Universiteit Leiden

Master Computer Science

TCNA: Analysing patterns in time series using TCN and Self-Attention

Name: Tushar Pal
Student ID: s3208796
Date: 28/08/2023

Specialisation: Artificial Intelligence

1st supervisor: Dr. E.M. Bakker
2nd supervisor: Prof.dr. M.S.K. Lew

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Forecasting on multivariate time series data requires the extraction of patterns in feature sequences. Current state-of-the-art deep neural network (DNN) architectures are capable of high forecasting accuracy, but lack high interpretability due to the complex non-linear nature of neural networks. Prior academic works exist that can extract the trend and seasonality in the input features of cyclical data by using a combination of DNN and statistical methods, but are unable to do so for stochastic data. Recurrent Neural Network (RNN) and Attention based methods have higher spatiotemporal explainability for both types of data, but have worse forecasting performance compared to the benchmarks. Temporal Convolution Networks (TCN) based architectures perform better than RNNs, but have yet to be explored as much for explainability.

In this paper, we introduce the TCNA architecture - using TCN blocks to extract spatiotemporal embeddings and a self-attention encoder layer for further context refinement we achieve performance that is on par with current DNN methods across multiple real-world datasets. Our innovative methods for interpreting the attention matrix allow us to extract temporal patterns directly from cyclical datasets, as well as perform standard regime identification operations on more stochastic data sources.

1 Introduction

With the constant advancements in technology, there has been an exponential increase in time series data generation and collection from varied sources, of both a univariate as well as a multivariate nature. Time series forecasting is used in domains like retail (Kechyn et al., 2018; Lai et al., 2017), medical (Zhang and Nawata, 2018) and finance (Capistrán et al., 2010; Akioyamen et al., 2020). Alongside forecasting performance, an increasing amount of effort is being applied towards making model performance more explainable. Identification of the temporal patterns in the input features and their relative importance is a significant requirement in many domains.

Deep Neural Networks are now considered the state of the art approach for solving most regression and classification tasks, including time series forecasting. Previous approaches involved using sequence to sequence architectures like Recurrent Neural Networks (RNN) (Lai et al., 2017; Zhang and Nawata, 2018), and convolutional architectures like Temporal Convolutional Networks (TCN) (Bai et al., 2018; Wan et al., 2019). In more recent years attention-based mechanisms have been recognized as a valuable addition (Vig, 2019), and when combined with RNNs (Liu et al., 2019; Lim et al., 2021) and TCNs (Fan et al., 2021; Pantiskas et al., 2020) offer additional performance and interpretability.

While forecasting performance of an architecture has generally been a key metric for evaluation and comparison (Makridakis et al., 2020), an increasing amount of work is being done towards increasing model interpretability. Unlike regression or classification, time series forecasting requires interpretation of features in a spatiotemporal manner. Traditional after-the-fact feature importance calculation methods like SHAP (Lundberg and Lee, 2017) and LIME (Ribeiro et al., 2016) calculate the importance of an input feature without factoring in the contribution of each individual time step. Most current methods for temporal pattern analysis require the model architecture to be able to extract the trend and seasonality from the input data (Oreshkin et al., 2019; Challu et al., 2022). However in case of stochastic data sources like those from the finance domain, seasonality might not be very prominent and may require alternatives like regime analysis (Akioyamen et al., 2020; Dias et al., 2015). Regime analysis requires

clustering time step features based on a distance metric, thereby treating each cluster as a significant regime. Using the model to generate embeddings from the time step features allow for better spatial distribution in the embedding space, leading to improved clustering and regime separation (Li et al., 2019).

Combining the insights from above concepts, we propose the Temporal Convolutional Network Attention (TCNA) architecture for multivariate time series forecasting and analysis, using stacked temporal convolutional network blocks to extract embeddings from the features at each time step, followed by a self-attention layer for spatiotemporal pattern inference. First, the forecasting performance of the TCNA architecture is evaluated against the current current state-of-the-art architectures on the Traffic, UberTLC, BikeSharing and Volatility datasets (Lai et al., 2017; Flowers, 2015; Fanaee-T, 2013; Han et al., 2015). This is followed by a demonstration of the temporal interpretability of our architecture on each of the above dataset. The seasonality of all four datasets is extracted, where Traffic, UberTLC and BikeSharing yield significant weekly temporal patterns owing to their inherent cyclic nature. The Volatility dataset, owing to its stochastic nature does not show a temporal pattern over any fixed period of time. However it responds well to regime segmentation using cosine distance for separation between the mean and current time step embeddings, an alternate mode of operation of our model.

The paper from here is organised as follows: Section 2 contains the related work on the concepts used in our approach. Section 3 describes our proposed novel model architecture. Section 4 describes the datasets, followed by the baseline and model architectures used in our benchmark experiments, as well as the training and hyperparameter optimization process. Section 5 presents the various methods employed in our approach towards the explainability of our model. Finally, Section 6 concludes the paper, and discussed potential further research avenues. All code can be found at this github repo.

2 Related work

2.1 Deep Neural Networks in time series forecasting

Deep Neural Networks (DNNs) are currently the standard approach for time series forecasting (Makridakis et al., 2020). Problem statements such as forecasting anomalous demand for certain time periods of the year (Laptev et al., 2017; Kechyn et al., 2018), understanding and separating regimes in the performance of financial instruments (Lim et al., 2021), and predicting disease breakouts based on environmental factors (Zhang and Nawata, 2018) require learning complex non-linear relationships between the input variables across time steps to be learned by the models, something DNNs are well suited for. A large part of the DNN architectures used to forecast time series are sequential in nature, like RNNs and LSTMs (Zhang and Nawata, 2018; Lai et al., 2017), to better model context across time steps. Convolution based methods like TCNs (Bai et al., 2018; Wan et al., 2019; Pantiskas et al., 2020) were later developed to improve over the shortcomings of RNN based architectures, and increase the observable input length for additional context. Simultaneously, attention based methods like Autoformer(Wu et al., 2021) and Informer (Zhou et al., 2020) have been developed using their ability to extract temporal interactions (Vaswani et al., 2017) between non-contiguous time steps. A more recent development is the use of generic DNN layered architectures like NBEATS (Oreshkin et al., 2019) and NHiTS (Challu et al., 2022) to reduce the computational resource requirements and maintain model interpretability.

