Nonpooling Convolutional Neural Network Forecasting for Seasonal Time Series With Trends

Shuai Liu, Hong Ji, and Morgan C. Wang

***Abstract*— This article focuses on a problem important to automatic machine learning: the automatic processing of a nonpreprocessed time series. The convolutional neural network (CNN) is one of the most popular neural network (NN) algorithms for pattern recognition. Seasonal time series with trends are the most common data sets used in forecasting. Both the convolutional layer and the pooling layer of a CNN can be used to extract important features and patterns that reflect the seasonality, trends, and time lag correlation coefficients in the data. The ability to identify such features and patterns makes CNN a good candidate algorithm for analyzing seasonal time-series data with trends. This article reports our experimental findings using a fully connected NN (FNN), a nonpooling CNN (NPCNN), and a CNN to study both simulated and real time-series data with seasonality and trends. We found that convolutional layers tend to improve the performance, while pooling layers tend to introduce too many negative effects. Therefore, we recommend using an NPCNN when processing seasonal time-series data with trends. Moreover, we suggest using the Adam optimizer and selecting either a rectified linear unit (ReLU) function or a linear activation function. Using an NN to analyze seasonal time series with trends has become popular in the NN community. This article provides an approach for building a network that fits time-series data with seasonality and trends automatically.**

***Index Terms*— Automatic learning, convolutional layer, neural network (NN), pooling layer, time series.**

I. Introduction

TIME-SERIES data are an important class of temporal data. They are organized by timeline and record a series of observations (e.g., the price of stocks or visit counts of a website). Studies on time-series data have a long history. One of the most important achievements, the autoregressive integrated moving average (ARIMA) model, can be traced back to the 1970s [1].

Seasonality, trends, and time lag correlations are the most important components of time-series data. Traditional modeling methods, such as ARIMA, attempt to decompose the time series to remove seasonality and trend effects and then model the autocorrelation using linear or nonlinear models. With the progress of machine learning and deep learning, it has become increasingly popular in modern studies to analyze time-series data using neural networks (NNs).

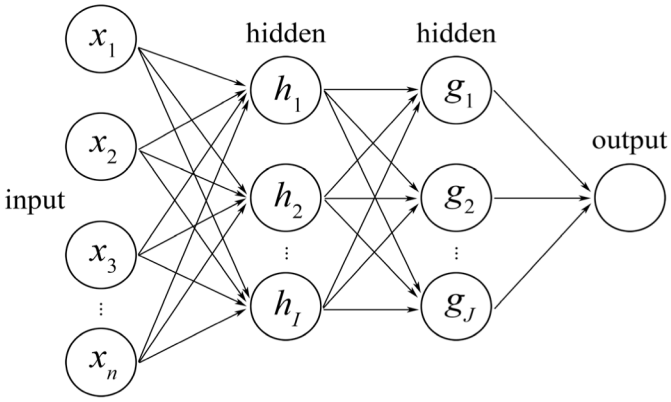

Fig. 1. Typical FNN structure with an input layer and hidden layers.

The artificial NN (ANN) is a supervised machine learning algorithm inspired by the way the brain processes information. An ANN contains many neurons that receive and send signals through their interconnections in a specific way. By training the network with data and optimization techniques, the neurons can find and memorize a pattern that best fits the data. Hinton [2] and Lippmann [3] introduced the ideas and principles behind modern NN techniques in the late 20th century. A statistical perspective of NNs can be found in [4].

NNs have been widely applied to many fields in recent years. Flexible modeling tools are available to freely design a network structure to fit data with both linear and nonlinear patterns. The fully connected NN (FNN) is the basic NN structure. A typical FNN has three parts: an input layer, a hidden layer, and an output layer. For example, the network depicted in Fig. 1 has n input features, I neurons in the first hidden layer, and J neurons in the second hidden layer. Each input feature is connected to all the neurons in the first hidden layer h. These connections follow the equation:

$$h_i = H_i(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n), \quad i = 1, 2, \cdots, I \tag{1}$$

where h_i is the output of a hidden neuron, $x_1 \ldots x_n$ are the input features connected to this neuron, and $H_i(x), i = 1, 2, \cdots, I$ represent I activation functions that can be used to model nonlinear relationships. The outputs of the first hidden

Manuscript received October 16, 2018; revised March 13, 2019, June 5, 2019, and July 31, 2019; accepted August 2, 2019. Date of publication September 4, 2019; date of current version August 4, 2020. *(Corresponding author: Shuai Liu.)*

S. Liu and H. Ji are with the School of Statistics, Capital University of Economics and Business, Beijing 100070, China (e-mail: liushuai@cueb.edu.cn; jihong@cueb.edu.cn).

M. C. Wang is with the College of Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: ccwang@bellsouth.net).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2934110

layer can also be the inputs to the second hidden layer using

$$g_j = G_j(\alpha_0 + \alpha_1 h_1 + \ldots + \alpha_I h_I), \quad j = 1, 2, \cdots, J \qquad (2)$$

where g_j is the output of the second hidden layer and $G_j(x), j = 1, 2, \cdots, J$ are J activation functions. The network can include multiple hidden layers, each of which can use the outputs from previous hidden layers as inputs.

Zhang and Qi [5] applied an FNN to seasonal time series with trends but found that the FNN did not address seasonality and trends well. A preprocessing step involving seasonal and trend adjustments is critical for building an adequate FNN. Jain and Kumar [6] came to a similar conclusion; they found that an FNN is better than AR-type models at capturing the hidden relationships among historical streamflow data. However, they tended to believe that ANNs require the time series to be stationary. Nevertheless, after an empirical study on solar radiation time series was conducted in [7], their experiments showed something different: applying an ANN to nonpreprocessed data gives the best results under some circumstances.

The convolutional NN (CNN) is a special type of NN that introduces convolution and pooling operations to generate deep features, which improves the network's ability to recognize patterns. Many studies have shown that a CNN is a good tool for addressing complex jobs, such as image recognition [8], text recognition [9], and video recognition [10].

Thus, due to the qualities of convolutional layers, it is easy to assume that a CNN is appropriate for seasonal time series with trends. A CNN framework containing convolutional layers, pooling layers, and fully connected layers for time series classification was designed in [11]. This framework performs well in terms of discovering and extracting the internal structure of data.

