model's generalization capabilities.

The *VR_sequence*, representing a subset comprising one-tenth of the training data, is reintroduced into the autoencoder as a validation set. This procedure facilitates model validation and employs early stopping techniques to mitigate overfitting. As the autoencoder iterates through compressing and reconstructing the training data, the *VR_sequence* provides an unbiased assessment of the model's ability to generalize beyond the training examples. This subset is crucial for calibrating the model, optimizing hyperparameters, and determining the optimal complexity of the network architecture.

The early stopping mechanism monitors the validation loss—calculated based on the *VR_sequence* throughout the training iterations. Training is halted if the validation loss does not show improvement or decreases only slightly over a predetermined number of iterations, known as the "patience" parameter. This process, commonly referred to as learning rate annealing or decay, allows the model to make finer adjustments as it approaches the minimum of the loss function, leading to more accurate convergence. The *VR_sequence*, which constitutes a portion of the test set, is employed for final validation across various regression models. This phase serves as a critical checkpoint to ensure that the representations learned by the encoding layer remain robust and accurate when applied to data not used in the initial training phase.

In the regression model validation stage, the features are generated and optimized through the autoencoder are evaluated using several regression techniques. The performance of these regression models reflects the encoded features' efficacy in capturing the original data's essential characteristics. This evaluation helps to determine the quality of the autoencoder's feature extraction process and its ability to facilitate accurate predictions across various modeling approaches.

3.4. Data Sample

The dataset utilized in this paper was sourced from the Jena Climate dataset, available through Kaggle. This dataset was collected at the Weather Station of the Max Planck Institute for Biogeochemistry in Jena, Germany, and is intended for climate and environmental research. It provides a comprehensive and detailed long-term record of various meteorological conditions. The data span a period of over eight years, from 1 January 2009 to 31 December 2016, offering a robust foundation for the time series analysis in this study. Data introduction was adapted from Chen, J.; Yang, Z. Enhancing Data Preprocessing using Positional Self-Attention Autoencoders. In Proceedings of the 2024 16th International Conference on Intelligent Human Machine Systems and Cybernetics (IHMSC 2024), Hangzhou, China, 24–25 August 2024 [2].

Table 2 provides a comprehensive summary of the range and mean values for each feature, reflecting distinct aspects of the atmospheric state. Atmospheric pressure, ranging from 913.60 to 1015.35 mbar, indicates the weight of the air and is crucial in shaping weather patterns. Temperature data, ranging from −23.01 °C to 37.28 °C, offer insights into atmospheric stability, including the dew point temperature, a key indicator of atmospheric moisture. Relative humidity, spanning from 12.95% to 100%, impacts weather conditions and ecological systems. Water vapor metrics, including the maximum water vapor pressure, actual vapor pressure, vapor pressure deficit, specific humidity, and water vapor concentration, collectively delineate the moisture level of the air, influencing precipitation, cloud formation, and overall atmospheric processes. Air density, affected by temperature, Version 7 August 2024, submitted to Journal Not Specified 4 of 24 pressures and humidity, has implications across various fields, from aviation to engineering. Despite some data anomalies, the wind speed and direction are fundamental in understanding air movement, weather systems, and pollutant dispersion. The range and mean of these variables illustrate the dynamic and complex nature of the climate at the Jena station, offering deep insights into weather patterns and providing a robust foundation for extensive climatic research and applications in forecasting, environmental planning, and beyond.

Table 2. The overall range and mean for each feature reflecting distinct aspects of data features.

Feature	Mean	Minimum	Maximum
p (mbar)	989.21	913.60	1015.35
T (degC)	9.45	−23.01	37.28
Tpot (K)	283.49	250.60	311.34
Tdew (degC)	4.96	−25.01	23.11
rh (%)	76.01	12.95	100.00
VPmax (mbar)	13.58	0.95	63.77
VPact (mbar)	9.53	0.79	28.32
VPdef (mbar)	4.04	0.00	46.01
sh (g/kg)	6.02	0.50	18.13
H2OC (mmol/mol)	9.64	0.80	28.82
rho (g/m^3)	1216.06	1059.45	1393.54
wv (m/s)	1.70	−9999.00	28.49
max. wv (m/s)	3.06	−9999.00	23.50
wd (deg)	174.74	0.00	360.00

Note: data anomalies exist in wind speed measurements.

Figure 4 illustrates the relationships between various features within the dataset. Temperature T (degC) demonstrates a strong positive correlation with both potential temperature Tpot (K) and dew point temperature Tdew (degC), suggesting a cohesive relationship among different measures of atmospheric heat. The specific humidity sh (g/kg), water vapor concentration H2OCmmol/mol, and actual vapor pressure VPact (mbar) exhibit close interrelationships, reflecting the interconnected nature of moisture variables. Conversely, relative humidity rh% shows an inverse correlation with temperature, highlighting the dynamic balance between temperature and the air's moisture-carrying capacity. Atmospheric pressure p (mbar) negatively correlates with temperature, consistent with the principles of thermal expansion. Wind speed wv (m/s) and maximum wind speed max. wv (m/s), while showing low correlation with other variables, indicate the relative independence of wind characteristics from other meteorological factors. It is important to acknowledge data anomalies for precise interpretation. The dew point temperature Tdew (degC) closely aligns with air temperature, reinforcing the relationship between temperature and moisture content. Finally, the vapor pressure deficit VPdef (mbar) underscores its significance in understanding the air's drying capacity, showing a positive correlation with temperature and a negative correlation with relative humidity, which is crucial for studies on evaporation and plant transpiration. These relationships underscore the intricate and interdependent dynamics of climatic factors within the dataset.

