Temporal Tensor Transformation Network for Multivariate Time Series Prediction

Yuya Jeremy Ong, Mu Qiao, Divyesh Jadav

IBM Research - Almaden, San Jose, CA, USA

Emails: yuyajong@ibm.com, {mqiao, divyesh}@us.ibm.com

Abstract—Multivariate time series prediction has applications in a wide variety of domains and is considered to be a very challenging task, especially when the variables have correlations and exhibit complex temporal patterns, such as seasonality and trend. Many existing methods suffer from strong statistical assumptions, numerical issues with high dimensionality, manual feature engineering efforts, and scalability. In this work, we present a novel deep learning architecture, known as Temporal Tensor Transformation Network, which transforms the original multivariate time series into a higher order of tensor through the proposed Temporal-Slicing Stack Transformation. This yields a new representation of the original multivariate time series, which enables the convolution kernel to extract complex and non-linear features as well as variable interactional signals from a relatively large temporal region. Experimental results show that Temporal Tensor Transformation Network outperforms several state-of-the-art methods on window-based predictions across various tasks. The proposed architecture also demonstrates robust prediction performance through an extensive sensitivity analysis.

Index Terms—multivariate time series, deep learning, prediction, convolution, tensor transformation

I. Introduction

Multivariate time series prediction has gained wide spread adoptions in various fields and domains, including modeling financial markets [1], meteorology [2], and energy demand forecasting [3]. These algorithms model and predict future time series values based on their historical temporal patterns found amongst multiple variables within the dataset. Although various methods have been proposed to predict multivariate time series based on classical statistical modeling and deep neural networks, each method presents some key challenges when preprocessing and extracting the right set of features when working with multivariate features.

Classical statistical models, such as Vector Auto Regression (VAR), assume that the time series is stationary, i.e., the summary statistics of data points are consistent over time [4]. Thus, preprocessing procedures are usually required to remove trend, seasonality, and other time-dependent structures from the raw series in order to make the data stationary. Furthermore, these models also assume the independence condition in the underlying linear regression problem, i.e., the random errors in the model are not correlated over time [4]. Autocorrelation and partial autocorrelation functions are usually applied to identify the appropriate order of variables. Constructing statistically meaningful prediction models requires performing various preprocessing, transformations, and feature engineering, which

are both time consuming and difficult to scale in practical settings.

On the other hand, time series prediction models based on deep learning can be categorized into two approaches - stateless and stateful models. Stateful models, such as Recurrent Neural Networks (RNNs), are models where the internal weights account for each of the temporal state of the data. In contrast, stateless models, such as Convolutional Neural Networks (CNNs), do not explicitly account for the temporal states of the dataset. In particular, CNNs have been utilized as a method to directly extract features from raw time series and generate predictions, or as part of a downstream feature extraction process within a larger neural network architecture [5] [6].

When CNNs are utilized within a multivariate time series model, there are two primary ways the convolution operation can be applied to the data. One method convolves the input data over the temporal-variable axis, where one of the kernel dimensions are fixed to the total number of variables in the input matrix data [6]. The other method considers each variable as a separate 1D convolution, and correspondingly convolves the data over multiple channels for each feature [7]. However, a caveat to the former approach of utilizing CNNs assumes that the feature representation of the input matrix data is only able to observe a narrow set of variable interactions within the convolutional kernel. Alternatively, the latter convolution operation where we consider each feature in the time series as an independent channel also suffers from similar issues where the kernel is only able to observe a small local-window of time steps as its filters convolve over the data. In both scenarios, the convolution operations for both methods render a limited perspective of the time series. Different from traditional image-based use of CNNs, where objects in different regions could be quite distinct, time series data tend to be relatively homogeneous. The prediction of future values depend more on the global pattern within the historical time window, rather than a local pattern.

To address the two limitations presented above from both the classical statistical and deep learning methods, we propose a novel neural network architecture for multivariate time series prediction with a new class of transformation functions, known as *temporal-slicing stack transformation*. These operations transform the original input data structure into a higher-order tensor, where the individual features in the time series are rearranged from a 1D temporal sequence into a 2D matrix.

This transformation expands the view of the receptive field. As a result, the convolution is operated on a temporally larger region, which may help capture time-dependent features such as trend and seasonality, as well as variable interactions in a longer range. Furthermore, as these operations can be learned within an end-to-end manner, this allows the transformation process to be performed jointly with the downstream time series prediction task.
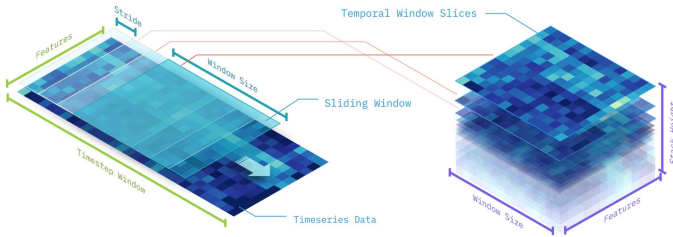

Fig. 1. Visualization of the Temporal-Slicing Stack Transformation process

The temporal-slicing stack transformation is demonstrated in Figure 1, where a window is utilized to slide over the 2D time series data and extract a collection of time series slices. These slices are then correspondingly stacked on top of each other sequentially to form a 3D tensor. Specifically the three dimensions are "features", "time", and "stack". For multivariate time series, the transformation will convert a 2D time series into a 3D tensor. For univariate time series, the transformation will render a 2D matrix. The resulting structure yields an emergent pattern, where a spatial feature extractor, such as CNN, can explicitly model complex and nonlinear autocorrelational features. To illustrate this, we use the transformation of univariate time series as an example. Figure 2 shows the transformation results of univariate time series from four different data sets. We also show their corresponding autocorrelation plots. The transformation demonstrates strong patterns for time series of high autocorrelations. Comparing with the original 1D time series, the 2D matrix has much richer information and therefore may enable more informative feature extraction.

The rest of the paper is organized as follows. We highlight various statistical and deep learning approaches for time series analysis in Section II. Definitions of the temporal tensor transformation, the temporal-slicing stack transformation, and the proposed model architecture are described in Section III. The experimental setup, data sets, and results are presented in Section IV. In Section V, we perform sensitivity analysis based on experiments with synthesized datasets. We finally conclude and discuss future directions in Section VI.

II. RELATED WORK

In this section, we review the prior work for multivariate time series prediction. We first introduce some of the known classical statistical methods and then present several state-of-the-art deep learning methods for time series modeling.

