




Article

Time Series Forecasting and Classification Models Based on Recurrent with Attention Mechanism and Generative Adversarial Networks

Kun Zhou ^{1,2} , Wenyong Wang ^{1,*} , Teng Hu ^{1,2}  and Kai Deng ²

¹ School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; zhoukun@std.uestc.edu.cn (K.Z.); huteng@caep.cn (T.H.)

² Institute of Computer Application, China Academy of Engineering Physics, Mianyang 621900, China; dengkai@caep.cn

* Correspondence: wangwy@uestc.edu.cn

Received: 11 November 2020; Accepted: 13 December 2020; Published: 16 December 2020



Abstract: Time series classification and forecasting have long been studied with the traditional statistical methods. Recently, deep learning achieved remarkable successes in areas such as image, text, video, audio processing, etc. However, research studies conducted with deep neural networks in these fields are not abundant. Therefore, in this paper, we aim to propose and evaluate several state-of-the-art neural network models in these fields. We first review the basics of representative models, namely long short-term memory and its variants, the temporal convolutional network and the generative adversarial network. Then, long short-term memory with autoencoder and attention-based models, the temporal convolutional network and the generative adversarial model are proposed and applied to time series classification and forecasting. Gaussian sliding window weights are proposed to speed the training process up. Finally, the performances of the proposed methods are assessed using five optimizers and loss functions with the public benchmark datasets, and comparisons between the proposed temporal convolutional network and several classical models are conducted. Experiments show the proposed models' effectiveness and confirm that the temporal convolutional network is superior to long short-term memory models in sequence modeling. We conclude that the proposed temporal convolutional network reduces time consumption to around 80% compared to others while retaining the same accuracy. The unstable training process for generative adversarial network is circumvented by tuning hyperparameters and carefully choosing the appropriate optimizer of "Adam". The proposed generative adversarial network also achieves comparable forecasting accuracy with traditional methods.

Keywords: time series forecasting; time series classification; long short-term memory; attention mechanism; generative adversarial network

1. Introduction

In this paper, we focus on time series forecasting and classification tasks using state-of-the-art deep neural network (DNN) models. A time series, $X = X_1, X_2, \dots, X_n$, contains indexed data points in a timely order. It has been widely used in areas such as statistics, pattern recognition, communications engineering, etc. The task of time series forecasting is to establish some model based on the continuous or discrete observed values to forecast the future ones. Forecasting or prediction means transferring knowledge about samples to whole populations in statistics. Time series forecasting is different from conventional supervised learning in that the timely order should be preserved. Good models should extract information as much as possible to achieve "closest to the future" forecasting.

Time series classification aims at classifying new time series to a specific category. Suppose that there are several time series X in each of three categories $C_1 = [X_a, X_{a+1}, \dots]$, $C_2 = [X_b, X_{b+1}, \dots]$, $C_3 = [X_c, X_{c+1}, \dots]$; the task is to establish models and designate the new time series X_n to the correct category C . A large number of machine learning classification algorithms have been proposed, such as logistic regression [1], the naive Bayes algorithm [2], support vector machines [3] and k -nearest neighbors [4], etc. Metrics of accuracy, precision, recall rate and receiver operating characteristic curves, etc. are used to measure the performances and trade-off between true/false positive/negative rates.

The no-free-lunch theorem states that all optimization algorithms, averaged over all optimization problems without resampling, perform equally well [5]. There is no single model that works for all given problems. Therefore, various empirical experiments have been carried out to compare performances in the specified domains. Until now, determining the best model for the specific problem was more of an art than a science.

Inspired by the recent successes of attention-based mechanisms and the temporal convolutional network (TCN) in the area of natural language processing (NLP), we proposed and applied TCN with attention to the time series forecasting and classification based on the fact that NLP and time series share sequential similarity. We also desired to verify the proposition of TCN's superiority to the recurrent neural network (RNN, long short-term memory (LSTM) as the representative) in sequential problems. Applying generative adversarial networks (GANs) to time series forecasting provided new insights and could potentially enhance the understanding of time series. In this contribution, we proposed LSTM with an autoencoder and attention mechanism, TCN models and a GAN model to evaluate and compare their performances for the time series forecasting and classification tasks.

1.1. Time Series Forecasting and Classification

Traditional statistical methods such as the autoregressive integrated moving average (ARIMA) family and exponential smoothing (ETS) were often used for time series forecasting tasks. After surveying 105 academic papers, 28 golden rules of forecasting theory were proposed in [6]. Judgement of orders of autoregressive and time lags for autoregressive moving average (ARMA) and ARIMA were summarized as best practices. Time series could be decomposed to long-term trend, seasonal, recurrent and abnormal variations with additive or multiplicative models. CNN, RNN and the attention mechanism could be used together to avoid the weakness of CNN capturing only local dependency and promoting the strength of RNN in time series forecasting.