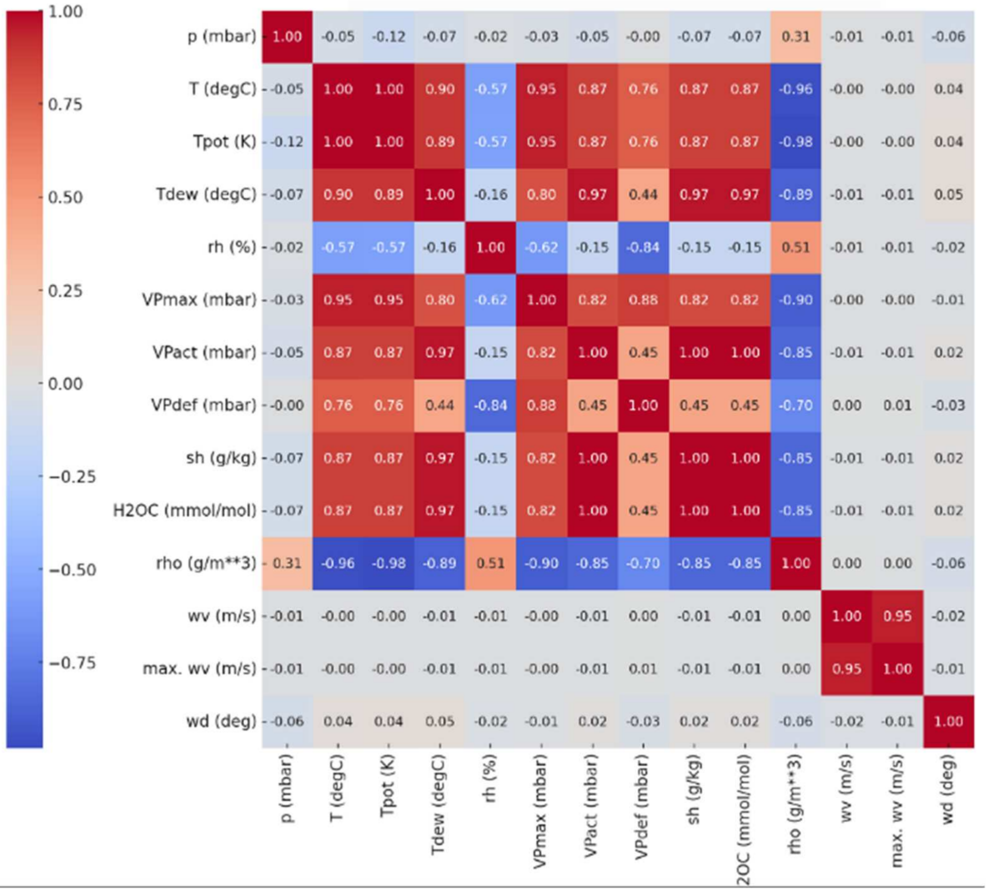

Figure 4. Correlation matrix of the Jena Climate dataset.

The experimental setup involves integrating the proposed cycling layer into a self-attention mechanism within an autoencoder architecture. We use the Adam optimizer and a learning rate of 0.001 for training, with early stopping based on validation performance to prevent overfitting. Meanwhile, we also need to undertake preprocessing steps to prepare the Jena Climate dataset for subsequent analysis using an autoencoder model. This section details the conversion of the date–time column from its original form into a Unix timestamp, thereby standardizing it for easier model ingestion and alignment with other time series data processing methods. Also, we aim to address the problem of missing values, a common issue in real-world datasets. If this problem is addressed, it can significantly distort model training and predictions. The chosen method, mean imputation, replaces missing values with the mean of each column, maintaining the overall statistical characteristics of the dataset. Lastly, this section discusses the creation of sequences from the processed data. Autoencoders, especially those designed for time series data, benefit from learning sequential patterns and dependencies, necessitating the transformation of the dataset into a sequence of data points.

4. Result and Discussion

Evaluating encoded features is vital for assessing autoencoder performance, as it reveals the quality of the data transformations and their impact on downstream tasks. This analysis helps determine how effectively the autoencoder compresses and represents the data, influencing the training efficiency and model accuracy. Comparing the correlation matrices of original and encoded features provides insight into the autoencoder's success in reducing redundancy and capturing the essential data structure, which informs future model and preprocessing choices.

The encoded features, following preprocessing and transformation by the autoencoder, exhibit a notably distinct correlation matrix compared to the original data (Figure 4). The preprocessing phase has regularized the relationships between variables, as evidenced by the more uniform and moderated correlation coefficients. In the original dataset, variables demonstrate significant positive and negative correlations, reflective of the inherent interdependencies in raw environmental data. In contrast, the encoded features reveal a correlation structure. Figure 5 indicates a transformation towards a space where variables are more orthogonal or independent from one another.