Based on the information summarized above, we assume that NNs can be applied to nonstationary time series. In this article, a detailed experiment is used to study the application of NNs to time-series data with seasonality and trends. We first attempt to find a method for directly applying an FNN to seasonal time series with trends. Then, we use the FNN as a baseline to study the effects of the convolutional and pooling layers. The goal of this experiment was to investigate the benefits of modeling time series with a CNN and to present several guidelines on how to use a CNN to analyze time-series data with seasonality and trends.

Given the popularity of machine learning, many people are attempting to solve problems by applying machine learning techniques to avoid human labor. A machine learning task usually includes steps such as data cleaning, feature engineering, parameter tuning, and model selection. However, it is unrealistic to expect all users to become proficient at machine learning techniques; thus, methods facilitating the use of machine learning are needed. Many current studies have concentrated on automatic learning. Yan [12] attempted to develop an automatic ANN modeling scheme based on a generalized regression neural network. His ideas about automatic time series learning inspired us greatly. However, his initial study still needed considerable work to address preprocessing steps such as feature selection and parameter selection. Some commercial software packages provide solutions for automatic learning. For example, Data Robot[1] builds a large number of potential models at the same time and then selects the best models based on a given criterion. Yiming Tech[2] seeks to create models that can address multiple data patterns and perform feature engineering and modeling with a single click. As a part of our research on automatic learning, this article seeks to find a modeling method such that users do not need to concern themselves with time series preprocessing.

The original motivation for this article stems from a business project that focused on developing an automatic machine learning tool to predict time-series data without manual intervention. One key problem is to find a modeling method or a general-purpose algorithm that can address seasonality, trends, and autocorrelations in time-series data. This goal was inspired by [5]–[7] and [11], in which the authors compare the performance of ARIMA to those of an FNN and a CNN. Here, we want to use a CNN. However, their experiments and conclusions are inadequate for our needs. Zhang and Qi [5] concluded that an FNN is inappropriate for seasonal time series with trends; however, that study did not consider the use of activation functions and optimizers, which might be essential to the results. In [11], a CNN framework for time-series data was proposed; however, the effects of the convolutional and pooling layers were not reported. In this article, we aim to investigate the effects of each component in a CNN and to consider activation functions and optimizers in an FNN. Through this article, we hope to develop a model or some general guidelines that allow NNs to better identify seasonality and trends. Furthermore, we explore a method for modeling time-series data that does not require human intervention.

The main contributions of this article are as follows. First, we present a feasible method for modeling seasonal time series with trends with an FNN. This approach offers a simple method for fitting time-series data without requiring manual preprocessing. Second, we study and report the effects of the convolutional and pooling layers in a CNN. The results provide an important suggestion when modeling seasonal time series with trends with CNN to avoid pooling layers in the network. Finally, this article provides a theoretical and experimental basis for further automatic machine learning research for time-series data. Traditional time-series analyses require well-trained modelers, i.e., professionals trained in statistics, mathematics, programming, and many other related areas. However, our achievements make it possible to build a network that can automatically identify seasonality, trends, and autocorrelation, allowing time-series analysis to be performed by nonexperts.

This article contains six sections. Section I introduces the background knowledge, origin, and motivation, as well as our main contributions. Section II discusses the theoretical basis for fitting seasonality and trends with NNs. Section III describes the data, network structure, and experimental design, and raises three key questions to clarify the research goals of

[1] https://www.datarobot.com/
[2] http://www.yimming.com/

this article. Section IV reports the results on simulated and real data for each question raised in Section III. Finally, Section V provides conclusions, and Section VI discusses future work.

II. Modeling Seasonal Time Series With Trends

There are two main targets when discussing time series: classification and forecasting. Moreover, many models and algorithms can process time-series data. According to the purpose of this article, only forecasting is discussed and NN modeling is considered. We select the three models listed below.

1) FNN, which contains only fully connected layers.
2) Nonpooling CNN (NPCNN), which includes both convolutional layers and fully connected layers.
3) CNN, which contains convolutional layers, pooling layers, and fully connected layers.

A. Addressing Seasonality and Trends

Seasonality and trends are the most common components of nonstationary time series. We can consider them mathematically. Seasonality can be explained by partial autocorrelation with the following equation:

$$\rho_{X_t,X_{t-k}|X_{t-1},\ldots,X_{t-k-1}} = \frac{E[(X_t - \widehat{EX_t})(X_{t-k} - \widehat{EX_{t-k}})]}{E\left[(X_{t-k} - \widehat{EX_{t-k}})^2\right]} \tag{3}$$

where $\widehat{EX_t} = E[X_t|X_{t-1},\ldots,X_{t-k+1}]$ and $\widehat{EX_{t-k}} = E[X_{t-k}|X_{t-1},\ldots,X_{t-k+1}]$. Equation (3) indicates that the nature of seasonality is the correlation of X_t and X_{t-k} under the condition $X_{t-1},\ldots,X_{t-k+1}$, and k is the length of the cycle. Trends can also be explained by autocorrelation using the following equation:

$$\rho_{X_t,X_{t-k}} = \sum_{i=1}^{n-k} \frac{(x_i - \hat{\mu})(x_{i+h} - \hat{\mu})}{\sum_{i=1}^{n}(x_i - \hat{\mu})^2} \tag{4}$$

where $\hat{\mu}$ is the mean of the population. Equation (4) indicates that the nature of trends is the correlation of X_t and X_{t-k}.

The nature of seasonality and trends hints that their patterns are completely hidden in the series $\mathbf{X}_t = (X_t, X_{t-1},\ldots,X_{t-k}), t = k+1,\ldots,n$. For the model to be able to identify seasonality and trends, we need to create k lag variables from time point $t = k+1$ to the end of $t = n$. Neural networks can automatically recognize patterns of seasonality and trends. Because k represents the length of the cycle, we need to create all the lag variables so that they represent the full cycle; then, we can use NNs to automatically identify seasonality and trends.

B. Convolutional Layer

Generally, we can say that time-series predictability comes from its associated time-lag correlation coefficients. For any time series, there is an equation such as

$$X_t = f(X_{t-1}, X_{t-2},\ldots,X_{t-k}) + \varepsilon_t \tag{5}$$

where f is an unknown function, ε_t is the error term, and k is a parameter determining the memory period. To predict X_t from (5), we need to find an approximate form for the function f. In other words, we need to find sufficient patterns from $X_{t-1}, X_{t-2},\ldots,X_{t-k}$.