Time series prediction has been well studied with various statistical modeling methods for both univariate and multi-variate data. For univariate time series modeling, the Auto Regressive Integrated Moving Average (ARIMA) model is often applied using the Box-Jenkins modeling process [4]. Other variants of the ARIMA models have been proposed to model temporal patterns from the data, for example, seasonality (SARMA) and coefficient dependent periodicity (PARMA) [8]. Furthermore, methods combining ARIMA and neural networks have also been developed to model both linear and non-linear dynamics [9]. For multivariate time series prediction, vector based methods, such as Vector Auto Regression (VAR) methods, extend the Auto Regressive (AR) models for univariate time series modeling [4] [10]. Variants of the VAR model have also been proposed, such as VARMAX [11] and VARX [12], where the model subsumes properties of the original VAR model and jointly learns interactions between the given variables. Additionally, Gaussian Process [13], as a non-parametric statistical model, is used to predict over a distribution of continuous variables, as opposed to the aforementioned parametric methods. Statistical models often have very high computational complexity and face numerical issues, when the number of variables in the time series is high.

With the advent of deep learning, new neural network architectures have been proposed for multivariate time series prediction. Borovykh et al. [14] developed a multivariate time series model based on the WaveNet architecture [15], which is originally designed for speech audio signal processing. They augment the original architecture by simplifying and optimizing its core algorithms with the dialated convolution to capture long-term multivariate temporal data with noisy signals. Another multivariate time series modeling framework is proposed by Lai et al. [6], namely LSTNet, which combines CNN and RNN to extract hierarchical short-term and long-term temporal dependencies from the time series. In addition to model dependencies, LSTNet also accounts for the autoregressive component of the model as a residual connection between the CNN and LSTM components. Qin et al. [16] develop a dual-stage attention-based neural network architecture for multivariate time series modeling, which utilizes two sets of RNN as an encoder-decoder based architecture. Through each stage of the attention-based network architecture, the method attends to both the feature and temporal dimensions to adaptively select the relevant driving series.

III. MODEL ARCHITECTURE

In this section, we present the architecture of the proposed *Temporal Tensor Transformation Network*. First, we introduce the key notations used in the paper as well as the formal problem definition for multivariate time series prediction. We then introduce the *Temporal Tensor Transformation* operation. Finally, we present the proposed neural network architecture.

A. Notations

We use x to denote a one-dimensional vector, X to denote a two-dimensional matrix, and $\mathcal{X}$ to represent a three-dimensional tensor. Scalars with respect to their corresponding
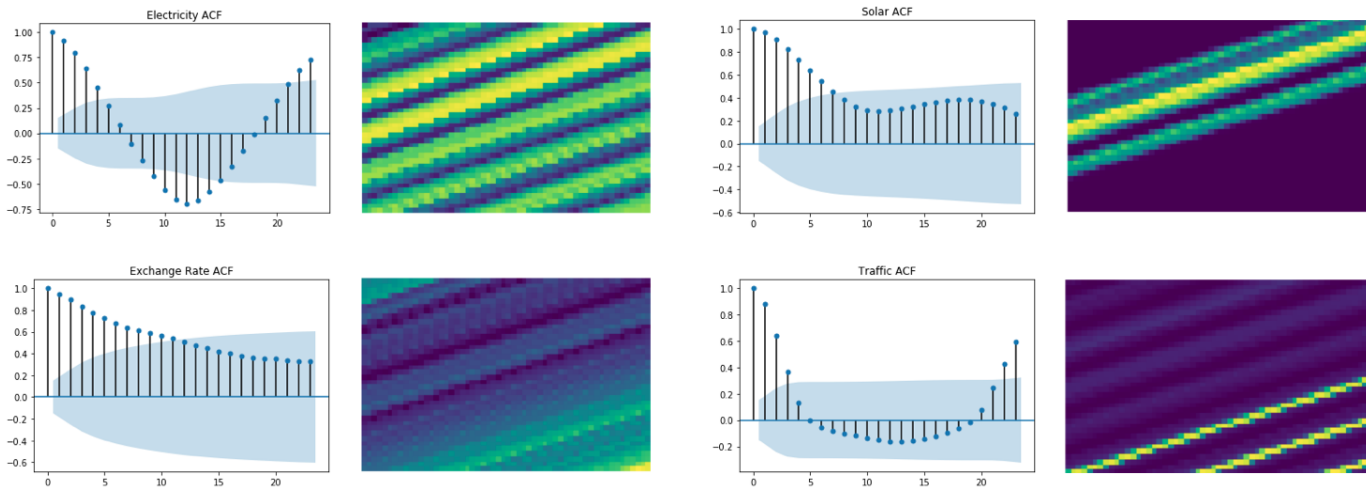
Fig. 2. Comparison of the autocorrelation plots and the resulting Temporal Tensor Transformations

indices are denoted by lower-case letters, followed by subscripted letter triplets. For instance, $x_{i,j,k}$ of a 3D-tensor $\mathcal{X}$ indicates the *(i, j, k)-th* scalar value of the tensor $\mathcal{X}$. Further notation and variables will be introduced and defined as they appear in the context.

B. Problem Definition

We now formally define the multivariate time series prediction problem. Given a complete multivariate time series dataset $X = \{x_1, x_2, ..., x_n\}$, where $x \in \mathbb{R}^{m \times n}$, m is the number of features, and n is the total number of time steps in the multivariate series, the objective is to predict a future series of values up to a defined horizon window. Specifically, given a subset of the time series up to time step T, our model, denoted as function $F(\cdot)$, will take a time series $X_T = \{x_1, x_2, ..., x_T\} \in \mathbb{R}^{m \times T}$ as input, and subsequently generate an output sequence of $\hat{Y}_{T+h} \in \mathbb{R}^{m \times h}$, where h is the horizon window. Hence, the model can be formulated as the mapping function $F(X_T) = \hat{Y}_{T+h}$.

C. Temporal Tensor Transformation

The proposed transformation augments the original data by adding a single higher-order dimension to the original data input. For example, if the input series data is a 1D vector of values (i.e., univariate time series), the transformed data structure will be a 2D matrix structure. Likewise, if the input is a 2D matrix (i.e., multivariate time series), the resulting structure will be a 3D tensor.