2.2 Recurrent Neural Networks

RNNs and their derivatives were built with the idea that a context can be developed from a sequential input, allowing for relationships across the time step items to be learnt and used to predict an output of length one or more. They were primarily developed for creating language models (Sundermeyer et al., 2012), but soon found applications in speech recognition and translation (Graves et al., 2013) as well as time series forecasting (Zhang and Nawata, 2018; Lai et al., 2017). One of the advantages of using such sequence to sequence models is that the latent space used as the memory of the model to store context, can also be used to generate embeddings for further context attenuation using transformer based attention mechanisms (Lim et al., 2021). However the performance and versatility of RNN based architectures are limited by some shortcomings. The training and inference process is sequential, where all previous time steps must be evaluated before proceeding to the next, leading to an inability to be parallelized. This sequential nature also results in errors adding up over time steps in the forward pass, and unstable gradients on backpropapagation. These factors combined make it difficult to use longer input sequences with RNN based architectures.

2.3 Temporal Convolutional Networks

Convolutional Neural Network (CNN) based architectures use convolution filters, also known as kernels, to extract patterns in input images (O'Shea and Nash, 2015), finding good use in object detection and classification (Redmon et al., 2016), classifying audio spectrographs(Mustaqeem and Kwon, 2019), and identifying game states in the groundbreaking AlphaGO agent (Silver et al., 2016). The Temporal Convolutional Network (TCN) architecture (Bai et al., 2018) further enhances sequential input processing in CNNs by introducing causality and dilation to the operation. This has the added benefit of reducing memory footprint when stacking TCN blocks, as well as learning both dense local and sparse global context. In a TCN model, the independence in processing of each time step allows for a more stable gradient and parallelization during training, resulting in a higher forecasting performance (Bai et al., 2018). Our approach TCNA uses TCN blocks to convert the input features into time step embeddings. This results in a faster training and better forecasting performance compared to RNN (Lim et al., 2021) based approaches .

2.4 Attention mechanisms

The attention mechanism is a way to extract intra and inter relationships of input and output sequence items. Notably, the transformer architecture (Vaswani et al., 2017) uses encoder-decoder attention blocks to derive relationships between input and output sequences. Transformer architectures have been used extensively for natural language processing (NLP) applications like BERT (Devlin et al., 2018), with the versatility of the attention mechanism allowing its use in other domains like speech synthesis (Baevski et al., 2020), image classification (Wang et al., 2021), and time series forecasting (Wu et al., 2021; Zhou et al., 2020). The attention matrix present in the mechanism maps interaction between the input and output items of the attention block, which lends itself to high interpretability (Vig, 2019), allowing for the visualization of temporal patterns and explanation of variable importance in time series (Lim et al., 2021; Pantiskas et al., 2020)

2.5 Pattern inference

With the development of DNN based state-of-the-art time series mechanisms, the problem of interpretability of the output has become quite pronounced. Key requirements towards interpretability are the visualization of feature importance, as well as trend and seasonality in the temporal domain. Due to the complex non-linear relationships between layer elements, such operations are unable to be easily carried out. Traditional approaches for feature importance calculation like SHAP (Lundberg and Lee, 2017) and LIME (Ribeiro et al., 2016) require the creation of surrogate models that approximate the effect of each feature towards the predictions. This approach performs well on one dimensional input feature vectors, but is unable to factor in the spatiotemporal nature of time series data, where the importance of a feature could be time step dependant.
Methods like NBEATS (Oreshkin et al., 2019) and NHiTS (Challu et al., 2022) use a modified architecture to fit a complex polynomial function to the raw data, extracting a global seasonality at the cost of predictive performance. These models make the assumption that the dataset is inherently cyclic in nature, with a repeating temporal pattern over a fixed time period. When trained on stochastic datasets like those from the finance domain, such trend/seasonality models fail to produce results (Lim et al., 2021). Further, since these architectures treat the derived polynomial function as a representative of the whole dataset, the minor perturbations in temporal patterns are misrepresented, preventing their further analysis. An example of such perturbations in temporal patterns would be reduced demand on a bike sharing system due to adverse weather conditions (Gebhart and Noland, 2014; Eren and Uz, 2020).
Attention based architectures use embedding sequences generated from the input time series through DNN/RNN/TCN mechanisms (Lim et al., 2021; Lai et al., 2017) to extract further temporal context. The attention weights provide a true representation of the input time steps, and are used to visualize temporal patterns in cyclic datasets. For stochastic datasets, the attention weights are interpreted as embeddings, allowing for the utilization of traditional regime clustering approaches to split the time series into meaningful segments (Akioyamen et al., 2020; Dias et al., 2015). As the attention weights correspond to the time steps directly, they can be used to visualize the dominant temporal patterns of the raw data. Our approach (TCNA) uses these abilities of attention mechanisms (self-attention encoder layer) to visualize the dominant temporal pattern in cyclical datasets, and different regimes in stochastic datasets.

3 Temporal Convolutional Network Attention (TCNA)

Our novel proposal TCNA uses state-of-the-art components in the field of time series forecasting such that a well performing model architecture is created that also yields high temporal explainability of the input dataset. The major components of the TCNA architecture that we see from Figure 1 are:

1. **TCN** blocks to extract local as well as global temporal patterns over the input time series, while simultaneously converting them into embeddings.

