Multivariate Time Series Prediction Based on Optimized Temporal Convolutional Networks with Stacked Auto-encoders

Yunxiao Wang YXIA_WANG@QQ.COM
Zheng Liu ZLIU@NJUPT.EDU.CN
Di Hu 3283765887@QQ.COM
Mian Zhang GUGUMIAN@GMAIL.COM
Jiangsu Key Laboratory of Big Data Security and Intelligent Processing,
School of Computer Science, Nanjing University of Posts and Telecommunications, China

Abstract

Multivariate time series prediction has recently attracted extensive research attention due to its wide applications in the area of financial investment, energy consumption, environmental pollution and so on. Because of the temporal complexity and nonlinearity existing in multivariate time series, few existing models could provide satisfactory prediction results. In this paper, we proposed a novel prediction approach based on optimized temporal convolutional networks with stacked auto-encoders, which can achieve better prediction performance as demonstrated in the experiments. Stacked auto-encoders are employed to extract effective features from complex multivariate time series. A temporal convolutional network is then constructed serving as the prediction model, which has a flexible receptive field and enjoys faster training speed with parallel computing ability than recurrent neural networks. The optimal hyperparameters in these models are discovered by Bayesian optimization. We performed extensive experiments by comparing the proposed algorithms and other popular algorithms on three different datasets, where the proposed approach obtain the best prediction results in various prediction horizons. In addition, we carefully analyze the search process of Bayesian optimization and provide further insights into hyperparametric tuning processes combining the exploration strategy with the exploitation strategy.

Keywords: Multivariate time series prediction, temporal convolutional networks, stacked auto-encoders, Bayesian optimization.

1. Introduction

The problem of time series prediction has been studied for decades and is still among the most challenging problems in many related applications. With the recent advancement of deep learning and GPU-based parallel computing, time series prediction has make fruitful progress in many fields, including finance investment, energy consumption, urban pollution and so on. For example, in smart city computing, accurate estimation of traffic flow, temperature and pollution level can lead to effective suggestion for citizens and help the government to make better decisions. However, due to the temporal complexity and nonlinearity in multivariate time series, the current prediction results of multivariate time series are not satisfactory. How to handle the above issues appropriately is still an open

problem. In this paper, we construct temporal convolutional networks to accomplish multivariate time series prediction. Stacked auto-encoders are utilized to extract valid features, and Bayesian optimization is employed to search the optimal hyperparameters.

Extracting valid features from multivariate time series can help to reduce the complexity of the training process, as well as improve the prediction accuracy. Manually feature extraction by domain experts is labor-intensive and time-consuming. Traditional feature extraction methods such as principal components analysis (PCA) and linear discriminant analysis (LDA) cannot work well on multivariate time series because of the failure of capturing the implicit nonlinearity. Popular feature extraction for multivariate time series can be categorized into two groups: supervised learning methods and unsupervised learning methods. The convolutional neural network (CNN) (Pak et al., 2018) and auto-encoder (AE) (Chen et al., 2018) are the representative methods in each group. In this paper, to minimize the leverage of domain knowledge and human intervention, we contract stacked auto-encoders (SAEs) (Coutinho et al., 2019) to extract valid features from multivariate time series and capture the complex relationship among multiple variables.

Deep leaning has been proven to be a promising approach in many applications including image recognition, speech recognition, pattern recognition, natural language processing, as well as time series prediction. The application of recurrent neural networks (RNN) (Elman, 1991) to sequence modeling problems dates back to 1980s. However, it is difficult for traditional RNNs to obtain good performance because of the gradient vanishing and gradient explosion during model training. Based on RNNs, long-short-term memory network (LSTM) (Hochreiter and Schmidhuber, 1996) achieves accurate prediction and stable gradient during the training process. Gate recurrent unit (GRU) (Chung et al., 2014) further simplifies LSTM without deteriorating the performance. Nevertheless, due to the serial training mode, RNNs have a relatively slow training speed, especially with large network depth. Temporal convolutional network (TCN) (Bai et al., 2018) is proposed to address this issue which can be trained in parallel. TCN achieves superior performance than RNNs in many applications. In this paper, we employ TCN to capture the long-term dependency in multivariate time series for future prediction. It is worth noting that the hyperparameters in the prediction model need to be explored carefully to achieve the best prediction results. We utilize Bayesian optimization to search these optimal hyperparameters efficiently.

In this paper, we propose an approach for multivariate time series prediction based on optimized temporal convolutional network (SAEs-BO-TCN) with stacked auto-encoders. The contributions of this paper are summarized as below.

1. We apply temporal convolutional network (TCN) for multivariate time series prediction to capture the long-term dependency in multivariate time series, where stacked auto-encoders are employed to extract valid features.

2. We employ Gaussian process to model the objection function of TCN and utilize Bayesian optimization with upper-confidence bounds heuristic to search efficiently the hyperparameters in parallel in the proposed approach with various balancing factor values.

3. We perform extensive experiments to demonstrate the effectiveness and efficiency of our proposed multivariate time series prediction approach by comparing the proposed

approach with both traditional and popular time series prediction methods. Experimental results show that the proposed approach achieves the best prediction results.

The rest of this paper is organized as follows. We formalize the multivariate time series prediction problem in Section 2 and explain in detail the proposed prediction approach in Section 3. We report the experimental results in Section 4. We present the related work of multivariate time series prediction in Section 5 and conclude this paper in Section 6.