We define the Temporal Tensor Transformation as a mapping function $TT : X \rightarrow \tilde{\mathcal{X}}$, where $X \in \mathbb{R}^{m \times T}$ is the input multivariate time series and the resulting transformation generates a 3D tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{m \times \omega \times o}$. Here, ω is the slice window size (i.e., the number of steps within a time window), and o is the number of slices or the stack height of the resulting transformed tensor. The value of o can be computed based on the values of hyperparameters. Our specific temporal tensor transformation is referred to as the

Temporal-Slicing Stack Transformation. Before introducing the Temporal-Slicing Stack operation, we will first introduce the hyperparameters involved in the slicing process.

The *window size*, denoted by ω, defines the overall slicing window of the transformation function, and determines one of the major dimensions of the output tensor. The length of the slicing window is fixed by the number of features in the time series, i.e., m. Therefore, the resulting slicing window is of dimension $m \times \omega$. The *stride* parameter of the slicing window, denoted by s, indicates how many time steps to advance the slicing window along the temporal dimension. The greater the stride, the lower the overall pattern resolution, due to the loss of information among contiguous values within the time series. The *dilation* parameter d is similar to the parameter introduced by Yu et al. [17] for dilated CNN, which allows us to slice a wider receptive window size without compromising to limited memory space. The *padding* value p is similar to how CNN inherently increases the dimensionality of the original data structure. It allows for shift-invariant transformations as well as retaining the dimensional size of the data. Specifically, padding in our context refers to the process of symmetrically appending a set of 1D-vectors of size m to *both* ends of the input time series matrix along the temporal dimension. However, we hypothesize that such values appended to the time series data may be potentially problematic, due to the risk of contaminating the original series data with noise or out-of-distribution values. Thus, we need to carefully choose the padding values, for example, using the same adjacent values or the local mean of vectors within a predetermined time window.

Given the hyperparameters defined above, we can deterministically derive the number of slices o as follows:

$$o = \left\lfloor \frac{T + 2p - 2d(\omega - 1) - 1}{s} + 1 \right\rfloor$$

We summarize the Temporal-Slicing Stack Transformation in Algorithm 1. For simplicity, we do not consider padding and dilation in the formulation (*i.e.* $p = 0$ *and* $d = 1$).
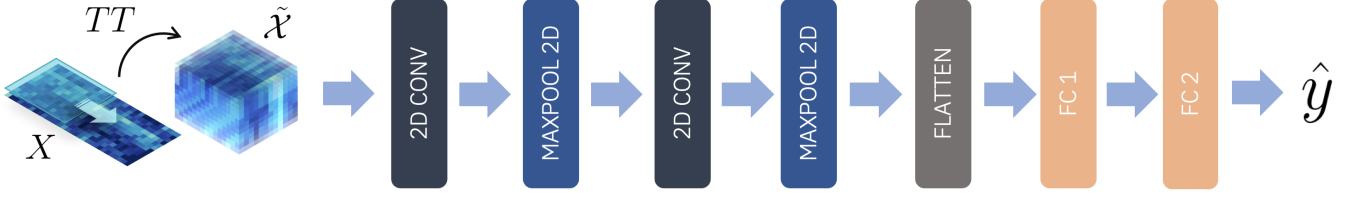
Fig. 3. Temporal Tensor Transformation Network Architecture

Algorithm 1 Temporal-Slicing Stack Transformation

Input: $X \in \mathbb{R}^{m \times T}$: 2D input multivariate time series
Output: $\tilde{\mathcal{X}} \in \mathbb{R}^{m \times \omega \times o}$: 3D output temporal tensor
    Init: $\tilde{\mathcal{X}} \in \mathbb{R}^{m \times \omega \times o}$
 1: **for** $i = 1$ to o **do**
 2:    $\tilde{\mathcal{X}}[:, :, i] \leftarrow X[:, i * s : i * s + \omega]$
 3: **end for**
 4: **return** $\tilde{\mathcal{X}}$

D. Neural Network Architecture

In this section, we describe the deep learning architecture on top of the proposed temporal tensor transformation, which is referred to as *TSSNet*. The neural network architecture is based on a fairly simple network structure, as shown in Figure 3.

1) Temporal-Slicing Stack Transformation: Given the initial input multivariate time series X, we first perform the temporal tensor transformation as described in Algorithm 1. As noted previously, the Temporal-Slicing Stack Transformation is defined as a mapping of $TT : X \rightarrow \tilde{\mathcal{X}}$, where the input data is a two dimensional matrix $X \in \mathbb{R}^{m \times T}$ and the output is a three dimensional tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{m \times \omega \times o}$.

2) Convolutional Neural Network: After transforming the input time series data to $\tilde{\mathcal{X}}$, we utilize a CNN to extract features from the tensor along the $\omega \times o$ plane. That is, we treat each feature as a separate channel. The total number of channels is therefore equal to the number of features m. We also fix one dimension of the CNN kernel to be the stack height o. The dimension of the kernel is thus $o \times k$, where k is the width of the convolving kernel. Suppose we have l CNN kernels, the i-th kernel learns the following set of weights:

$$h_i = W_i * \tilde{\mathcal{X}} + b_i, \quad i \in \{1, 2, ..., l\}$$

where $*$ denotes the convolution operator, W_i and b_i are the weight and bias parameters, respectively. Note that during this process, we do not apply any sort of activation function as the model is sensitive to any type of operation that can collapse the range of values from the input (e.g., negative values will become zero in RELU). We empirically set the number of kernels to be m. The output feature maps from the convolution operation is subsequently fed into a max pooling operation, which is applied over the same $\omega \times o$ plane. We repeat these two operations twice in a successive manner.

3) Dense Layer: After the features are extracted from the CNN layer, the corresponding feature maps are then flattened out as a single vector. It is then subsequently fed into one fully connected hidden layer, fc_1:

$$fc_1 = W_1 \times h + b_1$$

where h is the 1D flatted out feature map from the previous CNN feature maps, and W_1 and b_1 are the weight and bias parameters, respectively. As a heuristic, the dimension of the hidden weight parameters used in this dense layer is usually greater than that of the output layer dimensions. Finally the subsequent latent vector fc_1 is fed into the output layer, where we learn the following set of parameters:

$$\hat{y} = W_2 \times fc_1 + b_2$$

where W_2 and b_2 are the weight and bias parameters. The final output is a 1D vector, $\hat{y} \in \mathbb{R}^{m \times h}$, which can also be reshaped as a 2D matrix of dimensions $m \times h$.