2. **Self-Attention Encoder** to process the sequence of temporal embeddings and generate an attention matrix of the time steps.

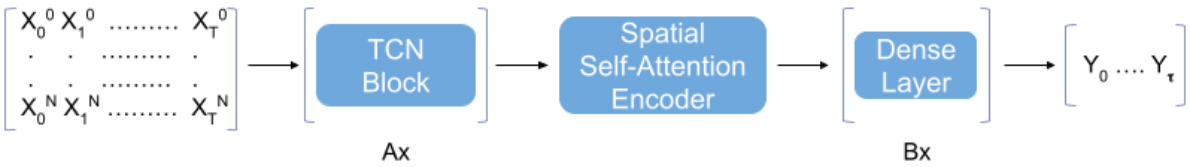
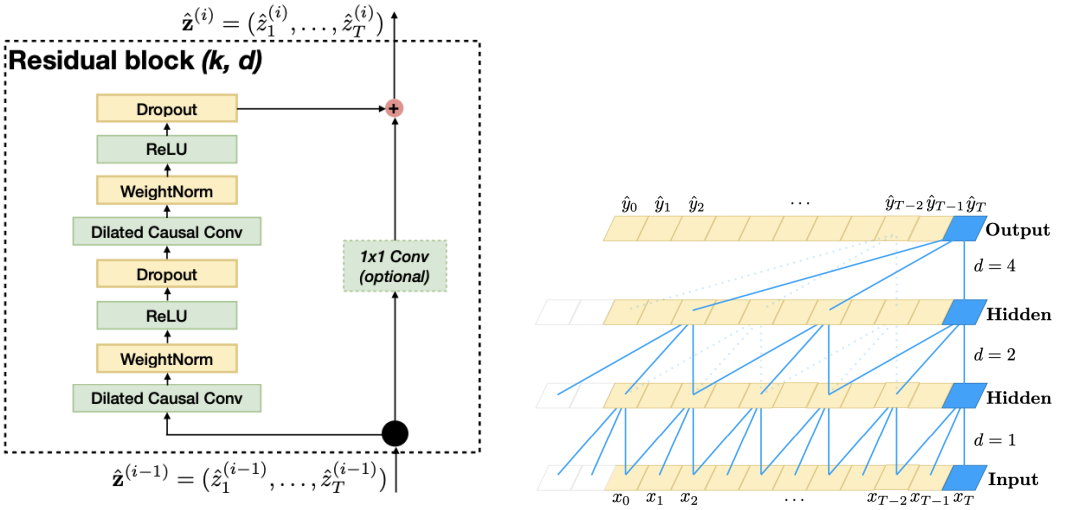


Figure 1: TCNA architecture block diagram

3.1 Temporal Convolutional Network



(a) TCN block architecture showing the dilated caausal convolution and residual connection

(b) Dilation across TCN blocks, increasing exponentially as the number of blocks increases

Figure 2

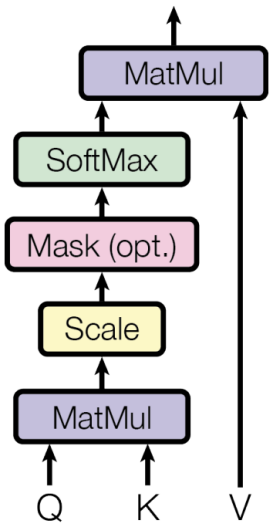
A TCN block, as depicted in Figure 2a, is based on the idea of a 1 dimensional causal convolution. The causality is introduced by padding the input sequence from the left, so as to prevent the kernel from seeing future input time steps and leaking information. Stacking TCN blocks adds predictive power, but also quickly increases memory requirements (Bai et al., 2018). To prevent this, a dilation factor d is instituted across the blocks, with a multiplication factor of 2 being usually applied as shown in Figure 2b. The dilation factor increases the distance between consecutive time steps being operated on by the kernel, decreasing the total number of operations as the number of blocks increases and consequently increasing the receptive field of operation of the kernel. Thus for a filter f of size k, $f \in R^k$, bias b, and dilation factor d operating on a 1-D input $x \in R^L$ of length L, an output $x' \in R^L$ is produced. Equation 1 displays the calculation for the i^{th} element in x' as:

$$x'_i = \sum_{j=0}^{k-1} f_j . x_{i-(d*j)} + b \tag{1}$$

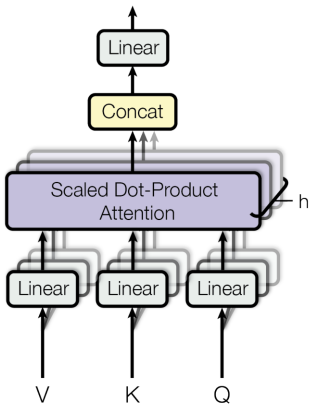
Where $i - (d * j)$ is the padding required to make the operation causal. Thus for cases where $i - (d * j) < 0$, we use zero padding, i.e. $x_{i-(d*j)} = 0$

As shown in Figure 1, the input to the first TCN block is of the shape $N * C_{in} * T$, where N is the number of input features, C_{in} is the corresponding number of input channels per feature, and T is the input sequence length. Consequently the output is of the shape $N * C_{out} * T$. To isolate the operation of the kernels on individual features, we ensure that for a given number of kernels K, the equation $C_{in} * K = C_{out}$ holds true. Thus, for each feature, an embedding stack of size $C_{out} * T$ is generated.
Beyond the convolution, weight normalization is applied on the convolution outputs, followed by a dropout with probability p, and a ReLU activation. Alongside this stack, is a residual connection from the input that is summed to the output of the convolution stack (He et al., 2015).