A convolutional layer can detect local conjunctions of features from the previous layer [13]. Thus, a convolutional layer is suitable for extracting features or patterns from $X_{t-1}, X_{t-2},\ldots,X_{t-k}$. Therefore, we can expect an improvement by applying a convolutional layer in a time-series model.

C. Pooling Layer

CNNs have had a typical standard structure since LeNet-5 [14]. Stacked convolutional layers are optionally followed by max pooling. The pooling layer has the ability to merge semantically similar features, which allows representations to be robust to the variations that occur when elements in the previous layer vary in terms of position and appearance [13].

Applying a pooling layer in the CNN can shrink the parameter dimensions and create invariance to small shifts and distortions, which is quite important in image recognition. Although pooling obviously loses some information due to shrinkage, CNN models are still successful. Such losses can be ignored in image deep learning [15].

However, when modeling seasonal time series with trends, the effects of the pooling layer are not clear. Time-series data are not as complex as images, even though they may not be appropriate for fitting a deep model. We conjecture that the pooling layer may lose too much useful information when modeling seasonal time series with trends. This article is intended to test that conjecture.

D. Fully Connected Layer

A fully connected layer is a basic structure in NNs. Each neuron is connected to all the neurons in the next layer. By applying different types of activation functions, a fully connected layer can represent any linear or nonlinear relationship. Usually, we put fully connected layers at the end of a sequence of layers because their purpose is to find a good way to summarize all the features extracted by previous layers to generate an output.

Convolutional layers extract patterns from time-series data, including seasonality, trends, and autocorrelation. Fully connected layers should be placed after convolutional layers.

III. Research Design

An FNN has the simplest network structure. As discussed in Section I, we hypothesize that an FNN can be used to model seasonality and trends directly when the activation functions and optimization methods are appropriate. It is important to verify this question because it may lead to a simpler modeling method that does not require seasonal trend decomposition or lag functions selection. Moreover, the selected optimizer and the activation functions of a fit FNN may also be used to fit a CNN.

TABLE I
SEASONAL INDEXES USED BY SIMULATED DATA [5]

	Jan	Feb	Mar	Apr	May	Jun
SI	0.75	0.80	0.82	0.90	0.94	0.92
	Jul	Aug	Sep	Oct	Nov	Dec
SI	0.91	0.99	0.95	1.02	1.20	1.80

CNNs represent a popular modeling methodology in deep learning, especially in image recognition. In most image recognition cases, a convolutional layer is followed by a pooling layer. The convolutional layer extracts features from images, and the pooling layer shrinks the parameter dimensions and fosters invariance to small shifts and distortions.

Generally, we can regard time-series data as a 1-D or flattened image. We extract signals or features from the time series and perform forecasting. Thus, CNNs should also perform well on time series. However, in most cases, especially in financial fields, we do not have many efficient variables. Pooling layers may lose too much information due to shrinkage.

Based on these considerations, the main purpose of this article is to provide empirical evidence for the following three questions.

1) Is an FNN able to model seasonality and trends directly and yield satisfactory forecasts?
2) Does a CNN perform better than an FNN when modeling and forecasting seasonal time series with trends?
3) Are convolutional layers and pooling layers helpful when using a CNN to make forecasts on seasonal time series with trends?

To address these and other related questions, we first conduct a simulation study and then validate our findings using real data. The data used in the simulation study were generated from known seasonal and trend processes, and the real data set used for validation consists of web traffic data from company X.

A. Data

The simulated data are generated using the same method as in [5]. We adopted the same simulated data approach so that we could easily and directly make fair comparisons with the results from [5]. Moreover, the data generated by [5] produce a seasonal series with trends that are highly similar to those in real data. Using a seasonal function such as a sine wave is too simple [11]. The following simple multiplicative model generates the simulated series:

$$y_t = T_t \mathrm{SI}_t + E_t \tag{6}$$

where $T_t = 100 + 0.6t$ represents the trend aspect, SI_t represents the seasonal aspect, and E_t is an error term. The seasonal part is a cycle of 12 steps per year, as listed in Table I. The error term follows a normal distribution $N(0, \sigma^2)$. Four different variance levels are used for the error term ($\sigma = 1$, 5, 10, and 15).

A total of 384 points are generated according to (6) for each of the four variance levels. Fig. 2 shows a plot of the simulated series when $\sigma = 15$. The first 360 points are used for model training, and the last 24 points are used for testing the model's performance.

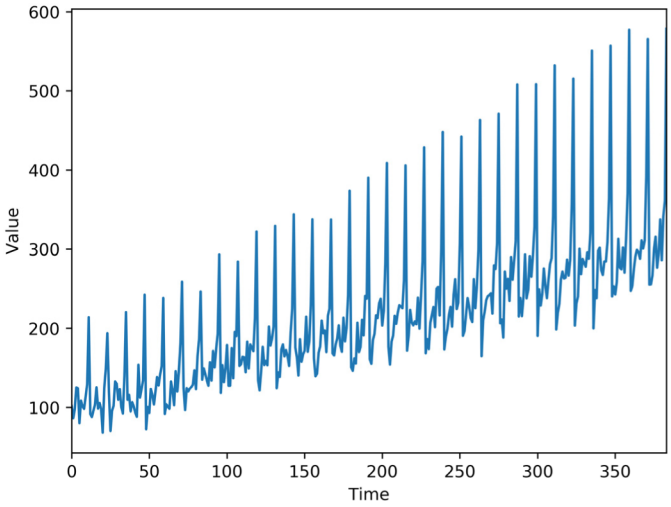

Fig. 2. Plot of simulated time series ($\sigma = 15$).

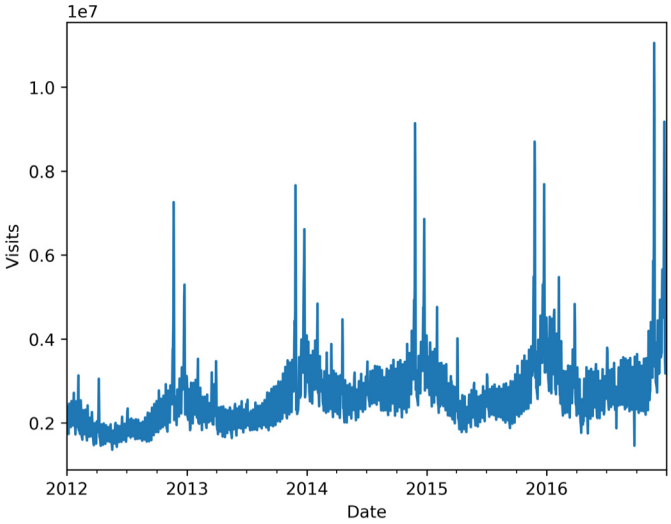

Fig. 3. Plot of real data (daily visits to website X).