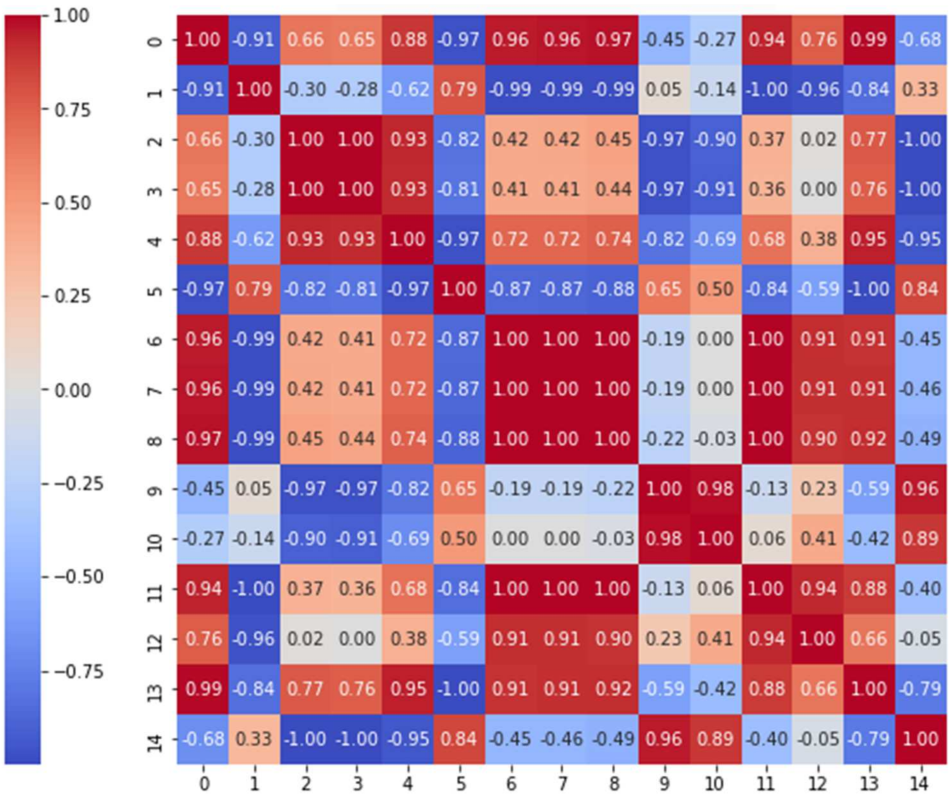

Figure 5. Correlation matrix for encoded feature.

Figure 5 displays the correlation matrix for the encoded features, essential for understanding the interrelationships among variables in the dataset. Each matrix cell indicates the correlation between features, with 1 representing a perfect positive correlation and −1 a perfect negative correlation. The shades of gray illustrate the strength of these correlations. This matrix is crucial for identifying highly correlated features that may be redundant, thereby aiding in feature selection and dimensionality reduction. This phase enhances the model's efficiency and accuracy by thoroughly analyzing the encoded features, ensuring that only the most relevant and independent features are utilized for training.

This new structure within the encoded features implies a significant dimensional reduction and a distillation of the raw data into a form where the overlapping information is minimized. The correlation matrix from the encoded features, typically the output from an autoencoder's bottleneck layer, would show the relationship between each pair of encoded variables. In a well-trained autoencoder, especially one created to produce disentangled representations, we would expect to see a correlation matrix with lower off-diagonal values. This indicates that the encoded features are less correlated with each other, meaning the autoencoder has learned a representation that separates the underlying factors of the data. This is desirable because it suggests that each encoded feature captures a unique aspect of the input data, reducing redundancy and potentially improving the performance of downstream tasks. Furthermore, the correlation matrices of the original and encoded features serve as compelling visual evidence of the impact of preprocessing. Where the original data's correlation matrix shows extensive correlation coefficients and the encoded data's matrix demonstrates a more subdued variance, signaling a more refined feature set.

The practical implications of these improvements are significant. In anomaly detection, models trained on less correlated and more disentangled features are likely to respond more to deviations from standard patterns, thereby enhancing their detection capabilities. In forecasting, the refined features can contribute to models that generalize more effectively from the training data to new, unseen data, leading to more accurate and reliable predictions. This enhanced ability to capture and represent the underlying structure of the data translates into better performance across various analytical tasks.

The numerical research serves two purposes: (1) comparing fitting speed of different models; (2) testing regression model performance based on different models.

Comparing the model fitting speed allows for a comparison of the efficiency of autoencoder models employing traditional positional layer self-attention against those using cycling layer self-attention. This comparison is vital as it reveals the practicality and performance of the models in real-world applications, particularly in handling large and complex datasets, like the Jena Climate dataset. Understanding which configuration yields faster convergence to the best fit can significantly impact the choice of model architecture in future implementations and research.

The key metric for this comparison is the overall number of iterations taken by each model to reach the best fit, which is determined by the lowest training loss achieved. This metric is chosen as it directly reflects the time efficiency of the model in learning from the data and reaching a point of optimal performance.

Regression model performance comparison aims to evaluate and compare the performance of different regression models that utilize the representations learned by autoencoder models. This evaluation is crucial for understanding how effectively the learned representations can predict or reconstruct relevant outputs, thereby determining the practical utility of the autoencoder models in real-world applications. This comparison uses MSE, MAE, RMSE, and median MAE as the metrics for different regression methodologies. To assess the specific contribution of the cycling layer, we conducted an ablation study comparing different model configurations: (1) with only the positional layer, (2) with only the cycling layer.

4.1. Model Fitting Speed

To maintain the integrity of the comparison, the number of iterations and losses were meticulously recorded for each autoencoder model throughout the training. This recording began from the initialization of training and continued at each iteration, capturing the time taken and the loss incurred until the end of the training.

From Table 3, the number of iterations for the positional layer autoencoder shows significant variability, ranging from as low as 5 to as high as 48. The loss values also vary considerably, indicating fluctuations in model training performance across different runs; its average number of iterations without counting outliers is 277. The variability suggests that the positional layer autoencoder may struggle with consistency, potentially due to the complexity or non-cyclic nature of the data. The cycling 12 h autoencoder has more consistent and generally lower iteration times, with a minimum of 72 and a maximum of 255, considerably lower than the positional layer autoencoder times. The loss values are also more consistent, hovering around a narrower band. The average number of iterations of the cycling 12 h autoencoder is around 191. The cycling 24 h autoencoder also demonstrates improved consistency in the number of iterations, ranging from 81 to 210, and maintains loss values close to those of the 12 h cycling model. It has the lowest number of iterations, which is around 169 compared to the others. The 24 h cycling model is likely capturing the full diurnal cycle, which might be a strong component in the data due to natural daily rhythms in weather patterns.