2. Problem Definition

In multivariate time series prediction applications, one needs to predict ahead up to a ceratin time horizon, which sometimes is called *lead time* or *prediction horizon*. Mathematically, Let $X = (X_1, X_2, ..., X_T) \in \mathbb{R}^{m \times T}$ denote a multivariate time series, where m is the number of variables in the time series, and $X_i = (x_i^1, x_i^2, \cdots, x_i^m)$ is the measurements of the input multivariate time series at time i. The problem of multivariate time series prediction is to predict the potential values of the multivariate time series at certain moments in future, denoted as $Y = (Y_{T+1}, Y_{T+2}, ..., Y_{T+p}) \in \mathbb{R}^{m' \times p}$, where usually $m' \leq m$ and $Y_j = (y_j^1, y_j^2, \cdots, y_j^{m'})$ is the predicted measurements of the multivariate time series at time j. p is the prediction horizon and the above problem is also known as p-step-ahead prediction.

The multi-step-ahead prediction task is often solved by either explicitly training multiple prediction models for different steps, or by iteratively performing one-step-ahead prediction up to the desired horizon. Training multiple prediction models for different steps considers much less context of the predicted values because it only uses the context information p-step ahead, while in the iterative methods, the p-step-ahead prediction problem is done by iteratively performing one-step-ahead prediction, where previously predicted values serve as the context of the later predicted values. Another advantage of the iterative method is that explicitly training multiple prediction models need to know the number models beforehand, that is the value of p, while the iterative method is appropriate for any arbitrary value of p, so long as the prediction performance is acceptable.

By using predicted values instead of real observations, errors might be propagated and accumulated in the prediction model, resulting in poor prediction performance. Hence, we need to carefully select the appropriate prediction strategy, as well as the prediction horizon. In this paper, we adopt the iterative method to perform one-step-ahead prediction up to the desired horizon.

3. The Proposed Model

The overall framework of the proposed model SAEs-BO-TCN is presented in Figure 1. The model is divided into four phases. The first is the feature extraction stage. The original multivariate time series is input to the SAEs to obtain the reduced dimensions of reconstructed time series. The last hidden layer serves as the new feature representation of the original time series. In the second phase, we split the data set into fixed length subsequences using a time window of size T with sliding step S shown in Figure 2.

In the third stage, the split time series is fed to TCN to train the prediction model. Finally, we apply the Bayesian optimization to search for the optimal hyperparameters of

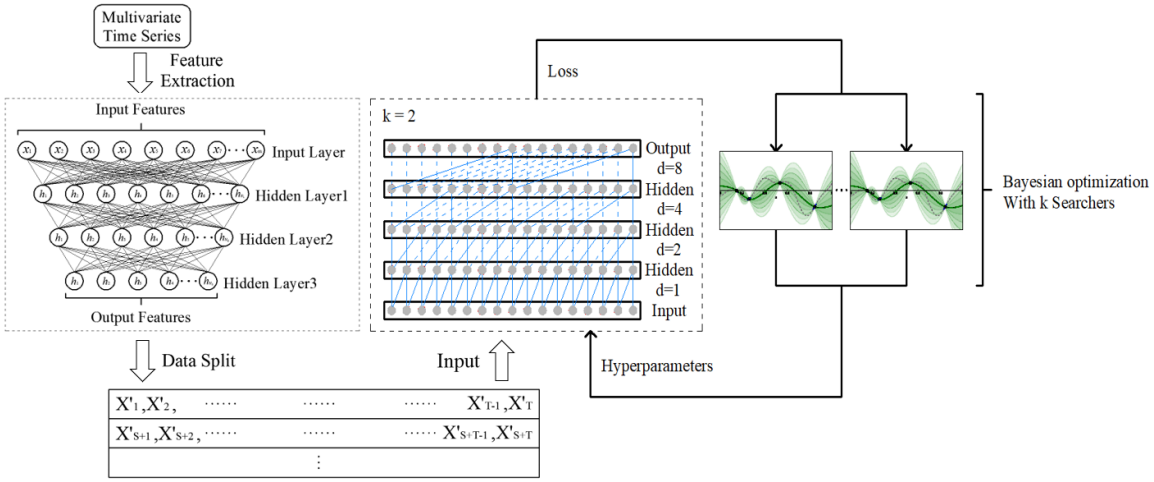


Figure 1: The Overall framework of SAEs-BO-TCN.

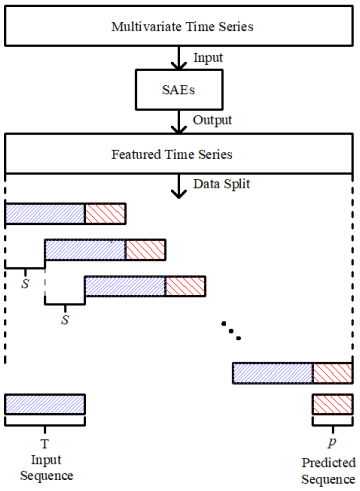


Figure 2: Data split using time sliding window.

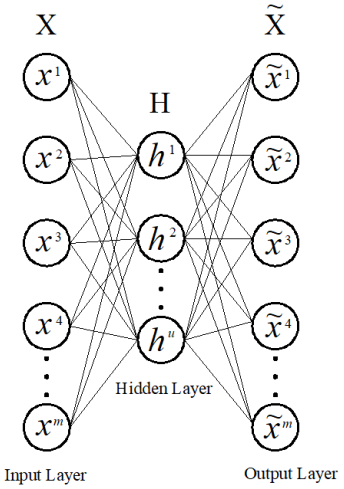


Figure 3: The single-layer Auto-encoder.