The real data include the daily visitor count to company X's website on dates ranging from January 1, 2012 to December 31, 2016. There are 1827 total data points. A plot of the real data is shown in Fig. 3. The data from 2012 to 2015 are used for training, and the remainder of the data is used for testing.

It is obvious that both the simulated data and the real data contain a clear seasonal pattern with increasing trends. Both these time series meet the requirements of this article.

B. Modeling Strategy

Both the simulated data and the real data are univariate time series. To fit NN models, a set of time lag variables spanning the full cycle period length are created as input variables, and the original value of the series is used as the target variable. The lag variables are the series of time t prior to the original

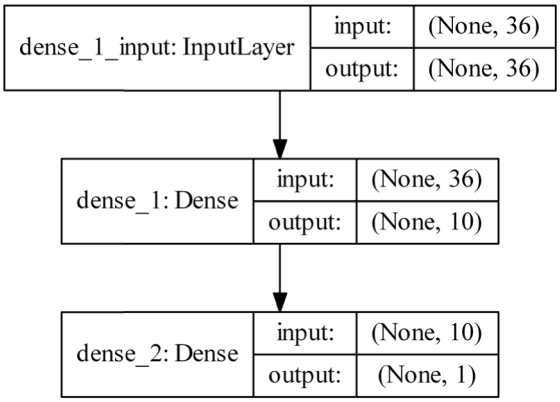

Fig. 4. FNN structure with 36 input neurons and 10 hidden neurons.

series, and they represent a time shift of the original time series, i.e., a historical value of the current time point.

There are 384 points in the simulated series, which has known seasonality with 12 steps. It is a monthly series of 32 years and cycles by year. We used the last 2 years of data from this series for testing and the first 30 years of data for training. We created lag variables with lag numbers from 1 to 36. The first 3 years of data were deleted after the lag variables were created because they did not include all the lag variables. Finally, we used 27 years of data with 36 lag variables for training and the last 2 years of data with the same lag variables for testing. The test data were treated separately by year. By comparing the results of the first and second years of test data, we hope to find a model that achieves good performance in both years. Overfitting can be identified if the model performs well in the first year and badly in the second year.

For the real data, we adopted the same procedures as for the simulated data except that lag variables were created with lag numbers from 1 to 366. The full data for 2016 consisting of 366 points were kept for testing. The data from 2012 including 366 points were dropped because some lag variables were missing.

This article does not need to select the lag variables, as explained in Section II-A; instead, we can simply create all the lag variables that exceed a cycle.

Three models are considered in this article: an FNN, a CNN, and an NPCNN. All the models were trained using the open-source Keras and TensorFlow frameworks.

An FNN exemplifies the simplest NN structure. Each neuron in a hidden layer is connected to all neurons in the surrounding layers. In this article, we use only one type of FNN structure: one hidden layer with ten neurons. We do not test different FNN structures because identifying the best structure of the FNN is not our goal. Finding a relatively good FNN compared with that of [5] is sufficient.

The structure of the FNN used in the simulation is shown in Fig. 4. All the inputs X_i, $i = 1, 2, \ldots, 36$ (dense_1_input) are connected to the hidden layers (dense_1), which each have ten neurons. Then, they are connected to the output layer (dense_2), which has a single neuron. The activation function is a rectified linear unit (ReLU), the optimizer is Adam, the loss function is the mean squared error (MSE), and the identity activation function is used in the output to generate real numbers.

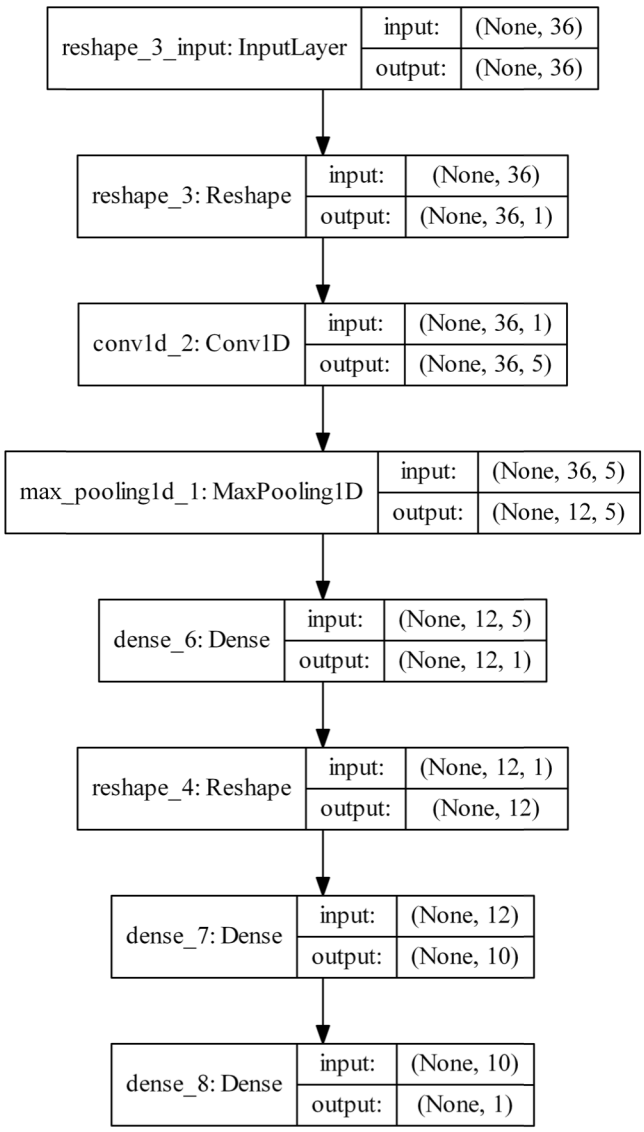

Fig. 5. CNN structure with a convolutional layer of five filters, a max pooling layer of pool size 3, and a fully connected layer as shown in Fig. 4.

The CNN model in this article is based on the FNN model described above. A convolutional layer with five convolutional kernels (conv1d_2) and a pooling layer (max_pooling1d_1) is added to the FNN. The kernel size is 5, and the other parameters are the same as those of the FNN. See Fig. 5 for the model structure of the CNN used in the simulation. We tested two pooling methods, average pooling and max pooling, with pooling sizes of 2, 3, 4, and 6 for each method.