3.2 Self-Attention



(a) Scaled dot-product attention. Q, K and V are matrices of the sequence length times input embedding size



(b) Multi-head attention splits the Q, K and V into equal subsets and performs the attention process before joining the results back, resulting in more varied context learning

Figure 3

The TCNA architecture uses a self-attention mechanism to accept embedding sequences generated by the TCN blocks and learn temporal relationships across varying distances. The general form of attention as described in (Vaswani et al., 2017) uses a scaled dot-product as shown in Figure 3a, where the input queries and keys of dimension d_k are scaled by $\sqrt{d_k}$, passed through a softmax function and then multiplied by values of dimension d_v. In practice, all queries, keys and values are stored as matrices Q, K and V. Equation 2 describes this operation mathematically as:

$$Attention(Q, K, V) = \frac{softmax(Q.K^T)}{\sqrt{d_k}}V \tag{2}$$

To emphasize temporal patterns as well as feature importance from the model inputs, we transform the output of the final TCN block and pass it to separate temporal and spatial attention encoders. Recall that the input to the TCN stack is N features of length T with

C_{in} equal to 1. With each TCN block operating with K kernels, the output of the final block is $(N * C_{out} * T)$ where $C_{out} = K * C_{in}$. For the temporal attention layer, the TCN output is transformed into a sequence of length T and width $N * C_{out}$. For the spatial attention layer, the TCN output is transformed into a sequence of length N and width $C_{out} * T$. The outputs of the spatial and temporal attention encoders are converted back to the dimensions $(N * C_{out} * T)$, added to the output of the last TCN block and passed through a series of dense layers, the output of which is the predicted sequence of variable forecast horizon

We also employ a multi-head attention to learn different relationship dynamics across the input embedding sequences, as shown in Figure 3b. Each head learns a different representation of the input sequence, with the resultant diversity helpful in inferring temporal patterns better (Vaswani et al., 2017). Multi-head attention for h heads is described by equations 3 and 4 as:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^0 \tag{3}$$

where

$$head_i = Attention(QW_i^Q, VW_i^V, VW_i^V) \tag{4}$$

Here the query, key and value matrices for a particular head $head_i$ are $W_i^Q \in R^{d_{embedding} * d_k}$, $W_i^K \in R^{d_{embedding} * d_k}$ and $W_i^K \in R^{d_{embedding} * d_v}$, with $d_{embedding}$ as the embedding size for each element in the input sequence to the attention layer. For a given number of heads h, the dimensions of the query, key and value matrices are $d_k = d_v = d_{embedding}/h$.

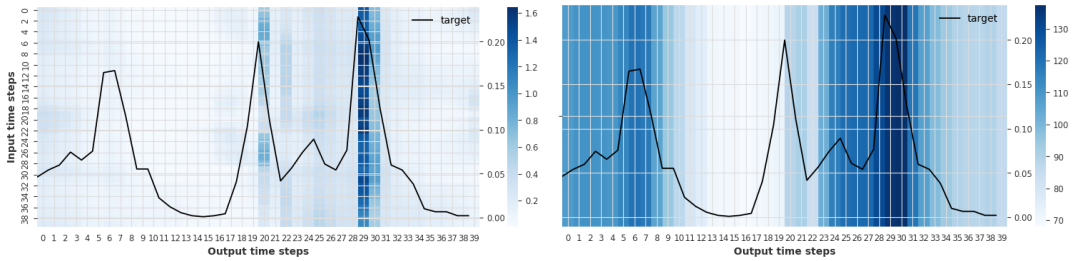
3.3 Attention inference

Using the attention weights from the self-attention encoder layer mentioned in section 3.2, we calculate the relative importance of each input time-step with respect to the forecasted output. For a given multivariate input sequence $x \in R^{N*T}$ of N variables and T time steps, the corresponding attention weights matrix is $\alpha \in R^{T*T}$. The attention weights are converted to a 1-D vector $\overline{\alpha}$ through the dot product of the attention weights matrix α with a 1-D matrix of ones of length T as described in equation 5

$$\begin{vmatrix} \overline{w_0} \\ \vdots \\ \overline{w_T} \end{vmatrix} = \begin{vmatrix} w_0^0 & \dots & w_T^0 \\ \vdots & \ddots & \vdots \\ w_0^T & \dots & w_T^T \end{vmatrix} . \begin{vmatrix} 1 \\ \vdots \\ 1 \end{vmatrix} \tag{5}$$

Figure 4a shows us an attention matrix for a sample of length $T = 60$, where darker regions show more weight attached to the relationship between the corresponding input and output time steps. The attention vector for the same is shown in Figure 4b, with the darker time step regions having more importance. To further refine the temporal patterns, the attention vectors of each input feature set of the dataset are summed (Lim et al., 2021), with each consequent vector being shifted to the right to align the patterns in the vector.

The above technique will however not work for a stochastic dataset due to the inherent randomness. However, by treating the attention vector at each time step as a representative embedding, the dataset can be segmented based on standard regime identification methods (Dias et al., 2015; Akioyamen et al., 2020)



(a) Attention matrix α (b) Attention vector $\overline{\alpha}$

Figure 4: Attention matrix and vector of a sample input of length $T = 60$. Target variable is displayed as the black line across X axes

4 Performance Evaluation

4.1 Datasets

To compare the performance of our model architecture against earlier work in time series forecasting, we use the following datasets that were previously used in comparisons with our baseline models.