Table 3. Comparative analysis of the number of iterations and loss metrics for three different autoencoder architectures: a positional layer autoencoder, a cycling layer autoencoder with a 12 h cycle, and a cycling layer autoencoder with a 24 h cycle.

Positional		**Cycling 12**		**Cycling 24**	
Number of Iterations	**Loss**	**Number of Iterations**	**Loss**	**Number of Iterations**	**Loss**
485	98,786,208	176	98,713,768	169	98,713,792
51	101,394,888	187	98,713,768	175	98,713,792
96	99,171,104	166	98,713,768	151	98,713,808
199	99,171,104	202	98,713,792	149	98,713,936
265	98,720,472	225	98,713,792	144	98,713,856
305	98,720,472	169	98,713,752	210	98,713,768
272	98,713,776	193	98,713,768	81	98,727,216
103	98,713,776	201	98,713,944	192	98,713,832
269	98,715,536	199	98,713,848	200	98,713,920
239	98,713,896	72	98,804,704	132	98,713,856

Table 3 enumerates the number of iterations and losses for positional, cycling 12, and cycling 24 layers. Cycling layers demonstrate a reduced number of iterations, implying faster processing than the positional layer. For instance, the cycling 24 layers exhibit the number of iterations as low as 81, whereas the positional layer's minimum of a similar loss value is 96, suggesting a more efficient training process. Thus, loss values across cycling 12 and cycling 24 are consistently lower than positional, which can imply better convergence. Table 2 suggests that cycling layers enhance speed and minimize loss, indicating a more efficient training process. Therefore, the graph infers that both cycling layer autoencoders outperform the positional layer autoencoders regarding training efficiency. This is evident from the less time for each iteration, suggesting that these models adapt to the training data more quickly. The more consistent loss values across iterations for the cycling models indicate stable learning and potentially better generalization capabilities.

Additionally, Table 3 presents the loss values for different models incorporating either a positional layer or cycling layer with varying cycle lengths (12 h and 24 h). The loss value in this context typically represents the difference between the predicted values and the actual values of the time series data, measured using the Mean Squared Error (MSE) and Mean Absolute Error (MAE). These loss metrics evaluate how well the model performs, with lower loss values indicating better accuracy in the model's predictions. The table shows that models using the cycling layers (both 12 h and 24 h cycles) have consistently lower loss values compared to those with the positional layer, suggesting that the cycling layers are more effective at capturing long-term cyclical patterns in time series data. The cycling 24 h model has the lowest overall loss, indicating that it aligns well with the natural daily cycles present in the weather data, leading to more accurate predictions.

4.2. Regression Model Performance

The chart in Figure 6 uses a logarithmic scale to display the MSE values, which allows for a clear visual representation of differences across multiple values. Lower MSE values indicate better model performance as they signify that the model's predictions are closer to the actual values (the content below shows numbers estimated to five decimal places).

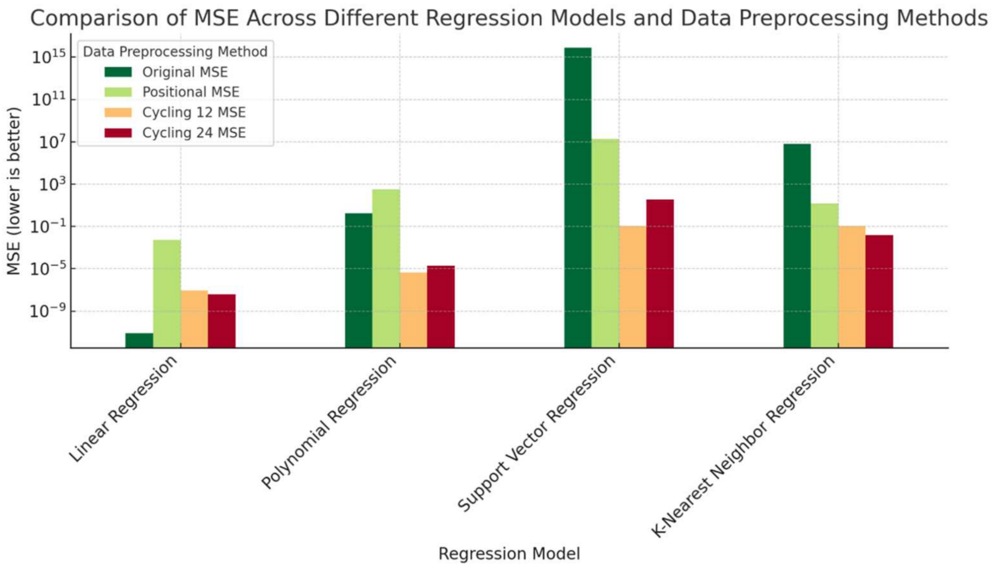

Figure 6. MSE for four different regressions.

This method processes the data by considering a 12 h cycle, which could be useful for capturing daily patterns that repeat on a half-day basis. When analyzing the cycling 12 method, it is observable that for SVR, the MSE is very low at around 0.10717, indicating that this model has performed well with the 12 h cycle data preprocessing. However, for the Polynomial and Linear Regression models, the cycling 12 preprocessing does not yield the lowest MSE, suggesting that these models may not benefit as much from this preprocessing method as SVR. The cycling 24 method considers an entire 24 h cycle, potentially capturing daily patterns. For KNN, just like with the cycling 12, the MSE is exceedingly low at around 0.01508, implying that KNN is particularly well-suited to benefit from the cycling approach. Polynomial Regression and SVR are slightly improved in MSE with cycling 24 compared to cycling 12, indicating some benefit from considering the entire daily cycle.