The structure of the NPCNN model is the same as that of the CNN, provided that the pooling layer is removed. See Fig. 6 for the model structure of the NPCNN used in the simulation. Because of the removal of the pooling layer, the fully connected part will have a different number of input neurons (dense_4). With respect to the real data, the modeling strategy is identical except that the number of input neurons is 366.

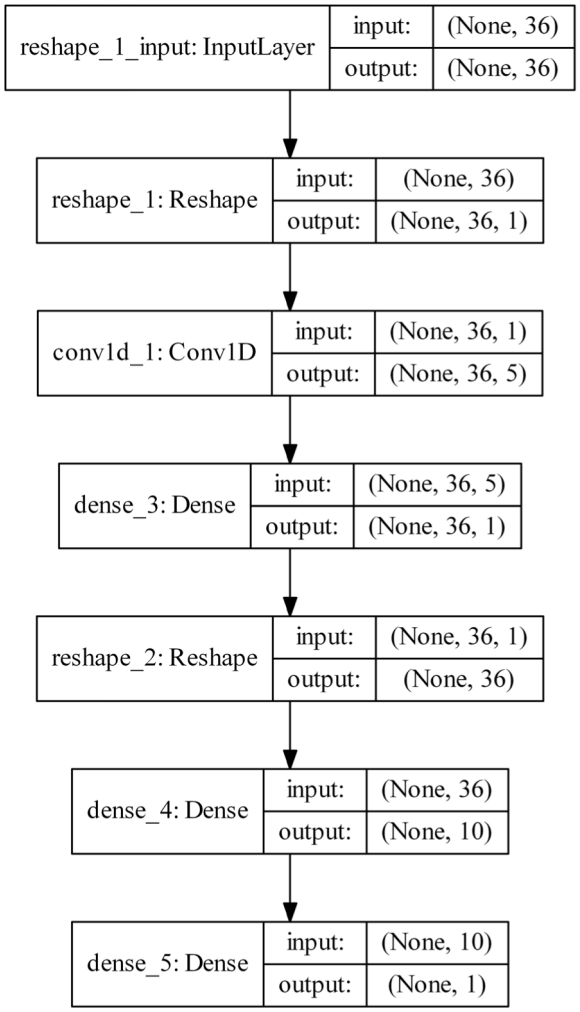

Fig. 6. NPCNN structure with a convolutional layer of five filters. The fully connected layer is the same as that shown in Fig. 4.

We varied the convolutional and pooling layers and set the others to be control variables. Both the simulated data and the real data were tested. This experiment can answer the questions raised in this section.

IV. Empirical Findings

A. Simulation Results

We conducted a simulation using the FNN to verify the first question raised in Section III. Three activation functions, i.e., ReLU, tanh, and linear, and four optimizers, i.e., Adadelta, AdaGrad, Adam, and SGD, were tested. The results of the simulation are shown in Table II. Because the performance of NNs always depends on the initial values, we generated 20 different random seeds as initial values and then summarized all 20 results by taking the mean.

As shown in Table II, Adam and ReLU are the best optimizer and activation function combination for the FNN. The root mean squared error (RMSE) is quite close to the true value of σ, which is 15. Adam optimization and a linear activation function are also good choices with only slightly worse MSE and RMSE values. The tanh activation function is not appropriate in this scenario because it always obtains the worst results. The SGD optimizer does not converge, and a constant value and NaN are generated. As a result, we use Adam and ReLU in the following simulations.

TABLE II
Compare Active Function and Optimizer on FNN ($\sigma = 15$)

Optimizer	Activation	MSE	RMSE	MAE	MAPE
Adadelta	ReLU	474.1722	21.47815	18.14858	0.054112
	tanh	57712.29	240.1282	209.4214	0.639925
	linear	615.3907	24.54555	16.43667	0.054109
AdaGrad	ReLU	284.0534	16.82767	13.48237	0.040861
	tanh	101046.3	317.8759	306.7489	0.954649
	linear	405.1174	19.87711	15.105	0.044385
Adam	ReLU	255.8582	15.96876	12.85148	0.037869
	tanh	88971.52	298.2785	285.6229	0.885695
	linear	286.3559	16.85081	12.72349	0.038669
SGD	ReLU	17391.56	131.877	290.6687	0.948806
	tanh	17373.65	131.8088	127.6329	0.526013
	linear	nan	nan	nan	nan

The results come from 20 different random initial seeds, summarized by mean. All test data are used for evaluation. Adam and ReLU are the best optimizer and activation function combination. The RMSE is quite close to the true value of σ, which is 15.

TABLE III
Modeling Results of the Simulated Data

		FNN	NPCNN
σ=15	best initial seed	8	15
	mse 1	202.7628567	185.732067
	mse 2	288.2572445	208.677631
σ=10	best initial seed	8	15
	mse 1	108.3335604	105.4259218
	mse 2	176.3606485	97.66537248
σ=5	best initial seed	15	11
	mse 1	54.9335324	38.50415162
	mse 2	43.00790151	36.35825157
σ=1	best initial seed	16	0
	mse 1	7.315805427	5.450051351
	mse 2	6.720450939	5.734923454

mse 1=first year of test data; mse 2=second year of test data.

The FNN is sensitive to the error term because its long-term prediction worsens as σ increases. NPCNN has a smaller MSE than FNN at all σ levels, and its performances on both years of the testing data are similar.

To answer the second question raised in Section III, we compared the modeling results of the FNN and the NPCNN. The effects of the convolutional layer were testable because the only difference between the FNN and the NPCNN is whether the convolutional layer is included. We used 20 different initial seeds in this simulation with four noise levels. The test data were processed separately by year. We selected the results of the best initial seeds (those that performed well on both test data sets).