1. **Traffic:** The UCI California Department of Transportation road occupancy dataset has hourly sensor data measuring traffic occupancy at 861 points in the San Francisco bay area (Lai et al., 2017). There are 861 occupancy sensor readings, each in the range of 0 and 1. The sensor aggregation cadence is hourly, with data collected between the years 2015 and 2016. We select a target with the highest variance in the dataset, as this allows for the best visualization of temporal patterns. For our experiments, we use sensor 617 as the target, and sensors 632, 26, 562, 410 and 442 as covariates due to their high covariance with the target, along with hour of the day and day of the week.

2. **UberTLC:** The Uber TLC dataset was obtained by FiveThirtyEight, containing 14.3 million Uber trip details between January to June 2015 (Flowers, 2015). The subset available in Darts contains hourly pickup counts across 262 locations. As in the Traffic dataset, we select a target with the highest variance in the dataset, as this allows for the best visualization of temporal patterns. For our experiments, we use location ID 140 as the target, with ID 236, 237, 233, 141 and 238 as covariates due to their high covariance with the target, along with hour of the day and day of the week.

3. **Bike Sharing:** The UCI Bike Sharing dataset contains the hourly and daily counts of bike rentals between the year 2011 and 2012, in the Capital Bikeshare system (Fanaee-T, 2013). Also included are weather and seasonal information at an hourly and daily frequency. For our experiments, the hourly number of non subscriber bike rentals is the target, with weather and season data as covariates, along with hour of the day, day of the week and month of the year.

4. **Volatility:** The OMI Realized Library dataset contains data for 31 stock indices at a daily frequency (Han et al., 2015). Columns include (but are not limited to) the open and close price for the day, daily returns and mean realised volatility across multiple time frames. For our experiments, the close price for the SPX index is used as the target,

with the mean realized volatility as covariates. The realized volatility is calculated over a rolling 5 and 10 minute time frame, and the daily mean is used in our experiments. Also included are the day of the week and day of the year.

All time features are encoded cyclically to sin/cos values. An example on an input feature x_T with length T is given in equations 6 and 7.

$$x_T^{sin} = sin(\frac{2 * pi * x_T}{max(x_T)}) \tag{6}$$

$$x_T^{cos} = cos(\frac{2 * pi * x_T}{max(x_T)}) \tag{7}$$

where $max(x_T)$ is the upper bound on values possible in the input.
All features are then scaled with the MinMaxScaler in Scikit-learn (Pedregosa et al., 2011) to the range 0-1 using the equations 8 and 9:

$$X\_std = (X - min(X)/(max(X) - min(X) \tag{8}$$

$$X\_scaled = X\_std * (range\_max - range\_min) + range\_min \tag{9}$$

where $range\_min = 0$ and $range\_max = 1$
Each dataset is used to study the performance of our architecture over key aspects. The Traffic, UberTLC and Bike Sharing datasets are used to establish performance over homogenous and heterogenous input features with respect to prior academic work. The Volatility dataset is used to demonstrate robustness towards overfitting on a smaller, noise filled heterogenous dataset that lacks inherent temporal patterns. In terms of interpretability, the Volatility dataset is used to demonstrate the models ability to perform regime identification, while the other three datasets are cyclical and used to show the ability of the attention mechanism to extract repeating temporal patterns.

4.2 Training and hyperparameter optimization

Training was performed on a single CUDA capable Nvidia 3060 GPU with 3584 cores and 12 gigabytes of VRAM. Each dataset is split into a training (70%) and validation (20%) set for learning and hyperparameter optimization, and a testing (10%) set for final performance evaluation. Hyperparameter optimization is conducted through the Optuna framework (Akiba et al., 2019) using the default Tree Parzen Estimator sampler (Bergstra et al., 2011, 2013) over the hyperparameter sample space. The hyperparameter optimization is performed for 30 trials, with each trial model being trained over 20 epochs. Following this, each architecture is trained for 30 epochs on all datasets across five seed values for mean error and standard deviation. The metric used for comparison is Mean Squared Error, as denoted in equation 10 for a dataset with D samples:

$$\sum_{\mathbf{i=0}}^{\mathbf{D}} (\mathbf{y\_target_i} - \mathbf{y\_predicted_i})^{\mathbf{2}} \tag{10}$$

where for the i^{th} sample in the dataset, y_target_i is the value to be forecasted, and $y\_predicted_i$ the model prediction.

Dataset	Bike Sharing		Traffic	
MSE	mean	std dev	mean	std dev
NBEATS	0.001264	0.000102	0.008892	0.000944
NHiTS	0.002593	0.000475	0.008281	0.000635
TCN	**0.001013**	0.000037	**0.007155**	0.000103
TFT	0.001600	0.000173	0.010782	0.000469
TCNA	0.001221	0.000083	0.041007	0.045317
Dataset	UberTLC		Volatility	
MSE	mean	std dev	mean	std dev
NBEATS	0.007675	0.002634	0.002156	0.001301
NHiTS	0.006562	0.001837	0.004477	0.006104
TCN	0.006805	0.000408	**0.000155**	0.000098
TFT	0.011979	0.001501	0.043094	0.013341
TCNA	**0.006209**	0.000588	0.023529	0.015176

Table 1: Mean Squared Error mean loss and standard deviation (lower is better) on BikeSharing, Traffic, UberTLC, and Volatility datasets

4.3 Benchmarks and results

We compare TCNA to a range of models for multi horizon multivariate forecasting, based on desired characteristics defined in Section 2. All baseline model implementations are from the Darts library (Herzen et al., 2022), for ease of comparison and benchmarking. The NBEATS (Oreshkin et al., 2019) and NHiTS (Challu et al., 2022) architectures are chosen to emulate the performance of models using generic DNN layers. The TCN block (Bai et al., 2018) is used to make the TCNA architecture, and thus warrants comparison. Lastly, the Temporal Fusion Transformer (TFT) architecture demonstrates the combination of LSTM and attention layers (Lim et al., 2021).
From the results of the benchmarking in Table 1, we see that the TCN model is the best in the Bike Sharing, Traffic and Volatility datasets. The TCNA model performs the best for the UberTLC dataset and comes in second for Bike Sharing. UberTLC and BikeSharing have distinct weekly patterns, with the latter having addition heterogenous features like weather, aiding in their predictability. The Volatility dataset is particularly difficult on attention based models like TFT and TCNA, due to the stochastic nature of the the features. Attention mechanisms work better when more deterministic inputs like those from the NLP domain are used, and not on the time series domain. However, as we shall see in Section 5, the tradeoff in performance is made up with respect to interpretability.