E. Objective Function

To train the proposed neural network architecture, we minimize the squared error loss function:

$$\min_{\Theta} \sum_{t \in T_{train}} \|Y_t - \hat{Y}_t\|_F^2$$

where the cost is minimized w.r.t. the parameters Θ, and $\|\cdot\|_F^2$ denotes the Frobenius norm.

F. Optimization Method

To optimize the cost function, we utilize canonical methods for optimizing standard neural network. Specifically, we can apply common gradient-based methods such as stochastic gradient descent (SGD) or the Adam algorithm [18].

IV. EXPERIMENT

We evaluate the performance of the proposed neural network architecture in this section. We first introduce several baseline models used for the benchmark comparison, and then the evaluation metrics and data sets, and finally present the experimental results.

A. Baseline Models

To evaluate the performance and robustness of our proposed model, we compare it against the following methods:

- Vector Auto Regression (VAR)
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- 1D Convolutional Neural Network (CNN)
- LSTNet [6]

All implementations are developed in PyTorch [19], with the exception of the Vector Autoregression (VAR) model which is based on the Python StatsModels package [20]. Our evaluations are performed across models which are stateful (i.e. VAR and RNN variants), stateless (i.e. CNN variants), and a hybrid of the two (i.e. LSTNet).

For Recurrent Neural Network variants, we implement a many-to-one single layer vanilla architecture for the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) with both having two fully connected layers appended at the end. Likewise, the 1D Convolutional Neural Network (CNN) architecture utilizes a single 1D CNN and a max pooling block with two fully connected layers. In this architecture, we treat the 2D input time series as a single channel image and perform convolution over the temporal-variable plane. The LSTNet architecture is based on the code package from Lai et. al [6]. For all the neural network based models, they generate a batched multistep prediction values instead of a single time step output. For Vector Auto Regression, the predictions are generated in an iterative manner, where the prediction at the current time step is appended to the previous input in order to predict the value at the next time step.

B. Evaluation Metrics

To compare all the methods, we use the following two evaluation metrics. The objective of our models is to minimize the Root Mean Squared Error (RMSE), while jointly maximizing the empirical correlation coefficient (CORR).

Root Mean Square Error (RMSE):

$$RMSE = \frac{1}{n} \sum_{i=1}^{n} \sqrt{\sum_{j=1}^{m} \sum_{t=1}^{h} (\mathcal{Y}_{ijt} - \hat{\mathcal{Y}}_{ijt})^2}$$

where $\mathcal{Y}$, $\hat{\mathcal{Y}} \in \mathbb{R}^{n \times m \times h}$, and n is the total number of samples being computed. We aggregate the error across all m features for a total of h horizons of the predicted values.

Empirical Correlation Coefficient (CORR):

$$CORR = \frac{1}{n} \sum_{i=1}^{n} \frac{\sum_{j=1}^{m} \sum_{t=1}^{h} (\mathcal{Y}_{ijt} - \bar{\mathcal{Y}}_{i:t})(\hat{\mathcal{Y}}_{ijt} - \bar{\hat{\mathcal{Y}}}_{i:t})}{\sqrt{\sum_{j=1}^{m} \sum_{t=1}^{h} (\mathcal{Y}_{ijt} - \bar{\mathcal{Y}}_{i:t})^2 (\hat{\mathcal{Y}}_{ijt} - \bar{\hat{\mathcal{Y}}}_{i:t})^2}}$$

where $\mathcal{Y}$, $\hat{\mathcal{Y}} \in \mathbb{R}^{n \times m \times h}$, $\bar{\mathcal{Y}}$, and $\bar{\hat{\mathcal{Y}}}$ denote the ground truth value, the predicted value, the mean value of the ground truth for each feature, and the mean value of the predicted value for each feature, respectively.

C. Data Sets

We use four benchmark data sets across a wide variety of domain applications to compare the performance of all the models. Each data set presented here has various degrees of non-stationarity. This allows us to evaluate the potential strengths and weaknesses of our proposed model architecture. Due to memory-constraints, for each data set, excluding the Exchange Rate data set, we select at random ten features as a method of dimensionality reduction. We plot the autocorrelational functions for each data set in Figure 4.
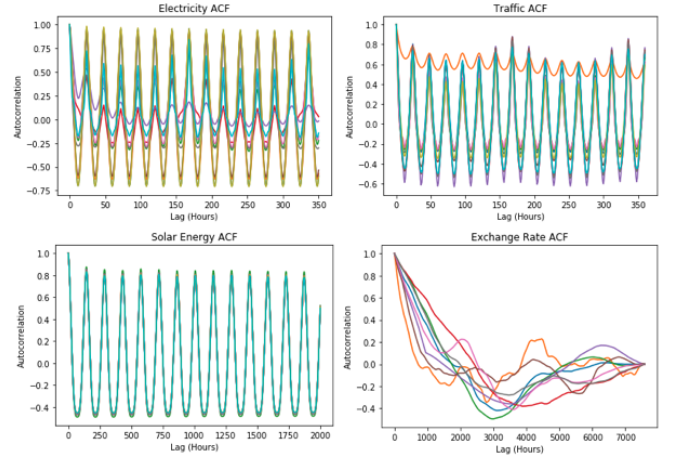

Fig. 4. Autocorrelation function plots of the raw datasets

1) Electricity Usage [21]: The data set contains electricity consumption recordings sampled at 15 min intervals between the year of 2012 to 2014 for 321 clients. It is pre-processed to reflect the hourly consumption rate of the electricity usage instead. From the ACF plot, we can see that different features have different levels of autocorrelation, notably a strong presence of seasonality as most of the features tend to have a consistent lag at around 24 hours.

2) Foreign Exchange Rates: The data set is collected over a period of 26 years between 1990 and 2016, and consists of the daily exchange rates from eight different countries. It contains a total sample size of 7,588. Out of the four data sets, it does not show strong presence of the auto-correlational signal.

3) Solar Power Production [22]: The data set comes from the Solar Power Data for Integration Studies, which contains the solar power and hourly day-ahead forecasts for approximately 6,000 simulated PV plants, at a sampling rate of every 5 minutes, in the year of 2006. From the ACF plot, we can see that the selected features all have uniform seasonality patterns with lag values around 140 hours.

4) San Francisco Traffic [23]: The data set is from the Caltrans Performance Measurement System (PeMS), where traffic data was collected in real-time from over many detectors over the span of one year between 2015 and 2016, in the San Francisco Bay Area freeway. These detectors represent the density of the traffic between 0.0 and 1.0, where 0.0 indicates no traffic and 1.0 represents high amounts of congestion. From

the ACF plot, we can see that most of the selected features have a seasonality with lag values around 160 hours.