Table III summarizes the simulation results when comparing the FNN and NPCNN with different σ levels. First, the MSE of the FNN is quite close to the value of σ^2, which indicates that the FNN can recognize most of the useful information from the seasonality and trends. However, the FNN is sensitive to the error term; its long-term prediction worsens as σ increases. Second, the NPCNN has a smaller MSE than FNN at all σ levels, and its performances in

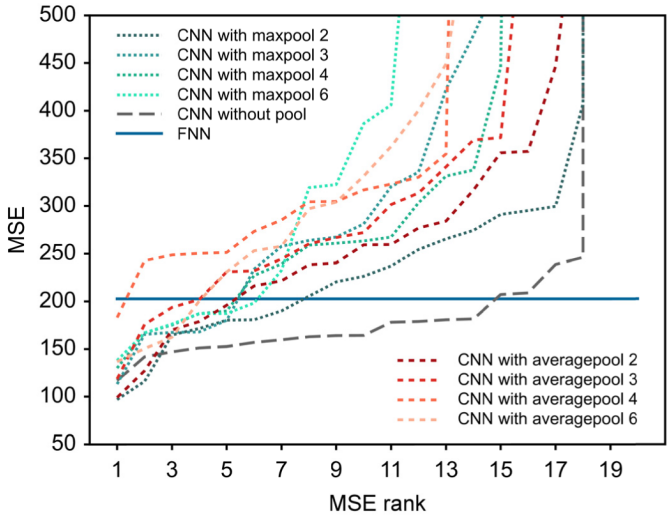

Fig. 7. MSEs resulting from 20 initial seeds on the first year of testing data ($\sigma = 15$). The FNN is extremely insensitive to the initial values, whereas the other networks exhibit substantial changes. The NPCNN can achieve a better forecast through appropriate initial seed selection. Adding a pooling layer to NPCNN always degrades the model performance.

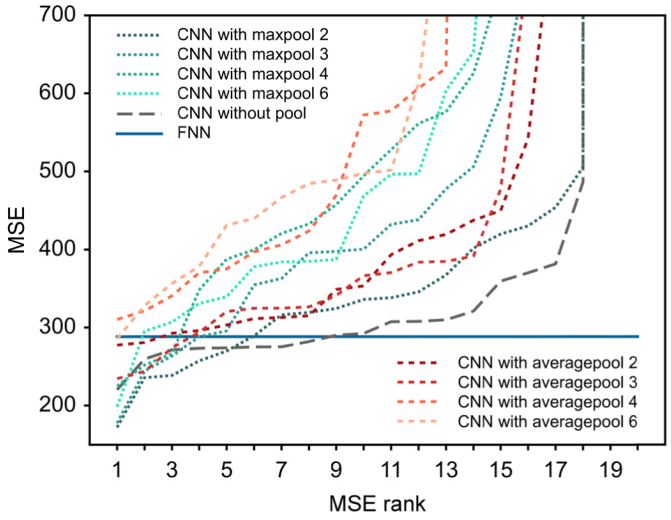

Fig. 8. MSEs resulting from 20 initial seeds on the second year of testing data ($\sigma = 15$). There are some findings similar to those of Fig. 7.

both years of the testing data are similar. Because the only difference between the two models is that the NPCNN has a convolutional layer, the convolutional layer clearly improves the ability to model seasonality and trends.

To determine the influence of the pooling layer, we tested several different types of pooling layers. Each model was trained using 20 different initial seeds. Figs. 7 and 8 show the full results on the simulated data ($\sigma = 15$). There are some similar findings from both the figures.

The horizontal blue solid line, which exhibits almost no change when using different initial values, is the MSE of the FNN on 20 different initial seeds. The FNN is extremely insensitive to the initial values when using the Adam optimizer, whereas the other networks exhibit substantial changes. This property suggests that the performance of the FNN can be

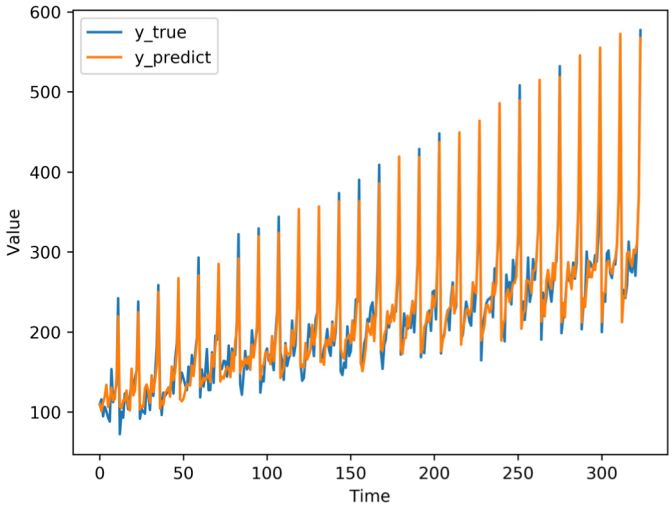

Fig. 9. Predicted values versus true values of the NPCNN on the training data ($\sigma = 15$).

used as a baseline. The lower long-dashed line is the result of the NPCNN. Note that the NPCNN may perform worse than the FNN. Some initial seeds fail to converge. However, the use of many initial seeds results in a much smaller MSE than the MSE produced by the FNN, which suggests that we can achieve a better forecast using the NPCNN with an appropriately selected number of initial seeds. All the dotted and short-dashed lines represent models with pooling layers, and all of them appear above the NPCNN line, which means that adding a pooling layer to the NPCNN always degrades the model performance. Moreover, the lines with large pool sizes always appear above those with small pool sizes, which supports the idea presented in Section II-C that a pooling layer may lose too much useful information when modeling seasonal time series with trends. Although the models with pooling layers have better performance than the FNN and NPCNN with few initial seeds, the models with pooling layers are too unstable, and the need to search for the initial number of seeds suffers from high computational costs.

A comparison between the predicted and true values of the NPCNN on the training data is shown in Fig. 9, which shows that the NPCNN can fit the seasonality and trends very well.

B. Real Data Results

Comparisons of the results of the various optimizers and activation functions on real data are shown in Table IV. The results are similar to those on the simulated data. Adam and ReLU or linear activation functions are the best combination for the FNN. The only difference is that, on the real data, their performances are reversed: Adam + linear is slightly better than Adam + ReLU.

The results of comparing the FNN, CNN, and NPCNN on real data are shown in Fig. 10. Because the visit counts of the website are of a large magnitude (millions), the MSE metric is no longer appropriate for evaluating the models' performances. Instead, we use the mean absolute percentage error (MAPE) when comparing the models. The real data produced results similar to those of the simulated data. The FNN has stable

TABLE IV

COMPARISON OF ACTIVATION FUNCTIONS AND OPTIMIZERS ON THE FNN (FOR REAL DATA)

Optimizer	Activation	RMSE	MAE	MAPE
Adadelta	ReLU	802192.4	1.2E+23	1.03E+17
	tanh	3162788	3004787.	1
	linear	782204.8	476855.6	0.150572
AdaGrad	ReLU	773025.2	458061.4	0.141015
	tanh	3162858	3004861	0.999996
	linear	791296.7	446240.2	0.143336
Adam	ReLU	539550.4	281688.7	0.087561
	tanh	3162841	3004843	0.999990
	linear	517001.7	274930.6	0.086762
SGD	ReLU	nan	nan	nan
	tanh	1027337	3004612	1.000000
	linear	nan	nan	nan

These results stem from 20 different random initial seeds summarized by mean. Adam and ReLU or linear activation functions are the best combination for the FNN.