Upon examining the Mean Absolute Error (Figure 7) across various regression models and data preprocessing methods, cycling 12 and cycling 24 methods offer substantial improvements in predictive accuracy over the original and positional methods. Specifically, for Linear Regression, cycling 12 and cycling 24 achieve a drastically lower MAE, which is 0.00024 and 0.00002, respectively, with cycling 24 edging out as the most effective, suggesting its superior capability in capturing daily patterns within the data. Polynomial Regression also benefits from the cycling approaches, albeit to a lesser degree, with cycling 12 (0.00160) outperforming cycling 24 (0.00214). Support Vector Regression reflects a similar trend, with cycling 12 (around 0.10680) reducing the MAE more than cycling 24 (around 0.16090) when compared to positional encoding. Remarkably, the K-Nearest Neighbor Regression model exhibits the most pronounced improvement with cycling 24 at around 0.00773, reaching the lowest MAE among all the preprocessing methods, underscoring the value of a complete 24 h cycle in capturing the inherent temporal dynamics of the dataset.

These findings highlight the effectiveness of incorporating cyclical temporal information into the preprocessing phase, particularly for models sensitive to time-based patterns.

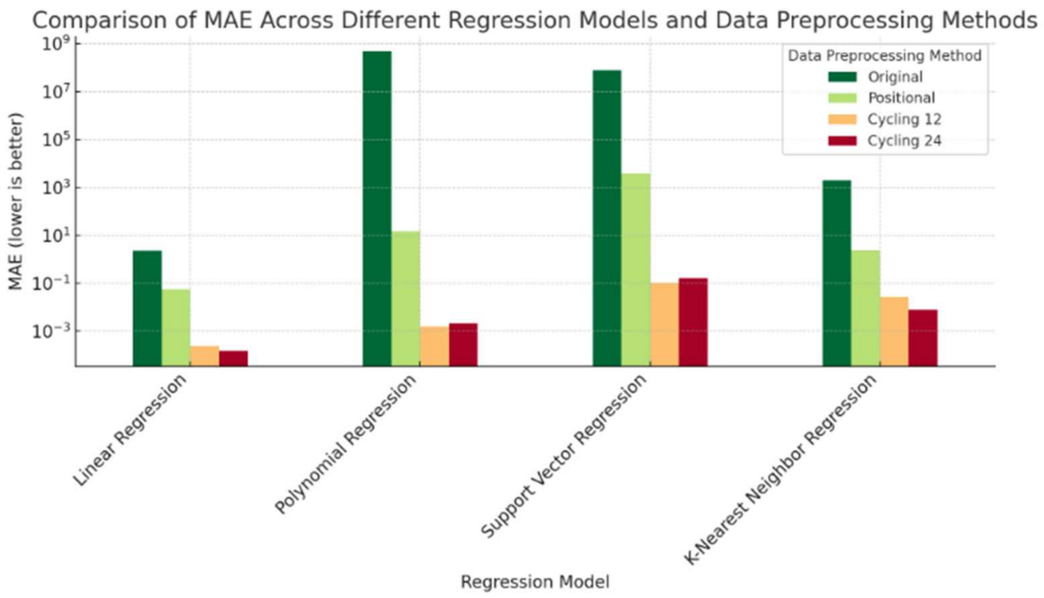

Figure 7. MAE for four different regressions.

In Figure 8, the RMSE values suggest that Linear Regression benefits significantly from the cycling 12 (0.00031) and cycling 24 (0.00020) methods, showcasing much lower errors compared to the original (2.84823) and positional (0.07078) methods. This indicates a strong fit of the model to the data when temporal cycles are considered. The RMSE for Polynomial Regression and Support Vector Regression, while lower for cycling 12 and cycling 24 than for positional, does not dramatically improve, as observed with Linear Regression. K-Nearest Neighbor Regression was also enhanced, with cycling 12 at around 0.34340 and cycling 24 at around 0.12270, suggesting that these models have a better ability to capture the central tendency of the data in a more accurate way with cycling-based preprocessing.

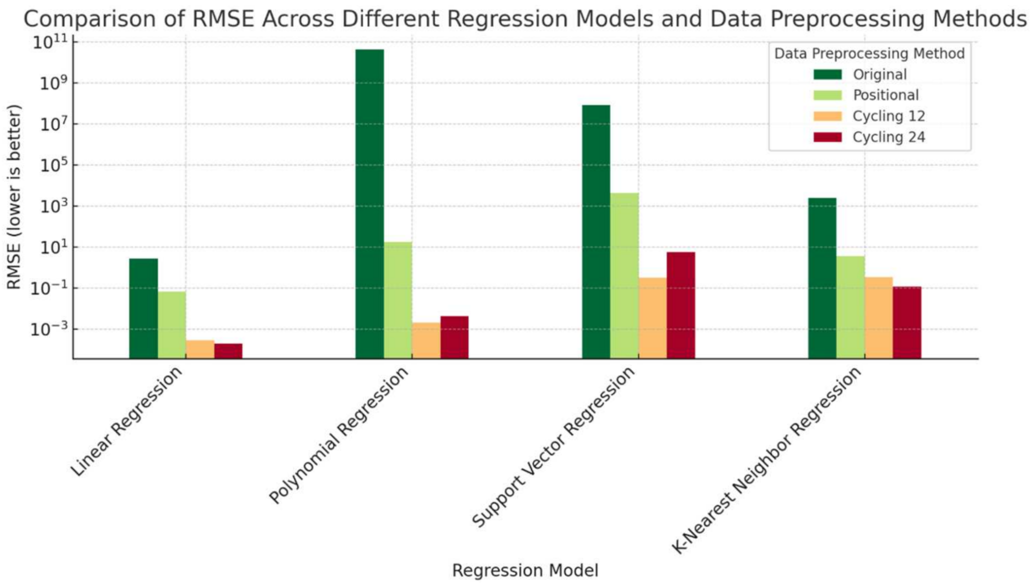

Figure 8. RMSE for four different regressions.