For evaluation, we split the data set into three different partitions in a chronological manner, with the first 60% of the data set used for training the model, the next 20% of the data for computing validation scores, and the remaining 20% of the data for testing.

D. Hyperparameters

For training models, we tune the hyperparameters using the *Gaussian Process Optimization Framework (GPyOpt)* [24]. We focus the parameter optimizations on the Temporal-Slicing Stack Transformation. In particular, the window size, ω, is varied between the range of 5 to 10, and the stride, s, is varied between the range of 1 to 5. Furthermore, we also tune the learning rate of our model between values of $[0, 0.01]$ and apply gradient clipping value of 10. Furthermore, we correspondingly adjust the hidden weight dimensions of the model such that the dimension of the first dense layer has a greater number of parameters than the second output dense layer, or the final output dimension of the model. We perform 100 iterations of the modeling process and select the model with the highest empirical correlation coefficient score as our objective value to optimize over.

E. Experimental Results

We present the experimental results on the two defined evaluation metrics in Table I. For each data set, we provide the model with a fixed input of $T = 168$ time steps and obtain predictions over a horizon window of 15, 30, 60, and 120 time steps. We note that these horizons are multi-step window based predictions, as opposed to a single-point estimation. The best performing models with respect to the empirical correlations are in bold for each data set and horizon. We also provide a comparative graph summary of the Empirical Correlation Coefficient in Figure 5.

Based on the empirical results, our proposed model (denoted by TSSNet) outperforms the baseline models on the Electricity, Solar Energy, and Traffic data sets, and outperforms some of the baselines on the Exchange Rates data set. We attribute our analysis based on two key observations: i) the use of stateful versus stateless model types and ii) the autocorrelations present in the data set.

In Figure 5, we observe a noticeable cluster between stateless models (i.e. 1D CNN, LSTNet, and TSSNet) and stateful models (i.e. LSTM, GRU, and VAR) in terms of the overall performance. Furthermore, we observe a sharper performance degradation over larger temporal horizon for stateful models than stateless models. Within stateless models, our proposed model consistently retains an upper bound performance, while the 1D CNN model primarily remains as a lower bound with the LSTNet in between. This trend is also present in the Exchange Rate data set.

The other factor affecting the overall performance of our model is the presence of autocorrelations in the data set, in particular, the seasonality. As previously shown in Figure 4, we

have noted that there is a strong presence of autocorrelationa within the Electricity, Solar Energy, and Traffic data sets, while weaker in the Exchange Rate data set. This realization in context with our empirical results shed light on some observations. For highly autocorrelational data, stateless models tend to perform better over stateful models. However, with weaker autocorrelational signals such as the Exchange Rate data set, our model performs better than the stateless models, but marginally worse than the stateful model variants.

In particular, our experimental results with 1D CNNs and LSTNet demonstrate that the methods in how convolution is applied over the temporal data can lead to difference in performance. Hence, this comparison indicates that our transformation method enables high efficiency of information gain with respect to the autocorrelational features captured over a large temporal region.

We also plot sample predictions against our test set at horizon $h = 120$, with an empirical comparison against LSTNet. The results are shown in Figure 6. We have chosen to evaluate LSTNet for its marginally close performance in terms of the empirical correlation coefficient, but having slightly lower RMSE scores as shown in Table I. From Figure 6, we can observe how effectively our model can predict non-linear patterns as the model is able to adapt to most of the acute signals in the data. From the Electricity data set, we observe much more precise adaptations to the peaks and troughs of the cyclical signals. In the Traffic data set, the proposed TSSNet is able to capture the sinusoidal patterns, however is slightly less precise to sudden peaks such as those found in the last two cycles of the data set. In the Solar Energy data set, we observe a closer fit to the overall shape of the data, as it closely captures both the non-linear and linear patterns within the data. However, for the Exchange Rate dataset, we see that both models have a relatively hard time with the predictions due to high noisy outputs. In TSSNet, we observe that the predictions are relatively stable and centralized around a consistent range of values, while LSTNet's predictions are sporadically shifting the prediction values from time to time.

V. Sensitivity Analysis

In this section, we perform sensitivity analysis to evaluate and understand the advantages and limitations of our proposed architecture. This analysis fixates the intermediate model architecture and manipulates the input and output data parameters in a controlled manner. We observe the resulting effects to the overall performance of the model and draw conclusions on the robustness of our architecture.

In particular, our objectives in this analysis are two folds. First, we empirically study and demonstrate the underlying properties of how our proposed model is able to learn emergent autocorrelational patterns from controlled and synthesized inputs. We subsequently analyze the activation maps to better understand the extracted features. In the second analysis, we evaluate the robustness of the proposed model architecture by varying the input and output dimensions. These two exper-