TCN and multiple searchers with specific search strategy are set to efficiently search the entire hyperparameter space in parallel. Through the above four stages, we can train a deep neural network to make the p-step-ahead prediction.

3.1. Stacked Autoencoders

The non-linear characteristic of the multivariate time series accompanied by noise data is difficult to predict directly. Extracting features in advance rather than feeding the original time series directly to the model can improve prediction performance and training speed. Autoencoder is an unsupervised feature extractor that learns important features of multivariate time series by constructing deep neural networks and using error backpropagation algorithms.

Multivariate time series is usually unlabelled and interdependent. Unlike using supervised methods such as CNNs which need to collaborate with the predicting model, autoencoders can separate the feature extraction and prediction process. Moreover, autoencoders can reduce the feature space to distill the valid and robust features for the predicting model.

A single-layer AE consists of only three layers of neural networks, the input layer, the hidden layer with multiple units, and the reconstruction layer. As is shown in Figure 3, the first step of AE is to map the input feature vector to the hidden layer, while the second step is to map the hidden layer vector to the output layer to complete the reconstruction of the input features. The above two steps can be summarized as the following formula:

$$a(X) = f(W_1 X + b_1) \tag{1}$$

$$\widetilde{X} = f(W_2 a(X) + b_2) \tag{2}$$

where $X \in \mathbb{R}^m$ and $\widetilde{X} \in \mathbb{R}^m$ are the input vector and reconstructed vector, $a(X)$ is the hidden layer vector mapped by the input layer, u is the encoding dimension of the hidden layer , W_1 and W_2 denote the weight vector of hidden layer and the output layer, b_1 and b_2 denote the bias of the hidden layer and the output layer respectively, and f is a nonlinear activation function such as sigmoid function, tanh or rectified linear unit (ReLu) (Cui et al., 2017).

This network can be trained by minimizing the reconstruction error $L(X, \widetilde{X})$, which measures the error between the input vector and the reconstructed vector. The optimization function can be formulated as:

$$\underset{W_1,W_2,b_1,b_2}{\arg\min}\ L(X, \widetilde{X}) = \underset{W_1,W_2,b_1,b_2}{\arg\min}\ \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} (\widetilde{x}_i^j - x_i^j)^2 \tag{3}$$

where n and m denote the length of the multivariate time series and the number of variables respectively. This unsupervised learning method can separate feature extraction and time series prediction, which effectively improves the training speed and prediction performance of the model. Since the single-layer AE is limited by the number of network layers, it is difficult to learn the complex features of multivariate time series. We build Stacked Autoencoders to enhance the feature extractor's learning ability and there is a five-layer SAEs in our model. We choose ReLu as the activation function and use the error back-propagation algorithm to train SAEs.

There are other types of autoencoders such as denoising autoencoders(DAEs) (Cohen et al., 2008), contractive autoencoders(CAEs) (Rifai et al., 2011). Denoising autoencoders can learn a more robust representation of the input series by stochastically masking entries of the input, which can alleviate the noise corruptions of time series. Contractive autoencoders share with DAEs the similar motivation, which is reducing the influence of small variations of the input. CAEs favor mappings that are more strongly contracting at the training samples, and they balance the reconstruction error with a penalty term. In this paper, we focus on SAEs due to its simplicity. It is worth noting that our proposed framework can incorporate with DAEs or CAEs directly by substituting either one for SAEs. Similar to the usage of SAEs in this paper, we could obtain better latent feature representations of multivariate time series by stacking multiple layers of DAEs and CAEs. We leave the comparison of the performance of various autoencoders in the context of time series prediction for future work.

3.2. Temporal Convolutional Networks

TCN uses a convolutional architecture to deal with sequence modeling problems. Benefiting from parallelization of convolution operations, TCN solves the time-consuming problem of RNNs during training and predicting. Beyond this, TCN network can flexibly acquire historical information by combining deep residual networks and dilated convolution. TCN consists of two design concepts: (1) using causal convolution operation to avoid information leakage from future to past, (2) mapping a sequence of arbitrary length to a fixed length sequence using a 1D fully-convolutional network (1D FCN) (Long et al., 2015). As is shown in Figure 4, TCN includes the following four main parts:

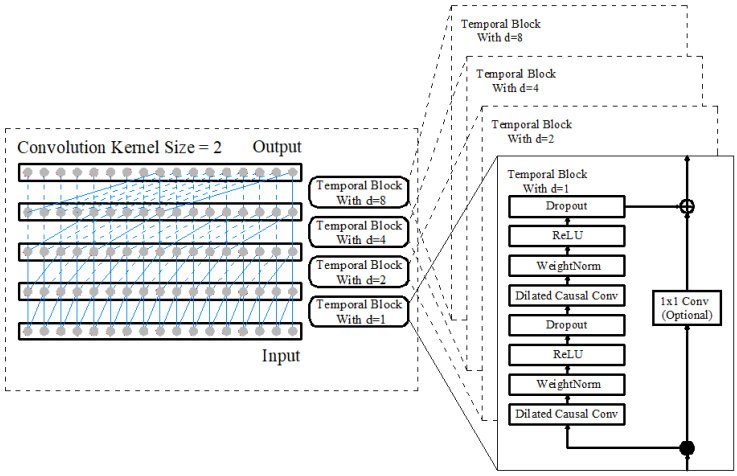


Figure 4: The Temporal Convolutional Network

Causal Convolution. Causal convolution splits the convolution operation in half so that it can only convolute the information of past time step. The prediction result of current time t is only related to historical information, thus avoiding information leakage, which is similar to time delay neural networks.