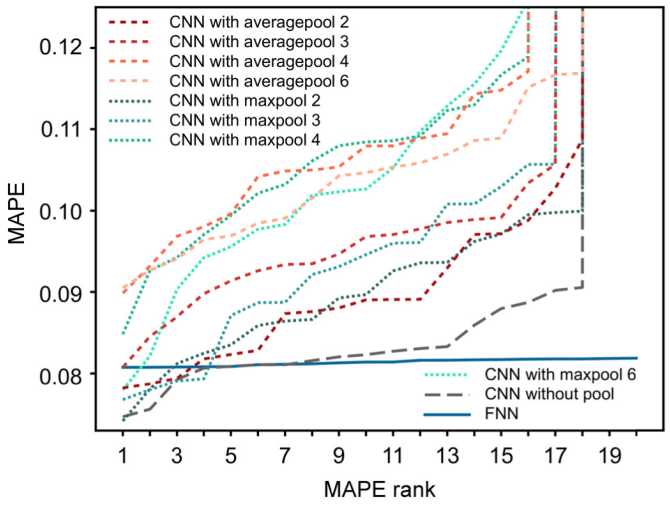

Fig. 10. MAPEs of 20 initial seeds on the web traffic test data. The real data produced results similar to those of the simulated data. The FNN has stable model performance, and including pooling layers degrades the model.

model performance, and including pooling layers degrades the model.

C. Comparison With ARIMA

ARIMA is one of the most popular tools used for traditional time-series analysis. Improvements and optimization of ARIMA have a lengthy history, but some questions remain unanswered. Because of the important role of ARIMA, many studies on NNs have conducted comparisons with ARIMA. Zhang and Qi [5] concluded that NNs with both detrending and deseasonalization can significantly outperform seasonal ARIMA models in out-of-sample forecasting. Jain and Kumar [6] came to a similar conclusion, reporting that ANNs are better than AR-type time-series models for modeling hydrologic time series.

We do not consider ARIMA models in this article because they require manual operations on the parameters (p, q, and d) and statistical tests on stationary data. These manual operations

TABLE V

P-VALUES OF THE ADF UNIT ROOT TEST

	p-value before preprocessing	p-value after preprocessing
σ=1	0.981954	0.000000
σ=5	0.977924	0.000000
σ=10	0.974669	0.000000
σ=15	0.970667	0.000000

Simulated data for each noise level are stationary after preprocessing.

TABLE VI

COMPARISON BETWEEN ARIMA AND FNN ON THE SIMULATED DATA

noise level	data type	model	MAE	MAPE	MSE	RMSE
σ=1	DSDT	ARIMA	1.5152	0.00486	3.17464	1.78175
	O	ARIMA	NA	NA	NA	NA
	O	FNN	1.83386	0.00581	7.00910	2.64747
σ=5	DSDT	ARIMA	3.80742	0.01231	19.7156	4.44022
	O	ARIMA	NA	NA	NA	NA
	O	FNN	4.12453	0.01329	48.9238	6.99456
σ=10	DSDT	ARIMA	7.17617	0.02341	73.9612	8.60007
	O	ARIMA	NA	NA	NA	NA
	O	FNN	6.67836	0.02139	144.696	12.029
σ=15	DSDT	ARIMA	10.9932	0.0354	173.782	13.1826
	O	ARIMA	NA	NA	NA	NA
	O	FNN	11.0927	0.03124	245.321	15.6627

O=original data, DSDT=deseasonalized and detrended data

ARIMA cannot work on the original data; thus, NAs are encountered in the simulation results. The FNN does not perform as well as ARIMA when both seasonality and trends are fully known, which is an unreasonable assumption.

do not satisfy our requirements for automatic learning. However, ARIMA can still act as a reference when evaluating the performance of NNs.

To achieve this goal, ARIMA requires additional preprocessing of the simulated data because it needs a stationary series. We created a 12-lag difference to remove the seasonality and then a 1-lag difference to remove the trends. The augmented Dickey–Fuller (ADF) unit root test results on the simulated data before and after preprocessing are shown in Table V. It is clear that the simulated data for each noise level are stationary after preprocessing.

Because of preprocessing with the difference, ARIMA functions as an autoregressive moving average (ARMA) on the stationary series. The final model is ARMA with $p = 12$ and $q = 1$, which were selected by the Bayesian information criterion (BIC). One-step-ahead and out-of-sample predictions of ARIMA were conducted using all the simulated test data. Table VI compares the model results on the simulated data. The mean absolute errors and MAPEs of both models are quite close on all noise levels, but the FNN performed slightly worse than ARIMA. Moreover, the MSE and RMSE scores of the two models show larger differences. ARIMA achieved much better MSE and RMSE values than the FNN; however, the gap decreases as the noise level grows. Because the ARIMA model was built on preprocessed data while the FNN was built on the original data, the model performance may not be that bad. The FNN still achieves a satisfactory result compared with ARIMA when the noise level is considered. The FNN

sacrifices performance somewhat, but as a tradeoff, it gains the ability to work on raw (nonpreprocessed) data.

However, the comparison given in Table VI is not that reasonable because the two models are trained on different data types; training the FNN on preprocessed data is irrelevant to this article purposes, and ARIMA does not work on nonstationary series. Thus, we should simply treat ARIMA as a reference to indicate how the NNs discussed in this article compare with the well-known ARIMA model.

Because the seasonality and trends are known in the simulation, we were able to remove seasonality and trends completely after preprocessing. Thus, the results of ARIMA on these data almost represent an upper limit. It is not surprising that an FNN trained on nonpreprocessed data would not perform as well as ARIMA trained on preprocessed data. These results indicate that ARIMA has an advantage and can achieve excellent results when the data patterns are known. However, real data usually contain unknown and complex seasonality and trends. In such cases, ARIMA performs poorly and may even fail when insufficient data preprocessing is available. A simple algorithm that can be executed directly on the original data is much more valuable to nonexperts.