Figure 9, focusing on Med AE, presents a similar trend. Linear Regression again shows a marked improvement with the cycling 12, at around 0.00024, and cycling 24, at around 0.00013, methods, with Med AE dropping significantly. This metric confirms that the central tendency of the predictions made by the Linear Regression model aligns closely with the actual values when cyclic patterns are integrated into the data preprocessing. For Polynomial Regression and Support Vector Regression, the Med AE is notably reduced with cycling 12 and cycling 24, albeit to a lesser extent than Linear Regression. K-Nearest Neighbor Regression displays the absolute consistency for Med AE at 0 with cycling 24 and cycling 12, underscoring the benefit of this preprocessing in reducing the typical error in predictions.

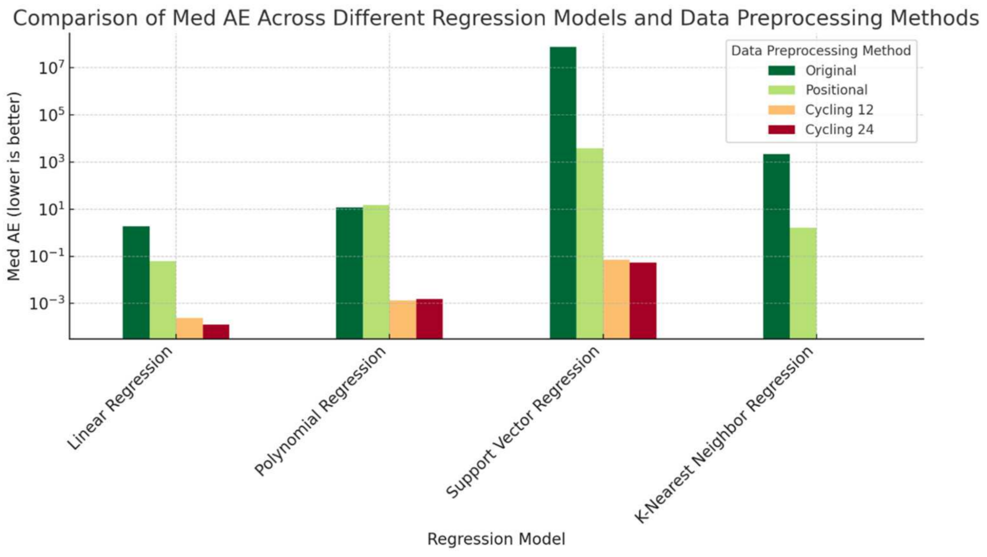

Figure 9. MedAE for four different regressions.

Cycling 12 and cycling 24 enhance Linear Regression performance, as indicated by substantially lower MSE and MAE values compared to the original and positional methods. Specifically, the cycling 24 method appears to be most effective in capturing full daily patterns within the data, which is beneficial for time series datasets with pronounced diurnal or seasonal variations.

For Polynomial Regression, a slight improvement in MSE with cycling 24 compared to cycling 12 is noted, indicating some benefit from considering the entire daily cycle. In the case of SVR, the improvements in MAE suggest that both cycling methods can reduce the typical prediction error, although the extent of improvement is less than that observed in Linear Regression. The KNN model exhibits the most remarkable improvement in MAE with cycling 24, emphasizing the value of an entire 24 h cycle in capturing the inherent temporal dynamics of the dataset. This improvement is significant, as it highlights that integrating cyclical temporal information into preprocessing can substantially enhance predictive accuracy for models sensitive to time-based patterns.

The RMSE analysis aligns with these findings, where Linear Regression benefits significantly from the cycling 12 and cycling 24 methods, revealing a solid fit to the data when temporal cycles are considered. KNN Regression also improves with these cycling-based preprocessing methods, validating the value of the cycling layer. Overall, the results from the comparative analysis suggest that cycling 12 and cycling 24 preprocessing methods redefine the paradigm in regression analysis by incorporating cyclical patterns, leading to more accurate predictions in strong cyclical patterns in time series data.

While Figure 6 indicates that Linear Regression achieves the lowest MSE in this specific dataset, which does not necessarily imply that it is the optimal choice for all scenarios. The proposed cycling layer integrated with self-attention mechanisms demonstrates superior performance in metrics, such as the Mean Absolute Error (MAE) and Root Mean Squared

Error (RMSE), which are more indicative of real-world performance when dealing with complex and non-linear time series data. Moreover, our approach captures long-term cyclical patterns that Linear Regression cannot model effectively, as evidenced by its stable performance across different datasets and extended time horizons. This adaptability and robustness make our method more suitable for applications that require handling dynamic, non-linear data patterns, such as environmental monitoring, financial forecasting, and anomaly detection in industrial settings.

To further assess the efficacy of the cycling layer self-attention autoencoder preprocessing, we conducted comparative experiments involving four regression models in conjunction with the existing sequence model. In this experiment, the same data split was used to maintain consistency, except that both LSTM and GRU models included 12 h and 24 h time factors as focal points. In Table 4, it can be seen that the cycling layer preprocessing with Linear Regression under the 24 h time factor significantly leads in MAE performance. For MSE, the cycling layer preprocessing with KNN regression achieves the best performance. Other regression methods also outperform LSTM and GRU models to varying degrees, which indirectly proves the advantage of using cycling layer self-attention preprocessing in this data environment.

Table 4. Four regression models with cycling layer preprocessing against LSTM and GRU.

	Linear Regression	Polynomial Regression	Support Vector Regression	KNN Regression	LSTM	GRU
MAE	**0.0001487 (24 h)**	0.00159 (12 h)	0.1068 (12 h)	0.0077 (24 h)	0.00238	0.00226
MSE	3.978 (24 h)	2.0274 (24 h)	0.1071 (12 h)	**0.0150 (24 h)**	1.2093	1.1162

Note: The bold marks the best MSE and MAE scores.