1D FCN. The 1D FCN generates an output sequence in the same length as the input sequence by using a zero padding strategy. The significance of introducing 1D FCN is the ability to make intensive predictions by making full use of the whole time series. The receptive field of high-level convolution widens with the network deepens, which helps to feel the entire input sequence's information for building long-term memory dependencies. Besides, using 1D FCN instead of the fully connected network can significantly reduce the connective parameters between layers, thus speeding up the convergence of the model.

Dilated Convolution. Dilated convolution overcomes the narrow receptive field problem of causal convolution. The size of the convolution kernel remain unchanged with the increase of the number of layers, while the interval between convolution units increases exponentially to obtain more long-term information.

Formally, the 1D dilated convolutional operation on the element of a sequence can be defined as follows:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{4}$$

where $f : \{0, ..., k-1\} \to \mathbb{R}$ is the convolution kernel, k is the kernel size, d is the dilation factor and $s - d \cdot i$ represents the data of the past. d increases as the network gets deeper, calculated by $d_l = 2^l$ at level i of the network. In Figure 4, the convolution kernel size is set to 2. Convolutions are applied on two timestamps, t and $t - d$.

Residual Temporal Block. TCN's receptive field depends on the network depth, kernel size, and the dilation factor. Residual networks (He et al., 2016) are employed to increase the depth of the network shown in Figure 4. A residual block consists of two branches. One branch transforms the input x through a series of neural network layers $\mathcal{F}$, including the dilated causal convolution with the weight normalizing layer, the rectified linear unit (ReLU), and the spatial dropout layer for regularization. The other branch is a direct projection of the input x. Thus, the output o of a residual block is:

$$o = \sigma(x \oplus \mathcal{F}(x)). \tag{5}$$

where σ denotes the activation function. The residual layers are able to learn the modifications to the identity mapping rather than the entire transformation. It is possible that the length of x and the length of $\mathcal{F}(x)$ are unequal in TCN. An additional 1×1 convolution network is employed to handle the different lengths.

Compared with RNNs, TCN has the following advantages. Firstly, it has flexible receptive fields that can be changed by kernel size, dilation factor, and the network depth. Secondly, TCN has a stable gradient during training period compared to RNNs that often have gradient vanishing and gradient explosion problems. Finally, TCN has high parallelism since convolution operations can be conducted without waiting for pre-convolution to be computed.

3.3. Bayesian Optimization For Hyperparameters

The quality of the network hyperparameters determines the performance of the model to a great extent. Aiming to find an automated and effective hyperparameter optimization method, two problems need to be considered thoroughly: (1) How to search high-dimensional hyperparametric space efficiently? (2) How to manage a series of large-scale experiments for hyperparameter optimization?

For a hyperparameter optimization problem, we define an expensive black box function f, which has no specific form and can only get the output y by inputting the sample point x. In this paper, f is the loss function of TCN and can only be described by observation point y. Mathematically, the hyperparameter optimization problem is transformed into minimizing the unknown objective function f:

$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta) \tag{6}$$

where Θ is high-dimensional hyperparameter space to be searched and θ^* is the optimal hyperparameters.

Bayesian optimization mainly consists of two steps. The first is to estimate and update the Gaussian Process through a new observation point D_{t+1}. The next is to guide the sampling of the hyperparameters by maximizing an acquisition function.

Gaussian Process. Suppose that the model f that needs to be estimated obeys the Gaussian Process:

$$f(\theta) \sim GP(\mu(\theta), K(\theta, \theta')) \tag{7}$$

where $\mu(\theta)$ is the mean function of $f(\theta)$ and $K(\theta, \theta')$ is the covariance(also called kernel function) of $f(\theta)$:

$$K = \begin{bmatrix} k(\theta_1, \theta_1) & \cdots & k(\theta_1, \theta_t) \\ \vdots & \ddots & \vdots \\ k(\theta_t, \theta_1) & \cdots & k(\theta_t, \theta_t) \end{bmatrix} \tag{8}$$

Since GP does not depend on observation data, $\mu(\theta)$ is set to 0 to simplify GP, thus we get $f(\theta) \sim GP(0, K)$. For a new hyperparameter search point θ_{t+1}, the covariance matrix of the Gaussian process is updated to:

$$K' = \begin{bmatrix} K & k^T \\ k & k(\theta_{t+1}, \theta_{t+1}) \end{bmatrix} \tag{9}$$

where $k = [k(\theta_{t+1}, \theta_1), k(\theta_{t+1}, \theta_2), \cdots, k(\theta_{t+1}, \theta_t)]$. With the updated covariance matrix, the posterior probability distribution of f can be obtained as follows:

$$P(f_{t+1}|D_{t+1}, \theta_{t+1}) \sim N(\mu_{t+1}(\theta), \sigma^2_{t+1}(\theta)) \tag{10}$$

where $\mu_{t+1}(\theta)$ and $\sigma^2_{t+1}(\theta)$ are the mean and covariance of $f(\theta)$ and can be formulated as follows: $\mu_{t+1}(\theta) = k^T K^{-1} f_t, \sigma^2_{t+1}(\theta) = k(\theta_{t+1}, \theta_{t+1}) - k^T K^{-1} k$. After selecting the appropriate kernel function such as polynomial kernels, spline kernels, Mercer kernel functions and so on (Chu and Ghahramani, 2004), we can update the posterior probability distribution of the hyperparameter to be searched by the above steps.