| Dataset		Electricity				Solar Energy				Traffic				Exchange Rate			
Model	Metric	15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
VAR	RMSE	0.123	0.135	0.148	0.162	0.078	0.129	0.180	0.219	0.082	0.089	0.098	0.108	**0.015**	**0.021**	**0.031**	**0.047**
	CORR	0.701	0.667	0.607	0.507	0.929	0.809	0.648	0.425	0.719	0.679	0.604	0.477	**0.979**	**0.961**	**0.922**	**0.845**
RNN (LSTM)	RMSE	0.126	0.125	0.129	0.131	0.132	0.190	0.215	0.219	0.084	0.087	0.091	0.094	0.032	0.034	0.048	0.057
	CORR	0.725	0.709	0.687	0.672	0.859	0.643	0.526	0.517	0.721	0.694	0.664	0.646	0.958	0.942	0.907	0.837
RNN (GRU)	RMSE	0.126	0.126	0.130	0.131	0.132	0.192	0.214	0.219	0.086	0.089	0.092	0.094	0.021	0.030	0.037	0.059
	CORR	0.720	0.706	0.683	0.668	0.857	0.634	0.522	0.513	0.711	0.687	0.663	0.640	0.971	0.945	0.911	0.837
1D CNN	RMSE	0.057	0.059	0.073	0.083	0.085	0.109	0.123	0.123	0.070	0.071	0.069	0.072	0.039	0.044	0.051	0.069
	CORR	0.891	0.879	0.831	0.796	0.946	0.918	0.899	0.885	0.837	0.831	0.835	0.835	0.937	0.909	0.847	0.729
LSTNet	RMSE	0.064	0.062	0.060	0.060	0.077	0.094	0.110	0.119	0.066	0.067	0.068	0.068	0.056	0.059	0.059	0.074
	CORR	0.890	0.867	0.873	0.868	0.956	0.936	0.906	0.899	0.855	0.847	0.843	0.843	0.815	0.796	0.747	0.699
TSSNet	RMSE	**0.055**	**0.057**	**0.059**	**0.058**	**0.076**	**0.094**	**0.105**	**0.098**	**0.062**	**0.062**	**0.067**	**0.065**	0.029	0.040	0.050	0.065
	CORR	**0.896**	**0.888**	**0.879**	**0.878**	**0.956**	**0.937**	**0.918**	**0.909**	**0.859**	**0.860**	**0.852**	**0.855**	0.944	0.914	0.895	0.780
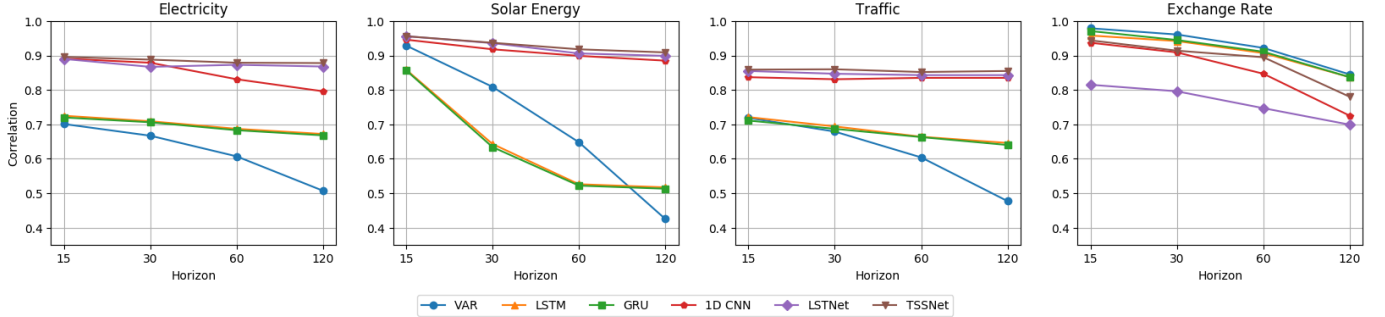
Fig. 5. Empirical correlation coefficient plots by prediction horizon
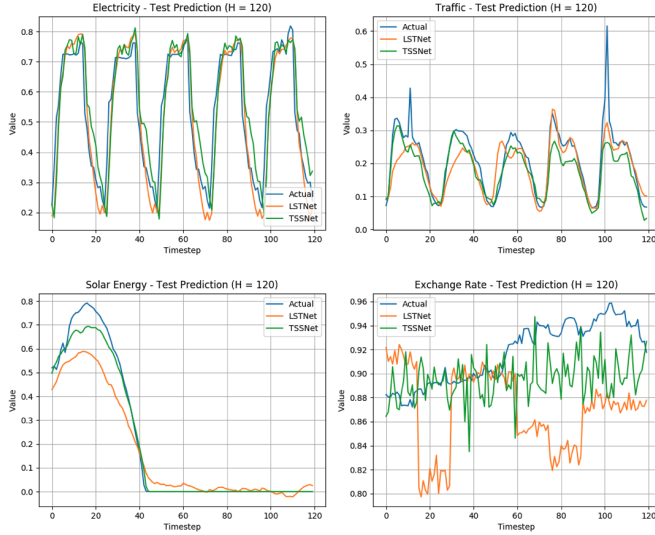

Fig. 6. Comparison of TSSNet and LSTNet predictions

iments will help us better identify some of the core inner workings and draw insights of the model.

A. Synthesized Data Sensitivity Analysis

In this section, we describe methods used for observing the underlying feature maps that are generated by CNN. We perform these experiments to empirically identify properties of autocorrelational signals such as seasonality and trend. For this experiment, we generate a set of synthetic data sets from simple functions which explicitly exhibit properties of seasonality and trend, as demonstrated in Figure 7. Through these controlled and simplified experiments, we can uncover some of the underlying mechanisms for extracting autocorrelational features from the data.

We overfit our models with inputs from known simple deterministic functions, such as the sine and linear functions. We then evaluate the resulting activation maps (after the first convolutional layer) generated by the model, in order to identify which salient features are extracted by the learned CNN filters. Furthermore, to evaluate robustness of our model, we inject noise of various degrees by adding a noise factor ϵ within the sine function argument as $sin(x + \alpha\epsilon)$, where α is the ratio of the noise present, and $\epsilon \in \mathbb{R}$ is a random number sampled from a standard normal distribution. For each function, we empirically evaluate the robustness by increasing the degree of α from 0.00 to 0.75.

Based on the experimental formulation above, we demonstrate the feature maps generated by the CNN layer, as presented in Figure 7. Along the vertical axis, we list the different deterministic functions utilized to generate the synthetic data set, while on the horizontal axis we show the effect over the different rates of noise injection.

Empirically, we found that the CNN filters perform the roles of both a feature extractor as well as a de-noising component of the model. For the first function, $y = sin(x)$, we can observe that the model is able to pick up the uniform repetitive pattern of the data as exhibited by the uniform
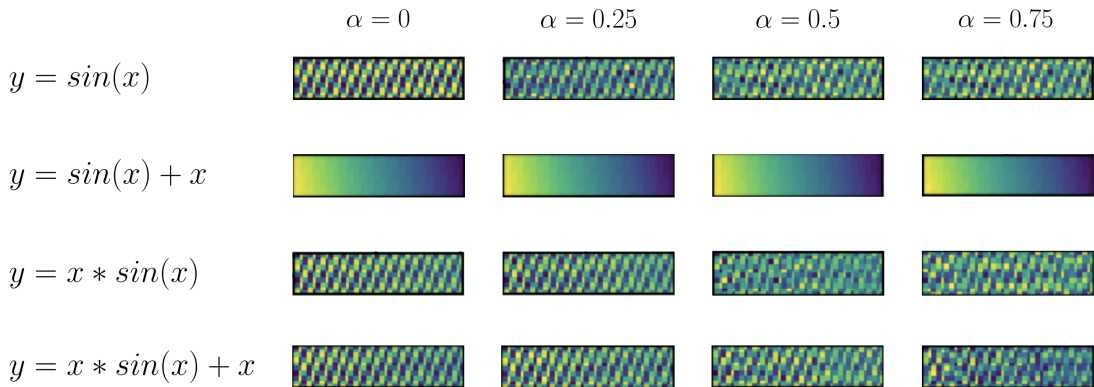
Fig. 7. CNN feature maps of synthesized function datasets