5 Model interpretability

Through our experiments in Section 4, we have established that the TCNA architecture is on par with prior academic work (in most cases) in time series forecasting with respect to forecasting performance. We now move on to the main aspect of our architecture, which is interpretability. The TCNA architecture can extract temporal patterns in two types of situations.

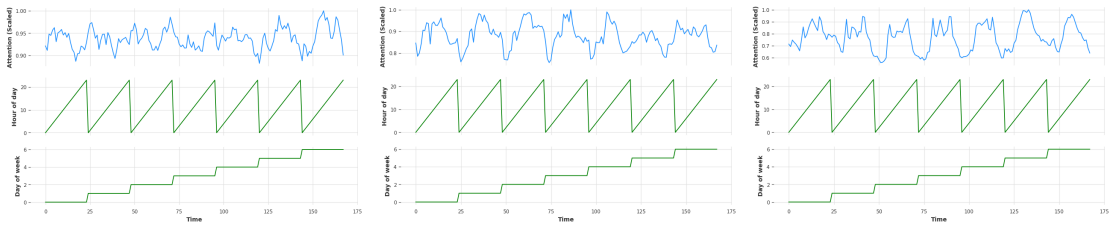
1. Extracting seasonality in cyclical datasets.

2. Identifying significant regime shifts in stochastic datasets.

Attention interpretation mechanisms like (Li et al., 2019; Pantiskas et al., 2020) offer the tools to investigate niche anomalies in specific samples, but fail to examine global patterns. Meanwhile, methods like (Lim et al., 2021) are capable of examining global scale temporal patterns, but the use of RNNs to generate the embeddings from the time steps limits the length of the context that the model can learn. The combination of TCN blocks and self-attention encoders in TCNA allow for the learning of longer context in more detail, as well as the extraction of seasonality.

5.1 Seasonality

Architectures that approximate seasonality and trend by fitting a polynomial function to the input dataset (Oreshkin et al., 2019; Challu et al., 2022) are able to show the overall temporal patterns while missing out on long term perturbations due to additional factors. An example of this would be reduced demand for bike rentals due to adverse weather conditions (Gebhart and Noland, 2014; Eren and Uz, 2020). TCNA has a dedicated self-attention encoder for learning temporal context, that we use to extract attention matrices α for each time-step. The flattened attention vector $\overline{\alpha}$ reveal the relative importance of input time steps in each sample. By shifting $\overline{\alpha}$ for each consequent sample in the dataset and summing, the global attention pattern is derived. To generate the seasonality for a fixed time period, such as over the days of a week, the global attention pattern is divided into lengths of the fixed time period. The summation and scaling of this windowed attention results in the seasonality learned by the self-attention encoder.



(a) Traffic (b) UberTLC (c) Bike Sharing

Figure 5: Seasonality over days of the week for Traffic, UberTLC and BikeSharing datasets . Resolution across X axes is hourly.

In Figure 5 we see the seasonality over the days of the week for the Traffic, UberTLC and Bike Sharing datasets. The topmost plot is the scaled attention for each time-step. The middle plot is the hour of the day in the range 0 to 23. The bottom plot is to indicate the day of the week, with Monday as 0 and Sunday as 6. From Figure 5c and 5b, we see that the attention vector mimics the broader movements of the target variable in Bike Sharing and UberTLC show clear daily trends. The difference in weekday vs weekend patterns is intuitive, as customers are likely to vary their usage of transport services during the week. From Figure 5a, we see the Traffic dataset has more sudden peaks in its sensor data, which prevents the model from learning more discernable patterns over the weekdays. But here too, the weekend patterns are distinct from the weekday. This ability to visualize the learned self-attention allows for validation of the model performance and learning direction.

5.2 Regime identification

Extracting seasonality from the attention vectors with respect to the input features yields good results when the dataset is inherently cyclical in nature. For a stochastic dataset like price data for a security, this does not hold true. Regime identification however is a technique that allows us to segment the time series into distinct segments(Akioyamen et al., 2020), and would work well for such datasets. We use the attention matrix α as a unique representation of each time-step, and perform traditional distance based operations to distinguish between the different regimes . In the simplest cases, the mean of the time-step embeddings is treated as the anchor point, and the distance to all time-step embeddings is measured. All time steps that are below a certain meaningful threshold in terms of distance are treated as the normal regime, while those above the threshold belong to a different regime (Lim et al., 2021).
In our approach, we define the average attention matrix for D samples as in equation 11:

$$\alpha^{avg} = \frac{\sum_{t=1}^{D} \alpha_t}{D} \tag{11}$$

Cosine similiarity as given in equation 12 calculates the cosine of the angle between two n-dimensional vectors in the range of 0 and 1, where 0 means the two vectors are completely dissimiliar, and 1 indicates complete similiarity. The cosine distance metric as show in equation 13 is the inverse of cosine similiarity:

$$CosineSimiliarity(A, B) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} \tag{12}$$

$$CosineDistance(A, B) = 1 - CosineSimiliarity(A, B) \tag{13}$$

where A and B are two high dimension vectors. We use cosine distance as metrics like Euclidean distance begin to fail in higher dimensions due to loss of significance (Mirkes et al., 2020).