Hyperparameter sampling. One problem with this method of minimizing the objective function is that a comprehensive evaluation of the entire hyperparameter space is often computationally complicated, because the calculation of the objective function is expensive. Many heuristic algorithms known as acquisition functions have been introduced to solve this problem, such as Thompson sampling, expected improvement (EI), upper-confidence bounds (UCB), and entropy search (ES) (Agrawal and Goyal, 2013).

Generally, there are two sampling strategies, exploration and exploitation. Exploration strategy prefers to explore new hyperparameter space, which tends to search for global optimum, while exploitation strategy is more conservative, preferring to sample near the current optimal parameters, and search for local optimum. From a statistical point of view, exploration wants to select data with greater variance, and exploitation wants to select data closer to the mean. We choose the simple and effective UCB to sample the next hyperparameter by maximizing the UCB acquisition function:

$$\theta_{t+1} = \arg\max_{\theta \in \Theta} S(\theta|D_t) = \arg\max_{\theta \in \Theta} \mu_t(\theta) + \beta^{1/2}_{t+1} \sigma_t(\theta) \tag{11}$$

where β_{t+1} are appropriate constants which trade off the exploration and exploitation strategy. It is important to determine the appropriate β_{t+1} to ensure that the next sampling point of the hyperparameter will not get stuck in the local optimum, nor make a large change.

Let $\epsilon \in (0, 1]$ and $\beta_{t+1} = 2\log(|D|(t+1)^2\pi^2/6\epsilon)$ where $|D|$ is the dimension of data and ϵ is the balance factor between exploration and exploitation strategy. By running GP to construct posterior unbiased estimate, we can obtain a near-zero regret bound which can be attenuated to a constant with the increase of the number of data and evaluation times(Srinivas et al., 2010). In this manner, the cumulative regret is bounded while obtaining the maximum information gain. Different ϵ is set in the experiments to control the increasing rate of β_{t+1} to balance the two search strategies.

Considering $f(\theta)$ sometimes cannot be described explicitly by GP, an ensemble searching strategy is proposed to speed up the convergence of Bayesian optimization and enlarge the searching space of hyperparameters. In detail, we set k searchers with different sampling strategies under specific balance factor ϵ to search the entire hyperparameter space in parallel. Above all, by sampling new hyperparameters and running GP repeatedly until reaching the specified number of iterations, we can get the optimal results efficiently.

The TCN hyperparameter search algorithm based on Bayesian optimization with multiple searchers is shown as Algorithm 1.

Algorithm 1 SAEs-BO-TCN

Input:

f the loss function of TCN, Θ hyperparametric space to be searched, S UCB function, D dataset, k the number of searchers .

Output:

return the optimal hyperparameter θ^*

for i = 1,2,$\cdots$,k **do**
  $D_1^{(i)} \leftarrow$ InitializeSample(f, D, i)
  for t = 1,2,$\cdots$ **do**
    fit GP and compute the posterior probability distribution $P(f_{t+1}^{(i)}|D_t^{(i)}, \theta_t^{(i)})$.
    select new $\theta_{t+1}^{(i)}$ by optimizing acquisition function: $\theta_{t+1}^{(i)} = \arg\max_{\theta\in\Theta} S(\theta|D_t^{(i)})$
    train TCN model and assign the loss of the valid set to $f_{t+1}^{(i)}$, and update current best value $f_{min}^{(i)}$.
    update observations with $D_{t+1}^{(i)} = D_t^{(i)} \cup (\theta_{t+1}^{(i)}, f_{t+1}^{(i)})$
  end for
end for
$\theta^* = \arg\min_{\theta\in\Theta, i\in[1,k]} f_{min}^{(i)}(\theta)$
return θ^*

4. Experimental Evaluation

In this paper, we compare the performance of six prediction models on three datasets, i.e. ARIMA (Box and Pierce, 1970), GRU (Chung et al., 2014), LSTM (Hochreiter and Schmidhuber, 1996), SAEs-LSTM, BO-TCN and SAEs-BO-TCN. ARIMA, LSTM, GRU are popular time series prediction methods. SAEs-LSTM is the method to apply stacked Auto-encoders in Section 3.1 with LSTM as the prediction model, and SAEs-BO-TCN is the proposed model in this paper.

We report our prediction results with various prediction horizons. In addition, we analyze the hyperparametric search process based on Bayesian optimization with multiple searchers and provide further insights into hyperparametric tuning combining the exploration strategy and the exploitation strategy.

4.1. Datasets

We use three different datasets with different scales as described as follows.

- **Purchase and redemption of capital flow(P&R)**: P&R includes the total daily amount of purchase and redemption on the cash fund investment platform. The correlated variables include daily active user number (DAUN), daily yield (DY), 7 days yield (7DY) and day of week (DOW). The size of P&R is 427, which is a very small data set.
- **Beijing PM2.5 dataset**: The dataset contains the records of hourly PM2.5 data and weather conditions including temperature (TEMP), pressure (PRES), combined wind direction (CBWD), cumulated wind speed (LWS), cumulated hours of snow (LS), and cumulated hours of rain (LR). The size of the Beijing PM2.5 dataset is 43824.
- **SML2010 dataset**: The data set has a high-dimensional multivariate time series, containing 21 internal and external measurements in a domotic house. We select three different variables to predict, which are Temperature Comedor Sensor (TCS), CO2 Comedor Sensor (CO2) and Humedad Comedor Sensor (HCS).