checkerboard like pattern. Even with the injected noise, we see that the filter is able to de-noise a fair amount of the artifacts from the model and still retain the core checkerboard-like pattern as exhibited when $\alpha = 0$. In the second function, $y = sin(x) + x$, we observe across all values of α, the activation map shows a uniform gradient-like pattern. This indicates that the model attenuates the $sin(x)$ component of the function and focuses only on the linear component of the model. In essence, the filter treats $sin(x)$ as a constant and emphasizes the monotonically increasing nature of the time series data from the linear x component of the equation. Hence, the resulting activation map reveals a gradient like pattern which indicates an increasing trend. For the third and fourth functions, we observe the same uniform checkerboard pattern similar to the first function. Furthermore, we observe a slight gradient effect indicating that the values on the left region of the plot is darker and the right region is lighter. This implies that the model's learned filter can extract both the seasonal component as well as the linear trend pattern concurrently from the synthetic data set. The fourth function contains a slightly darker pattern which is indicative of the extra linear factor.

From these controlled experiments, we empirically demonstrate the effectiveness of using CNNs as feature extractors to identify core autocorrelational components of the model, notably, seasonality and trend from the transformed temporal tensor representations. Furthermore, we also demonstrate that such learned filters are able to act as de-noising filters, which can enable generalizations for noisy input time series.

B. Sensitivity Analysis on Input Size v.s. Prediction Horizon

In our previous experiments, we have only evaluated our models against a fixed input size of 168 time steps while we correspondingly vary the output horizon time steps for prediction. In this experiment, we perform a sensitivity analysis by varying both the input and output dimensions. One key motivation behind this experimentation is to assess the balance of the input and output dimensions and how they influence the overall predictive power of the model. Various degrees of temporal granularity, such as daily, weekly, or monthly perspectives of the input data can strongly influence the respective outcome of the prediction due to the observational scope of the information provided to the model. This is particularly important when considering autocorrelational patterns, as long-range dependencies of repetitive patterns can significantly change with respect to the different temporal granularities of the data. As a result, the representation that the model learns will also differ and consequently affect the prediction performance. For this analysis, we constrain our experiment to only evaluate the performance of our proposed model architecture (i.e. TSSNet). We vary the input size by 32, 64, 128, and 256, while also vary the output horizon size by 15, 30, 60, and 120, respectively. We perform experiments with every combination of these input and output dimensions for evaluation.

The experimental results are shown in Table II. We also provide two plots, which evaluate the empirical correlation coefficient performance degradation from the perspective of the input and output dimensions respectively, as shown in Figures 8 and 9. From these plots, we can draw upon several key insights regarding the overall robustness of our model, as well as some heuristics which can potentially help with improving the overall performance of the model.

In Figure 8, we present a plot which demonstrates how the empirical correlation coefficient performance with different input dimensions degrades with respect to the different output horizon dimensions. First, we can observe a similar performance degradation effect shown from the previous experiments. In particular, one key observation to note is that the variance of the correlation with respect to each of the different output horizons are greater when the data is not highly autocorrelated. For example, the Exchange Rate time series does not have strong autocorrelation, therefore exhibiting a wider variance of correlation scores for the input window size with respect to the output horizon dimensions. In contrast, for data sets such as the Electricity and Traffic, the overall correlation performance degradation is relatively smaller and its variance bounds are significantly tighter. However, one detail to make note of in Figure 8 is the top performing models for each of

| Dataset		Electricity				Solar Energy				Traffic				Exchange Rate			
Input Size	Horizon	15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
32	RMSE	0.091	0.098	0.105	0.109	0.091	0.128	0.145	0.144	0.068	0.072	0.077	0.075	0.018	0.027	0.036	0.038
	CORR	0.862	0.835	0.810	0.796	0.941	0.879	0.837	0.840	0.846	0.825	0.800	0.804	0.974	0.949	0.919	0.918
64	RMSE	0.094	0.100	0.104	0.110	0.099	0.106	0.113	0.115	0.067	0.071	0.072	0.071	0.024	0.029	0.041	0.057
	CORR	0.850	0.828	0.811	0.809	0.927	0.917	0.905	0.907	0.848	0.829	0.818	0.823	0.960	0.942	0.899	0.824
128	RMSE	0.091	0.101	0.103	0.101	0.083	0.093	0.101	0.102	0.066	0.068	0.071	0.069	0.029	0.032	0.034	0.063
	CORR	0.861	0.828	0.818	0.829	0.950	0.937	0.926	0.923	0.845	0.836	0.832	0.835	0.954	0.936	0.887	0.802
256	RMSE	0.093	0.097	0.101	0.101	0.075	0.089	0.101	0.106	0.063	0.064	0.066	0.068	0.046	0.054	0.063	0.073
	CORR	0.857	0.854	0.821	0.826	0.958	0.942	0.925	0.919	0.856	0.854	0.844	0.832	0.886	0.858	0.773	0.715
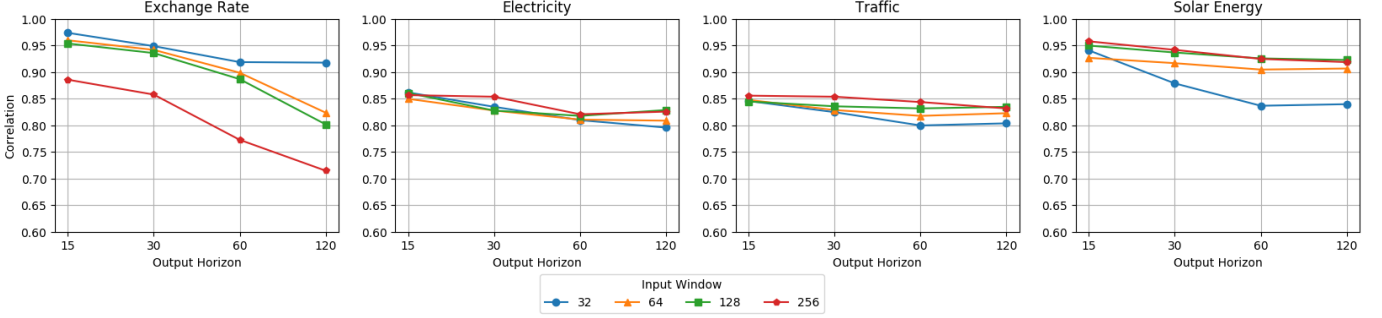
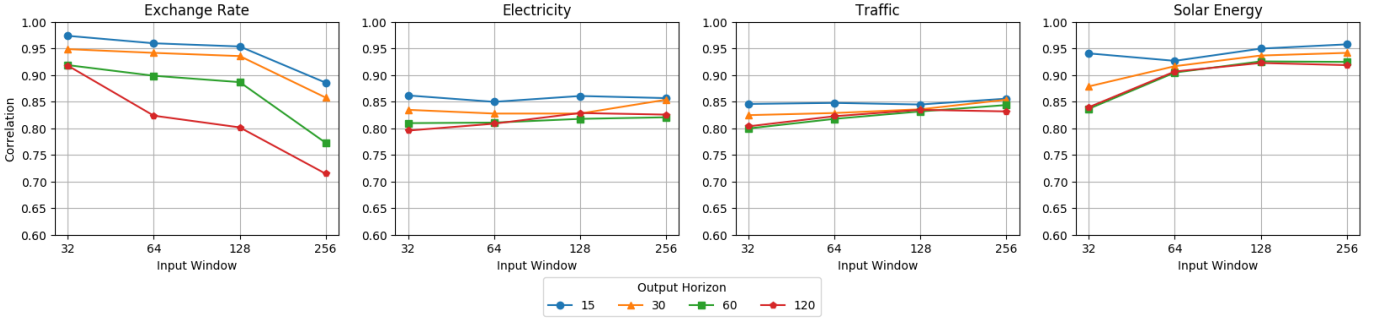Fig. 8. Input size v.s. output horizon correlation degradation