Comparing other research from the same dataset, it can still show the novelty. Qin et al. [24] applied LSTM networks to forecast climate data, achieving improved accuracy for short-term predictions but struggling with longer-term dependencies. Yacoub et al. [25] combined LSTM and Prophet models to capture daily cycles effectively, yet this approach required significant computational resources. Shen et al. [26] employed advanced RNNs to model temporal relationships, demonstrating good performance in handling sequential data but with limited capacity to capture cyclical patterns. Unlike these methods, our proposed approach introduces a cycling layer to better capture both short-term and long-term cyclical patterns, resulting in lower error rates and faster model fitting times.

Furthermore, the sensitivity of the model's performance to various hyper-parameters, such as the cycling period and the choice of activation functions, also plays a crucial role in determining its effectiveness. The cycling period defines how well the model captures repetitive patterns in time series data. We set 12 h or 24 h per cycling in the experiment. If the period is too short, the model may focus only on short-term dependencies, missing out on long-term trends. Conversely, a period that is too long might overlook shorter, important fluctuations. Similarly, the choice of activation functions impacts how the model handles non-linearities in the data. Functions like ReLU can speed up training but may suffer from issues like "dead neurons," while tanh or sigmoid activations may better capture smooth cyclic patterns but could be prone to the vanishing gradient problem. The paper does not provide a detailed exploration of how these hyper-parameters influence model performance, particularly in terms of the forecast accuracy and model fitting speed.

While the results are encouraging, several limitations of the testing process should be acknowledged. The models were evaluated using a narrow set of metrics, and critical factors, such as computational efficiency, robustness to outliers, and scalability, were not addressed. Additionally, the performance of the cycling layer may vary with different parameter configurations, and a more comprehensive hyperparameter search could potentially yield different outcomes. Moreover, the findings are constrained to the specific conditions of the tests that were conducted. The observed improvements with the cycling layer may not be universally applicable across all datasets or problem domains. It is also important to recognize that the cyclical patterns examined—specifically the 12 h and 24 h cycles—were selected based on specific assumptions that might not be relevant to other temporal patterns or phenomena.

5. Conclusions and Future Research Directions

This paper introduces a novel approach for time series data preprocessing by integrating cycling layers into self-attention mechanisms within an autoencoder framework. The proposed method specifically addresses the challenges of capturing both short-term and long-term cyclical patterns, such as those found in weather data, thereby improving the accuracy of time series forecasting models. Unlike traditional methods that struggle with these complexities, our approach demonstrates significant enhancements in identifying and leveraging cyclical dependencies, as shown through experiments on the Jena Climate dataset.

While the empirical results indicate improvements in both the forecast accuracy and model fitting speed, it is essential to contextualize these gains. The method excels in scenarios where cyclical patterns are prominent and must be captured effectively, such as in environmental monitoring and financial time series forecasting. However, these improvements in accuracy do come with computational costs. Although the cycling layer enhances the model's ability to handle complex temporal dependencies, it introduces additional computational overhead compared to simpler preprocessing techniques. Future research should explore optimizing this approach to balance accuracy with computational efficiency more effectively.

In conclusion, the integration of cycling layers into self-attention mechanisms offers a promising avenue for advancing time series analysis, particularly in domains where capturing periodic patterns is critical. Further investigation is needed to refine this technique for broader applications and to explore its utility in conjunction with other advanced machine learning methodologies.

Author Contributions: Conceptualization, J.C. and Z.Y.; methodology, J.C.; software, J.C.; validation, J.C. and Z.Y.; formal analysis, J.C.; investigation, J.C. and Z.Y.; resources, Z.Y.; data curation, Z.Y.; writing—original draft preparation, J.C.; writing—review and editing, Z.Y.; visualization, J.C.; supervision, Z.Y.; project administration, Z.Y.; funding acquisition, Z.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Science and Engineering Research Council (NSERC) of Canada, grant number RGPIN-2018-05988.

Data Availability Statement: The original contributions presented in the study are included in the article.

Acknowledgments: This article is a revised and expanded version of a paper entitled "Enhancing Data Preprocessing using Positional Self-Attention Autoencoders", which was presented at 2024 16th the International Conference on Intelligent Human Machine Systems and Cybernetics (IHMSC 2024), Hangzhou, 24–25 August 2024 [2].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Brockwell, P.J.; Davis, R.A. *Introduction to Time Series and Forecasting*; Springer: Berlin/Heidelberg, Germany, 2010.
2. Chen, J.; Yang, Z. Enhancing Data Preprocessing using Positional Self-Attention Autoencoders. In Proceedings of the 2024 16th International Conference on Intelligent Human Machine Systems and Cybernetics (IHMSC 2024), Hangzhou, China, 24–25 August 2024.
3. Fang, Z. Long-and Short-Term Sequential Recommendation with Enhanced Temporal Self-Attention. Master's Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2022. Available online: https://pure.tue.nl/ws/portalfiles/portal/199146119/Fang_Z.pdf (accessed on 11 October 2023).
4. Maharana, K.; Mondal, S.; Nemade, B. A review: Data pre-processing and data augmentation techniques. *Glob. Transit. Proc.* **2022**, *3*, 100065. [CrossRef]