4.2. Experiment Settings

For all datasets, we normalize the continuous variables using z-score and one hot encoding for dictionary data. Each dataset is split into three parts, the training set(60%), the validation set(20%) and the test set(20%) according to the chronological order. Two conventional metrics are selected to evaluate the prediction performance according to the characteristics of each dataset as follows.

- **Mean Absolute Percentage Error(MAPE)** for P&R dataset.

$$MAPE = \frac{100\%}{n}\sum_{t=1}^{n}|\frac{y_t - h_t}{y_t}| \tag{12}$$

- **Mean Square Error(MSE)** for Beijing PM2.5 dataset and SML2010 dataset.

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - h_t)^2 \tag{13}$$

where y_t and h_t denote the actual value and prediction value at time t during evaluation stage and n represents the length of time series evaluated.

To demonstrate the advantages of our proposed model, we compare SAEs-BO-TCN with traditional time series prediction methods ARIMA and RNNs, including LSTM, GRU

Table 1: Hyperparametric search settings

Hyperparameters	**Type**	**Min**	**Max**
No. hidden layers	integer	1	12
No. hidden units	integer	10	256
Learning rate	log-uniform	1×10^{-7}	1×10^{-2}
Dropout rate	uniform	0.1	0.5
Optimizer	categorical	{Adam, RMSProp, Adagrad, Sgd}	
L2 regularization	log-uniform	1×10^{-8}	1×10^{-1}
Gradient clipping norm	uniform	0	20
Gradient clipping value	log-uniform	1×10^{-2}	1
Sequence size	integer	3	256
Kernel size(TCN only)	interger	2	12

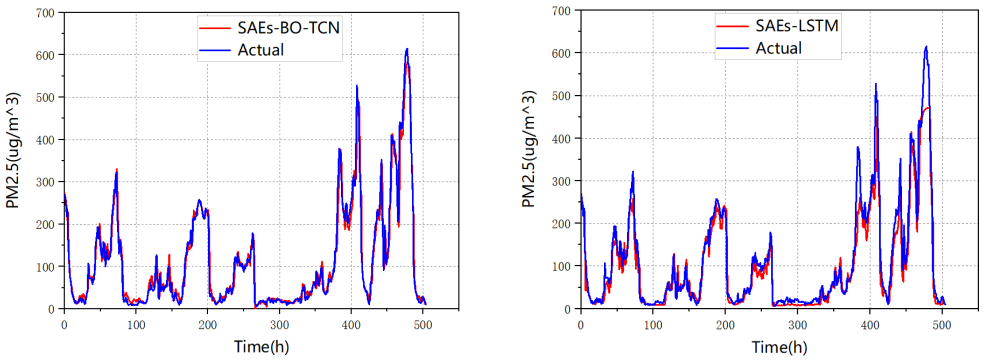
Table 2: The prediction results on three datasets

Datasets	P&R(MAPE%)			Beijing PM2.5(MSE, $\times10^2$)			SML2010(MSE)		
Prediction Horizons	7-days	14-days	21-days	7-days	14-days	21-days	2-days	4-days	6-days
ARIMA	11.85	12.68	13.62	23.65	33.20	69.92	1.41	1.83	2.53
GRU	13.45	14.13	15.63	16.36	28.75	52.49	1.25	1.59	2.14
LSTM	12.63	13.75	14.63	17.33	27.70	49.51	1.21	1.56	2.16
SAEs-LSTM	11.63	12.63	13.62	14.02	23.64	41.77	0.90	1.28	1.65
BO-TCN	11.74	12.66	13.21	13.89	22.72	38.41	0.77	1.13	1.54
SAEs-BO-TCN	**9.65**	**9.87**	**10.65**	**11.93**	**20.53**	**30.07**	**0.62**	**0.88**	**1.21**

and SAEs-LSTM. The SAEs is set to 5 layers, and the number of units in each layer varies in different data sets. For example, in SML2010 dataset, the depth of SAEs is set to 5, and the hidden units are set to 16,14,12,10 respectively. We choose ReLu as the activation function and use MSE as the loss function to train SAEs to extract the features of multivariate time series. For the deep neural network models, we use Bayesian optimization to search over tunable hyperparameters, where the UCB function is used as an acquisition function. Table 1 lists the hyperparameters, as well as their data type and range values.