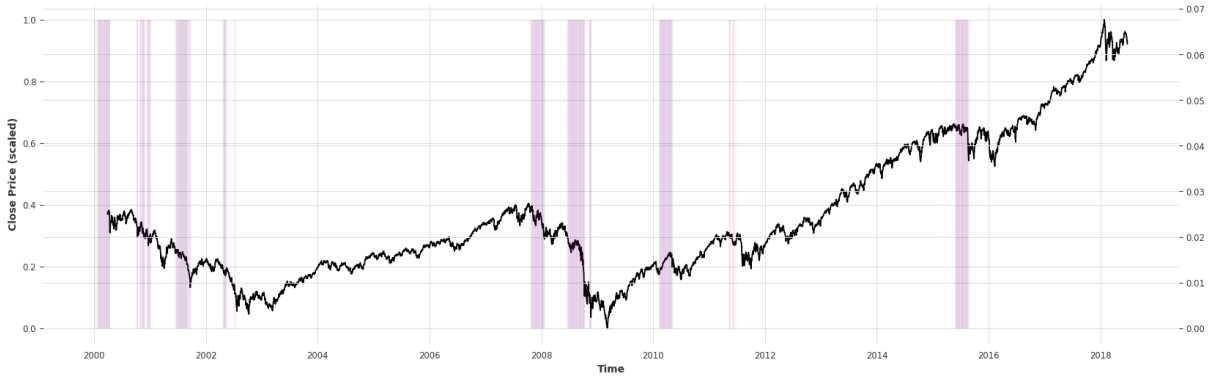


Figure 6: Regime identification in Volatility dataset. The scaled SPX close-price is plotted across time. Purple regions indicate the highly volatile regime, while white regions are less volatile

In Figure 6, we demonstrate regime identification in the Volatility dataset, with the SPX index close-prices (scaled) plotted across time. All time steps with cosine distance more than 90^{th} percentile distant from α^{avg}, are indicated with the vertical purple region as a volatile regime. We see that the Dot-Com crash of the early 2000 and the Sub-Prime lending crisis of 2008 are highlighted as periods of high volatility, while the bull markets of 2004 to 2007 and 2012

onwards are relatively normal. From Figure 7 we see the effect the two regimes have on the consolidated attention vectors. The normal regime on the left has more evenly spread out attention, while volatile regions on the right have spiked attention. This is in line with other attention based architectures (Lim et al., 2021) that indicate that the sharp trend changes require attention to more specific time steps.

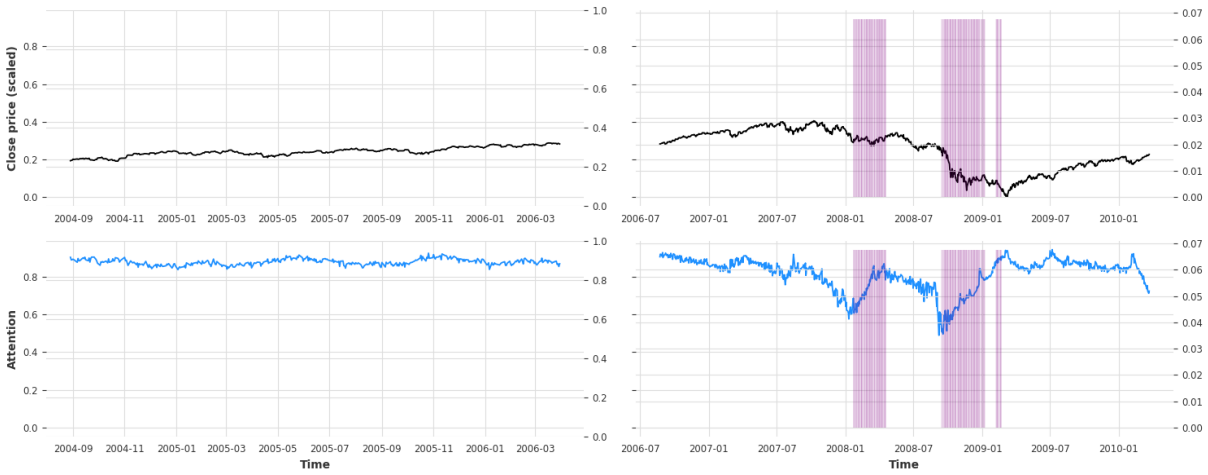


Figure 7: Closer look at a normal (left) and volatile (right) regime. The top graphs are for the scaled close price plotted against time, the bottom graphs are the consolidated attention vectors on the same time axis. Purple regions indicate the highly volatile regime, while white regions are less volatile

6 Conclusion

We introduce TCNA, a novel architecture built from current state-of-the-art TCN and attention encoder blocks for multivariate multi-horizon time series forecasting. We show that the architecture is capable of operating with a wide variety of inputs, and perform on par with other current DNN methods of time series forecasting for many domains. Keeping in mind the push away from black-box models and towards higher model explainability, the TCNA architecture has an innovative mechanism for visualizing the local and global context that the model has interpreted from the input features. The self-attention encoder outputs can be manipulated to visualize seasonality in cyclical datasets and perform regime identification on stochastic data, without modifications to the model architecture that may compromise performance. The benchmarking results reveal that the performance of the TCNA architecture has space for improvement. Future work can be done towards performing an ablation study to see which parts of the architecture are reducing performance. It would then be possible to use those parts only for interpretability, disabling them for maximum performance. Studying more datasets could further confirm which types of features and domains work best with the architecture. Finally, integrating the means to compute feature importance would unlock further interpretability, as then we would be able to visualize the interaction of the features across time.