5. Huang, L.; Huang, J.; Chen, P.; Li, H.; Cui, J. Long-term sequence dependency capture for spatiotemporal graph modeling. *Knowl. Based Syst.* **2023**, *278*, e110818. [CrossRef]
6. Yao, H.; Tang, X.; Wei, H.; Zheng, G.; Li, Z. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. *arXiv* **2018**, arXiv:1803.01254. [CrossRef]
7. Yang, B.; Dai, J.; Guo, C.; Jensen, C.S.; Hu, J. PACE: A PAth-CEntric paradigm for stochastic path finding. *VLDB J.* **2018**, *27*, 153–178. [CrossRef]
8. Dancker, J. A Brief Introduction to Time Series Forecasting Using Statistical Methods. Towards Data Science. 2022. Available online: https://towardsdatascience.com/a-brief-introduction-to-time-series-forecasting-using-statistical-methods-d4ec849658c3 (accessed on 10 October 2023).
9. Jujjuru, G. Learning Time Series Analysis & Modern Statistical Models. Analytics Vidhya. 2023. Available online: https://www.analyticsvidhya.com/blog/2023/01/learning-time-series-analysis-modern-statistical-models (accessed on 10 October 2023).
10. Haykin, S. *Adaptive Filter Theory*; Pearson: London, UK, 2013.
11. Box, G.E.P.; Jenkins, G.M.; Reinsel, G.C. *Time Series Analysis: Forecasting and Control*, 5th ed.; John Wiley and Sons Inc.: Hoboken, NJ, USA, 2008.
12. Brigham, E.O. *The Fast Fourier Transform and Its Applications*; Prentice Hall: Englewood Cliffs, NJ, USA, 1974.
13. Haykin, S. *Array Signal Processing*; Prentice Hall: Englewood Cliffs, NJ, USA, 1985.
14. Cohen, L. *Time-Frequency Analysis*; Prentice Hall: Englewood Cliffs, NJ, USA, 1995.
15. Carlin, B.P.; Louis, T.A. *Bayesian Methods for Data Analysis*, 3rd ed.; CRC Press: Boca Raton, FL, USA, 2008.
16. Hyndman, R.J.; Athanasopoulos, G. *Forecasting: Principles and Practice*; Otexts: Melbourne, VIC, Australia, 2018.
17. Massari, C.; Su, C.H.; Brocca, L.; Sang, Y.F.; Ciabatta, L.; Ryu, D.; Wagner, W. Near real-time de-noising of satellite-based Soil Moisture Retrievals: An intercomparison among three different techniques. *Remote Sens. Environ.* **2017**, *198*, 17–29. [CrossRef]
18. Cortés-Ibáñez, J.A.; González, S.; Valle-Alonso, J.J.; Luengo, J.; García, S.; Herrera, F. Preprocessing methodology for Time Series: An industrial world application case study. *Inf. Sci.* **2020**, *514*, 385–401. [CrossRef]
19. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
20. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
21. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. In Proceedings of the NIPS 2014 Workshop on Deep Learning, Montreal, QC, Canada, 12–13 December 2014.
22. Wibawa, A.P.; Utama, A.B.P.; Elmunsyah, H.; Pujianto, U.; Dwiyanto, F.A.; Hernandez, L. Time-series analysis with smoothed Convolutional Neural Network. *J. Big Data* **2022**, *9*, 44. [CrossRef] [PubMed]
23. Sagheer, A.; Kotb, M. Unsupervised pre-training of a deep LSTM-based stacked Autoencoder for multivariate time series 898 forecasting problems. *Sci. Rep.* **2019**, *9*, e19038. [CrossRef] [PubMed]
24. Kieu, T.; Yang, B.; Guo, C.; Jensen, C.S. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; pp. 2725–2732. [CrossRef]
25. Li, Z. Jena Climate EDA & ARIMA [Kaggle Notebook]. *Kaggle*. 2023. Available online: https://www.kaggle.com/code/zhiyueli/jena-climate-eda-arima/notebook (accessed on 6 December 2023).
26. Qin, L. Jena Climate Prediction with LSTM [Kaggle Notebook]. Kaggle. 2021. Available online: https://www.kaggle.com/code/lonnieqin/jena-climate-prediction-with-lstm (accessed on 5 November 2023).
27. Yacoub, L. Daily Forecasting LSTM & FB Prophet [Kaggle Notebook]. Kaggle. 2021. Available online: https://www.kaggle.com/code/leminayacoub/daily-forecasting-lstm-fb-prophet (accessed on 11 October 2023).
28. Shen, J. TensorFlow 3: RNN [Kaggle Notebook]. *Kaggle*. 2021. Available online: https://www.kaggle.com/code/jingxuanshen/tensorflow-3-rnn (accessed on 11 November 2023).
29. Muhammad, H.H. Wrangling Concepts with Time Series Data [Kaggle Notebook]. Kaggle. 2022. Available online: https://www.kaggle.com/code/muhammadhammad02/wrangling-concepts-with-time-series-data (accessed on 6 September 2023).
30. Goodfellow, I.; Bengio, Y.; Courville, A.; Bengio, Y. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016; Volume 1.
31. Zhang, G.; Liu, Y.; Jin, X. A survey of autoencoder-based recommender systems. *Front. Comput. Sci.* **2020**, *14*, 430–450. [CrossRef]
32. Sarker, I.H.; Abushark, Y.B.; Khan, A.I. ContextPCA: Predicting context-aware smartphone app usage based on machine learning techniques. *Symmetry* **2020**, *12*, 499. [CrossRef]
33. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
34. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv* **2017**, arXiv:1701.06538.
35. Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv* **2014**, arXiv:1409.1259.
36. Parikh, A.P.; Täckström, O.; Das, D.; Uszkoreit, J. A decomposable attention model for natural language inference. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016; pp. 2249–2255.

37. Shaw, P.; Uszkoreit, J.; Vaswani, A. Self-Attention with Relative Position Representations. In Proceedings of the 2018 Conference 914 of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, New Orleans, LA, USA, 1–6 June 2018; pp. 464–468.
38. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is All You Need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*; Curran Associates, Inc.: Long Beach, CA, USA, 2017; pp. 5998–6008.