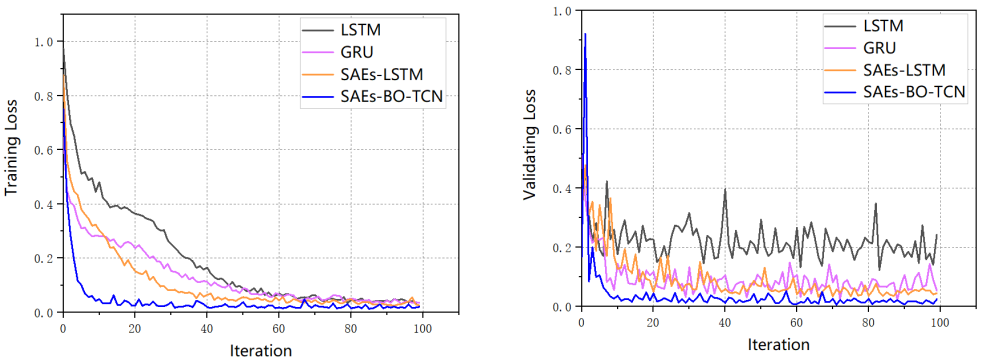
4.3. Experiment Results and Analysis

The evaluation results of all comparative models on three different datasets are shown in Table 2. In each dataset, with the increase of the prediction horizon, it becomes harder to make accurate prediction because of the accumulation of uncertainty. We can see that SAEs-BO-TCN has achieved the best performance in all three datasets. Traditional model ARIMA is hard to achieve good performance in complex multivariate time series prediction problems. LSTM has similar performance with GRU, but better than ARIMA in most cases. The prediction performances of SAEs-LSTM and SAEs-BO-TCN are both superior to LSTM and BO-TCN, where LSTM and BO-TCN use the original multivariate time series as the input. This confirms that the feature representations learned by stacked autoencoders are more effective than the original time series. In addition, in the small data set P&R, the performance of each model is very close, even the best model SAEs-BO-TCN is only 2.5% higher than ARIMA on average, while in the big dataset Beijing PM2.5 and high-dimensional dataset SML2010, deep neural networks can learn more non-linear knowledge and achieve better performance. For visualizing, we select the Beijing PM2.5 dataset for



(a) (b)

Figure 5: The prediction result(red) by SAEs-BO-TCN (a) and SAEs-LSTM (b) vs. the actual data(blue) on Beijing PM2.5 dataset with prediction horizon = 21 days.



(a) (b)

Figure 6: Training loss (a) and validating loss (b) change with the iteration of 4 different models on the Beijing PM2.5 dataset with prediction horizon = 21 days.

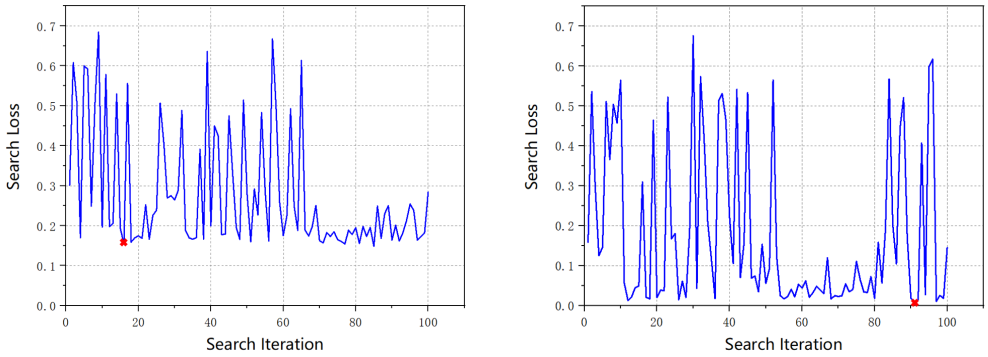
analysis. Figure 5 show the prediction results of the SAEs-BO-TCN(a) and SAEs-LSTM(b) models with 21-days prediction length.

The training loss and testing loss changing with iterations are shown in Figure 6. The results show that the SAEs-BO-TCN network has a faster convergence speed than RNNs, which converges at around the 15th iteration, followed by SAEs-LSTM converging at around the 40th iteration, and finally LSTM and GRU, which converge at around the 70th iteration. This confirms that TCN has faster training and convergence speed due to the advantage of parallel training. Generally, SAEs-BO-TCN is superior to other comparative models with smaller test loss.

In the three datasets, we use Bayesian optimization with multiple searchers to find optimal hyperparameters of SAEs-BO-TCN. Table 3 shows the hyperparametric searching result under different datasets. For demonstrating the optimizing performance of multiple searchers, we compare the performance of multiple searchers with the single searcher. The multiple searchers consist of 5 sub-searchers with different balance factor ϵ:{0.2,0.4,0.6,0.8,1}.

Table 3: Hyperparameter searching result of SAEs-BO-TCN on three datasets

Dataset / Hyperparameters	**P&R**	**Beijing PM2.5**	**SML2010**
No. hidden layers	8	10	5
No. hidden units	40	60	50
Learning rate	4.3×10^{-3}	6×10^{-3}	2.6×10^{-3}
Dropout rate	0.11	0.32	0.21
Optimizer	Adam	Adam	Adam
Sequence size	84	229	209
Kernel size	5	10	7



(a) (b)

Figure 7: Hyperparametric search loss changes of SAEs-BO-TCN on P & R dataset (The smallest loss is achieved in the 16th iteration.) and Beijing PM2.5 dataset (The smallest loss is achieved in the 91th iteration.) with prediction horizon = 21 days.

As is shown in Figure 8 and Figure 9, multiple searchers tend to have faster convergence speed and can find better global minimum than the single searcher. In addition, we analyse the trend of the Bayesian optimization's loss. Figure 7 shows the trend of search loss in the P & R and SML2010 datasets, where the global best value has been marked with a red cross. With the integration of multiple searchers, we can jump out of the local minimum and achieve global optimality.

5. Related Work

5.1. Traditional Methods of Multivariate Time Series Prediction

Traditional methods based on stochastic process theory and statistics have been applied to time series prediction for decades. Autoregressive Integrated Moving Average (ARIMA) (Box and Pierce, 1970) is the most widely used traditional time series prediction model whose main advantage is to use the periodic characteristics of the time series to provide reliable prediction results. However, the traditional ARIMA cannot obtain precise results under random fluctuations and uncertainties, especially for complex non-linear multivariate time series. Recently, Facebook proposed a modular regression model fbprophet (Taylor and Letham, 2017) with interpretable parameters that can be intuitively adjusted by ana-

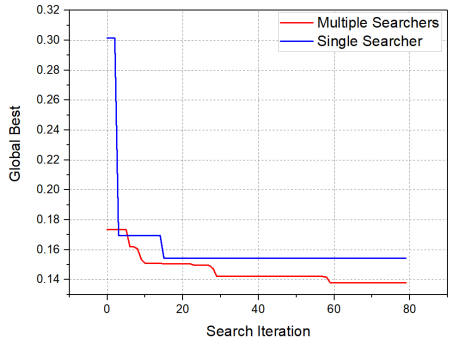


Figure 8: Multiple searchers vs. single searcher in P&R dataset.