Fig. 9. Output horizon v.s. input size correlation degradation

the different data sets. While in the Exchange Rate data set, the model with input window of 32 performs the best, the other data sets mostly have larger input window sizes performing better in general. This results suggest that the overall input size can significantly affect the performance of the model. This also provides further evidence that our proposed model can better extract features from data which exhibits long-range autocorrelational patterns.

In Figure 9, we present a different view of the results which demonstrates how the empirical correlation coefficient performance with different output dimensions is influenced by varying the input sizes. The motivation behind this analysis is to evaluate the overall robustness of the different output horizon dimensions of the model's prediction with respect to varying the overall input window dimensions. Unlike the previous analysis from the first experiment, we focus on

empirically evaluating the effect of the benefited information gain the model has obtained when provided a greater amount of temporal information (i.e., a larger input window). In Figure 9, we also notice very similar effects of the variance bounds of the correlation results with respect to the different input window sizes, as highly autocorrelated data tend to have smaller bounds while non-autocorrelated data have larger variance ranges. In particular, increasing the input window sizes with respect to the different output horizon dimensions, we note that the overall performance degradation is minimal, and in some cases performance improves for data sets which exhibit high autocorrelational properties (e.g., the Electricity, Traffic, and Solar Energy). In contrast, when the data does not contain any sort of autocorrelational signals (e.g., the Exchange Rate), the model performance drops significantly as we increase the input window sizes. These results empirically

reinforce the notion that our model is able to robustly learn long-range temporal dependencies with high autocorrelational features.

VI. CONCLUSION

We propose a neural network architecture for multivariate time series prediction through a new class of transformation function known as Temporal Tensor Transformations. In particular, we present the Temporal-Slicing Stack Transformation which utilizes a slice-based operator to transform the original 2D time series into a 3D tensor. This transformation encodes long-range auto-correlational features that can be extracted by a Convolutional Neural Network. Both experimental results and sensitivity analysis provide strong evidence that the proposed architecture is able to learn non-linear auto-correlational patterns effectively from the data.

For future work, we plan to investigate various components of the proposed architecture. To further understand the underlying mechanisms and sensitivity of the architecture, we aim to carefully identify which hyperparameters are more sensitive with respect to the overall model performance. Additionally, to better improve the prediction performance of the model on data without strong autocorrelation, we will explore the use of hybrid approaches that combine both our stateless approach in feature extraction in conjunction with recurrent neural networks. As an extension to the proposed temporal tensor transformation, we can also explore other types of transformation methods which utilize explicit components, such as multivariate interactions and variable sampling rates among different variables. The core challenges behind these classes of transformation functions lie in how to design new structures that can enable efficient information gain through the use of CNN for feature extractions.

REFERENCES

[1] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019," *Applied Soft Computing*, vol. 90, p. 106181, 2020.

[2] S. Balluff, J. Bendfeld, and S. Krauter, "Meteorological data forecast using rnn," in *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2020, pp. 905–920.

[3] M. Islam, H. S. Che, M. Hasanuzzaman, and N. Rahim, "Energy demand forecasting," in *Energy for Sustainable Development*. Elsevier, 2020, pp. 105–123.

[4] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[5] I. E. Livieris, E. Pintelas, and P. Pintelas, "A cnn–lstm model for gold price time-series forecasting," *Neural Computing and Applications*, pp. 1–10, 2020.

[6] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *Proceedings of the International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 95–104.

[7] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-Age Information Management*. Springer, 2014, pp. 298–310.

[8] E. Hannan, "A test for singularities in sydney rainfall," *Australian Journal of Physics*, vol. 8, p. 289, 1955.

[9] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[10] H. Lütkepohl, *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[11] A. Milhoj, *Multiple Time Series Modeling Using the SAS VARMAX Procedure*. SAS Institute, 2016.

[12] H. J. Bierens, "Var models with exogenous variables," *Retrieved October*, vol. 10, p. 2015, 2004.

[13] M. Alvarez and N. D. Lawrence, "Sparse convolved gaussian processes for multi-output regression," in *Proceedings of Advances in Neural Information Processing Systems*, 2009, pp. 57–64.

[14] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," *arXiv preprint arXiv:1703.04691*, 2017.

[15] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." *arXiv preprint arXiv:1609.03499*, 2016.

[16] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017, pp. 2627–2633.

[17] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[20] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, 2010.

[21] "Uci machine learning repository: Electricityloaddiagrams20112014 data set," https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014, accessed: 2019-06-18.

[22] "Solar power data for integration studies - grid modernization - nrel," https://www.nrel.gov/grid/solar-power-data.html, accessed: 2019-06-18.

[23] "Caltrans pems," http://pems.dot.ca.gov/, accessed: 2019-06-18.

[24] T. G. authors, "GPyOpt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.