References

Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Akioyamen, P., Y. Z. Tang, and H. Hussien (2020, oct). A hybrid learning approach to detecting regime switches in financial markets. In *Proceedings of the First ACM International Conference on AI in Finance*. ACM.

Baevski, A., Y. Zhou, A. Mohamed, and M. Auli (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems 33*, 12449–12460.

Bai, S., J. Z. Kolter, and V. Koltun (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR abs/1803.01271*.

Bergstra, J., R. Bardenet, Y. Bengio, and B. Kégl (2011). Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*, Volume 24. Curran Associates, Inc.

Bergstra, J., D. Yamins, and D. Cox (2013, 17–19 Jun). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In S. Dasgupta and D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning*, Volume 28 of *Proceedings of Machine Learning Research*, Atlanta, Georgia, USA, pp. 115–123. PMLR.

Capistrán, C., C. Constandse, and M. Ramos-Francia (2010). Multi-horizon inflation forecasts using disaggregated data. *Economic Modelling 27*(3), 666–677.

Challu, C., K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler, and A. Dubrawski (2022). N-hits: Neural hierarchical interpolation for time series forecasting. *CoRR abs/2201.12886*.

Devlin, J., M. Chang, K. Lee, and K. Toutanova (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805*.

Dias, J. G., J. K. Vermunt, and S. Ramos (2015). Clustering financial time series: New insights from an extended hidden markov model. *European Journal of Operational Research 243*(3), 852–864.

Eren, E. and V. E. Uz (2020). A review on bike-sharing: The factors affecting bike-sharing demand. *Sustainable Cities and Society 54*, 101882.

Fan, J., K. Zhang, Y. Huang, Y. Zhu, and B. Chen (2021, may). Parallel spatio-temporal attention-based TCN for multivariate time series prediction. *Neural Computing and Applications 35*(18), 13109–13118.

Fanaee-T, H. (2013). Bike Sharing Dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5W894.

Flowers, A. (2015). *Uber TLC Foil Response dataset*. https://github.com/fivethirtyeight/uber-tlc-foil-response [Accessed: Whenever].

Gebhart, K. and R. B. Noland (2014). The impact of weather conditions on bikeshare trips in washington, dc. *Transportation 41*, 1205–1225.

Graves, A., N. Jaitly, and A.-r. Mohamed (2013). Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 273–278.

Han, H., M. D. Park, and S. Zhang (2015). A multiplicative error model with heterogeneous components for forecasting realized volatility. *Journal of Forecasting 34*(3), 209–219.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *CoRR abs/1512.03385*.

Herzen, J., F. LÃ¤ssig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. V. Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, M. Dumonal, J. KoÅ›cisz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch (2022). Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research 23*(124), 1–6.

Kechyn, G., L. Yu, Y. Zang, and S. Kechyn (2018). Sales forecasting using wavenet within the framework of the kaggle competition. *CoRR abs/1803.04037*.

Lai, G., W. Chang, Y. Yang, and H. Liu (2017). Modeling long- and short-term temporal patterns with deep neural networks. *CoRR abs/1703.07015*.

Laptev, N. P., J. Yosinski, L. E. Li, and S. Smyl (2017). Time-series extreme event forecasting with neural networks at uber.

Li, S., X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Volume 32. Curran Associates, Inc.

Lim, B., S. Ö. Arık, N. Loeff, and T. Pfister (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting 37*(4), 1748–1764.

Liu, Y., C. Gong, L. Yang, and Y. Chen (2019). DSTP-RNN: a dual-stage two-phase attention-based recurrent neural networks for long-term and multivariate time series prediction. *CoRR abs/1904.07464*.

Lundberg, S. M. and S.-I. Lee (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, Volume 30. Curran Associates, Inc.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos (2020). The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting 36*(1), 54–74. M4 Competition.

Mirkes, E. M., J. Allohibi, and A. Gorban (2020, Sep). Fractional norms and quasinorms do not help to overcome the curse of dimensionality. *Entropy 22*(10), 1105.

Mustaqeem and S. Kwon (2019, Dec). A cnn-assisted enhanced audio signal processing for speech emotion recognition. *Sensors 20*(1), 183.

Oreshkin, B. N., D. Carpov, N. Chapados, and Y. Bengio (2019). N-BEATS: neural basis expansion analysis for interpretable time series forecasting. *CoRR abs/1905.10437*.

O'Shea, K. and R. Nash (2015). An introduction to convolutional neural networks. *CoRR abs/1511.08458*.

Pantiskas, L., K. Verstoep, and H. Bal (2020). Interpretable multivariate time series forecasting with temporal attention convolutional neural networks. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1687–1694.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016, June). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ribeiro, M. T., S. Singh, and C. Guestrin (2016). "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, New York, NY, USA, pp. 1135–1144. Association for Computing Machinery.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature 529*(7587), 484–489.

Sundermeyer, M., R. Schlüter, and H. Ney (2012). LSTM neural networks for language modeling. In *Proc. Interspeech 2012*, pp. 194–197.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need. *CoRR abs/1706.03762*.

Vig, J. (2019). A multiscale visualization of attention in the transformer model. *CoRR abs/1906.05714*.

Wan, R., S. Mei, J. Wang, M. Liu, and F. Yang (2019). Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics 8*(8).

Wang, W., E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao (2021, October). Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 568–578.

Wu, H., J. Xu, J. Wang, and M. Long (2021). Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *CoRR abs/2106.13008*.

Zhang, J. and K. Nawata (2018). Multi-step prediction for influenza outbreak by an adjusted long short-term memory. *Epidemiology Infection 146*(7), 809–816.

Zhou, H., S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang (2020). Informer: Beyond efficient transformer for long sequence time-series forecasting. *CoRR abs/2012.07436*.