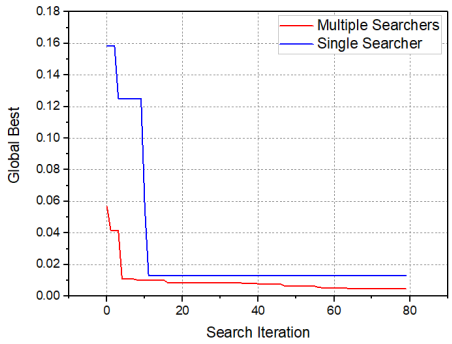


Figure 9: Multiple searchers vs. single searcher in the Beijing PM2.5 dataset

lysts with domain knowledge. This model is suitable for time series containing significant periodic characteristics, but can not accurately predict time series with high randomness and volatility.

Support vector regression (SVR) (Smola and Schölkopf, 2004) is another frequently used time series prediction method, by mapping the inputs into a higher dimensional feature space where a linear regressor is constructed by minimizing an appropriate cost function. Therefore, the nonlinear time series has linear characteristics in the high dimensional feature space, thus predictable. However, the SVR method requires precise selection of hyperparameters such as selection basis functions including radial kernel functions, linear kernel functions, etc. How to determine the kernel function has no suitable method. In addition, solving the convex optimization problem in the high dimensional feature space requires a large amount of storage space. Besides, there are also non-parametric methods for time series prediction such as Gaussian Process (GP) (Chu and Ghahramani, 2004), which is the representative non-parametric model to predict continuous functions.

5.2. Deep Learning Based Methods of Multivariate Time Series Prediction

The traditional time series prediction models are not suitable for non-linear multivariate time series. Therefore, more attention is paid to the deep learning based methods. The deep learning based methods employ deep neural networks to simulate the nonlinear structure to obtain better prediction results for multivariate time series. Deep learning related methods commonly rely on recurrent architectures as the default starting point for sequence modeling tasks. However, the traditional RNNs are often disturbed by gradient vanishing and gradient explosion in the training process (Pascanu et al., 2012). Recently, long-short-term memory (LSTM) has proven to be a promising method for sequence modeling problems because it can solve the gradient issues of RNNs and has a selective memory function (Hochreiter and Schmidhuber, 1996). Gate recurrent unit (GRU) (Chung et al., 2014) is a simplified version of LSTM and has comparable performance with LSTM. However, the serial calculation of RNNs slows down the training and prediction process with long sequences. Recent research shows that a CNN-based architecture, called temporal convolutional networks (TCN) (Bai et al., 2018), can overcome the shortcomings of RNNs and obtain better prediction results.

Compared with traditional time series prediction methods, deep learning based methods can extract implicit features from multivariate time series, and combine long-term and short-term dependencies with improving prediction accuracy.

6. Conclusion

In this paper, we focus on the multivariate time series prediction problem and propose a novel prediction model called SAEs-BO-TCN. We construct stacked Auto-encoders (SAEs) to extract the valid features from multivariate time series to improve the prediction performance. Then, we employ a temporal convolutional network to serve as the predictor. The hyperparameters are optimized based on Bayesian optimization to speed up the search process of the optimal hyperparameters. The extensive experiments on three different datasets show that our proposed prediction model outperforms other popular models. Furthermore, we analyze the searching process of hyperparameters to provide useful insights into the hyperparametric tuning process.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China under Grant 2018YFB1003702, Jiangsu Provincial Natural Science Foundation of China under Grant BK20171447, and Nanjing University of Posts and Telecommunications under Grant NY215045 and NY219084.

References

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. volume 28 of *JMLR Workshop and Conference Proceedings*, pages 127–135. JMLR.org, 2013.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

G. E. P. Box and David A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970.

Kunjin Chen, Jun Hu, and Jinliang He. A framework for automatically extracting overvoltage features based on sparse autoencoder. *IEEE Trans. Smart Grid*, 9(2):594–604, 2018.

Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6(3):1019–1041, 2004.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors. *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, 2008. ACM. ISBN 978-1-60558-205-4.

Maria G. F. Coutinho, Matheus F. Torquato, and Marcelo A. C. Fernandes. Deep neural network hardware implementation based on stacked sparse autoencoder. *IEEE Access*, 7:40674–40694, 2019.

Jia-Le Cui, Shuang Qiu, Mingyang Jiang, Zhi-Li Pei, and Yi-Nan Lu. Text classification based on relu activation function of SAE algorithm. volume 10261 of *Lecture Notes in Computer Science*, pages 44–50. Springer, 2017.

Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. LSTM can solve hard long time lag problems. In Michael Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pages 473–479. MIT Press, 1996.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Unjin Pak, Chungsong Kim, Unsok Ryu, Kyongjin Sok, and Sungnam Pak. A hybrid model based on convolutional neural networks and long short-term memory for ozone concentration prediction. *Air Quality, Atmosphere & Health*, 11(8), Oct 2018.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 833–840, 2011.

Alexander J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias W. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. pages 1015–1022, 2010.

Sean J. Taylor and Benjamin Letham. Forecasting at scale. *PeerJ PrePrints*, 5:e3190, 2017.