V. Conclusion

In the context of automatic machine learning, we expect to find methods that can help people avoid manual preprocessing tasks in time-series prediction. We designed an experiment to address several fundamental questions related to the modeling of seasonal time series with trends using NNs. Although our efforts may not provide comprehensive answers, we can reach some valuable conclusions that are useful for future studies.

In this article, we compared FNNs with different optimizers and activation functions. Then, we compared the FNN with ARIMA on simulated data with different noise levels. The experimental results indicate that even the simplest FNN can model seasonality and trends quite well. Moreover, by using ReLU or linear activation functions and the Adam optimizer, users can ignore time-series decomposition questions if slight model performance losses are acceptable. This conclusion differs from those in [5] and [6], which report that an FNN cannot directly model seasonality. The reason for this difference may be related to the lag variable selection. In Section II-A, we opined that creating all lag variables at sizes larger than a cycle is necessary for NNs to handle seasonality and trends. However, only a few possible lag variables were created by experience in Zhang's study. Another possible reason might be related to the differences between the studies regarding activation functions and optimizers.

Simply adding a convolutional layer to an FNN can improve its ability to address seasonality and trends. We used 20 different random initial seeds to make the comparison. The FNN results are extremely stable, with nearly no changes over all initial seeds. In contrast, an FNN with an added convolutional layer changes substantially over different initial seeds. Approximately one-third of the initial seeds result in model performance that is better than that of the FNN. Convolutional kernels can detect and extract useful features from all lag variables. However, the improvement of the convolutional layer is not as stable because it is sensitive to the initial values. Thus, more effort must be spent searching for initial values that obtain a better model.

In most cases, especially for image recognition tasks, a pooling layer is necessary for a CNN. However, the data complexity of time series is typically far less than that of images. It seems that pooling on time series results in too much information loss, which means that the use of pooling layers is not appropriate. We compared an NPCNN model with the CNN and tested many different pooling layers. However, the performance of a CNN with pooling layers is extremely unstable over different initial seeds and results in performances that are not much better than those of the NPCNN. The experimental results show that the pooling layer has a negative influence when modeling time-series data.

Summarizing the above conclusions, we suggest not using a pooling layer when modeling time series with seasonality and trends. Furthermore, comparing the FNN and NPCNN, the FNN is more sensitive to data variance, while the NPCNN is more sensitive to the initial value. We recommend using an NPCNN if forecasting accuracy is the top priority. Otherwise, a simple and fast FNN is a good choice.

VI. Future Work

Automatic machine learning is currently a hot topic in industry and academia. Many studies have focused on making machine learning available to people who are not experts in machine learning. Data Robot and Yiming Tech, as mentioned in Section I, simplify machine learning for nonexpert use. They help people analyze data, deploy models, and obtain conclusions. In addition, many studies have focused on topics other than the machine learning processes themselves. AutoML from Google is a project that focuses on automatically finding a network structure, making it possible for nonexperts to automatically build a network structure; Autosklearn and Autokeras perform similar tasks.

This article focuses on the question of predicting seasonal time series with trends. We reached several conclusions from the experiments, which are useful for building a network that fits seasonal time series with trends well. Our conclusions on an FNN suggest using particular combinations of optimizer and activation functions, while our conclusions on a CNN suggest adopting the basic network structure of an NPCNN. Given these conclusions, in future work, it will be possible to explore a more complete network structure that can directly fit seasonal time series with trends. Although such a model's predicted performance may not be the best, it could be applied extensively to real-world data sets. Finally, our approach makes it possible to automatically predict time series with machine learning techniques.

The convolutional layer is an important method for extracting patterns from time series. Moreover, recurrent connected hidden layers are also widely used in time-series learning. Han and Xu [16] recently proposed an approach using the Laplacian echo state network (LAESN), which is a novel type of recurrent NN (RNN). Their experimental results ensure

the model's effectiveness for predicting real-world multivariate time series. A further study on the combination of RNN and CNN is another future research direction.

REFERENCES

[1] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. San Francisco, CA, USA: Holden-Day, 1976, p. 575.

[2] G. E. Hinton, "How neural networks learn from experience," *Sci. Amer.*, vol. 267, no. 3, pp. 144–151, Sep. 1992.

[3] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987.

[4] B. Cheng and D. M. Titterington, "Neural networks: A review from a statistical perspective," *Stat. Sci.*, vol. 9, no. 1, pp. 2–30, Feb. 1994.

[5] G. P. Zhang and M. Qi, "Neural network forecasting for seasonal and trend time series," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 501–514, 2005.

[6] A. Jain and A. M. Kumar, "Hybrid neural network models for hydrologic time series forecasting," *Appl. Soft Comput.*, vol. 7, no. 2, pp. 585–592, Mar. 2007.

[7] C. Paoli, C. Voyant, M. Muselli, and M.-L. Nivet, "Forecasting of preprocessed daily solar radiation time series using neural networks," *Solar Energy*, vol. 84, no. 12, pp. 2146–2160, Dec. 2010.

[8] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Trans. Neural Netw.*, vol. 8, no. 1, pp. 98–113, Jan. 1997.

[9] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[10] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

[11] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *J. Syst. Eng. Electron.*, vol. 28, no. 1, pp. 162–169, Feb. 2017.

[12] W. Yan, "Toward automatic time-series forecasting using neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1028–1039, Jul. 2012.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[14] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[15] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9.

[16] M. Han and M. Xu, "Laplacian echo state network for multivariate time series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 238–244, Jan. 2018.

Shuai Liu received the M.S. degree in statistical computing from the University of Central Florida, Orlando, FL, USA, in 2015. He is currently pursuing the Ph.D. degree from the School of Statistics, Capital University of Economics and Business, Beijing, China.

He was with Yiming Tech, Beijing, where he has 2 years of experience in developing commercial data intelligence products. His current research interests include statistics, machine learning, data mining, and financial data analysis.

Hong Ji is currently a Professor with the School of Statistics, Capital University of Economics and Business, Beijing, China. He is the Dean of the Institute for Big Data and Statistical Science, Capital University of Economics and Business. His current research interests include economic statistics, statistical theory, and macroeconomics.

Morgan C. Wang is currently a Professor and the Director of Data Mining with the College of Science, University of Central Florida, Orlando, FL, USA. His current research interests include data mining, nonlinear time series, big data analytics, and intelligent model building.