While deep learning has achieved huge successes in many applications, only a few time series classification algorithms have been proposed using deep NN [7]. Generative and discriminative methods have been mostly used for time series classification tasks. Autoencoder and echo state networks represent the classical generative methods. A baseline for three end-to-end algorithms, namely MLP, FCN and residual network, without imposing heavy preprocessing on the raw data, was proposed [8]. InceptionTime, a GoogleNet-like NN ensemble model, was proposed for time series classification and slightly outperformed the existing models [9]. Facebook unleashed the "prophet", which improved the controllability and interpretability over traditional models. A comprehensive repository for research in time series classification is hosted in [10]. Fusion of deep NN models for time series represents the future direction and warrants in-depth studying.

1.2. Related Research

A hybrid of ARIMA and ANN models, taking advantage of the strength of ARIMA and ANN models in linear and nonlinear modeling, respectively, was proposed in [11]. A comprehensive review on forecasting methods, such as autoregressive and machine learning methods, for spatial-temporal data from a remote sensing satellite was given in [12]. The near-real-time disturbance detection method was based on least-squares spectral and cross-wavelet analyses and could be used as assessment of the results of time series [13].

Autoencoder frameworks were proposed for sequential modeling and achieved good results [14–16]. RNN or LSTM was used as an encoder to extract features from high-dimensional data and the decoder reconstruct dataset from feature embedding. Their results all showed that the proposed models matched or outperformed the existing methods. Research studies indicated that the dominant models for sequence tasks were mainly based on RNN or CNN with encoder-decoder architecture. Furthermore, the best performing models connect the encoder and decoder through attention mechanisms [14–16]. Attention mechanisms became a research hotspot and they could be applied to a variety of tasks such as machine translation, image caption generation, speech recognition, etc. Attention mechanisms improved neural machine translation (NMT) performances evidenced by BLEU (metrics of translation) scores. The combined architecture of connectionist temporal classification and LSTM (RNN) was proposed, and it achieved the best recorded score on benchmark test [17]. Transformer, the network architecture based on attention mechanisms, was proposed. Experiments presented superiority of the transformer in translation and could be potentially generalized to other sequence modeling tasks [18].

The fusion model of CNN and LightGBM to forecasting is proposed in [19]. Bias and variance for time series forecasting were investigated using the Monte Carlo method. Results suggested that ensemble NN models could potentially improve effectiveness [20]. The common research thoughts for time series classification were to extract features from time series and calculate features' distances [21–25]. Their methods all outperformed other existing models under forecasting metrics.

We surveyed recent influential GAN papers and found that a majority of these papers focused on images, and that several papers focused on video generation; however, there were much fewer papers that focused on generation and time series forecasting with GAN. GAN, the epoch-making framework for estimating generative models via an adversarial process, was put forward. Experiments demonstrated the potential of the framework through qualitative and quantitative evaluation in [26]. Great development of GAN has occurred in recent years. Several hundred variants of GANs have evolved—for instance, Wasserstein GAN (WGAN) [27], SeqGAN [28], Auto-GAN [29] and stacked GAN [30], to name just a few.

WGAN improved the stability of training, eliminated mode collapse, and provided tricks useful for debugging and hyperparameter searches. Numerous papers on improving the unstable GAN training process [31–33] and avoiding convergence failure [34,35] were published, and these problems were still open as of the writing of this paper. Applying GANs to NLP is rather limited due to the difficulty of backpropagation through discrete random variables plus the instability of the GAN training. The authors analyzed distributions that were not differentiable, which caused problems, and they proposed their solutions in [36,37].

Traditionally, RNNs were fit for sequence modeling; however, a recent study suggested that convolutional architectures outperformed them on tasks such as audio synthesis and machine translation. A systematic evaluation of the two architectures was conducted. Experiments showed that convolutional architecture outperformed recurrent networks across different tasks and datasets. The highlight of the paper suggested replacing RNN with CNN to model sequence tasks [38].

Inspired by these up-to-date research works, we proposed the related models to empirically verify the attention mechanism's superiority over RNN and applied GAN for the time series forecasting task.

1.3. Pros and Cons of the Models

RNN addressed the limitation of models having no memory, which is much needed for sequential data. LSTM extends RNN's capability by adding long short-term memory since word's meaning depends on the context. LSTM models were the mainstream technique for translations in the past and could also be used for text generation. Early successes and wide applications accelerated RNN's development. However, RNN is time-consuming and deep NN cannot proceed due to its property of one word/character at a time. The shortcomings of RNN block parallel processing, while TCN has no such limits, which roughly explains TCN's superiority to RNN in terms of efficiency.

Intuitively, CNNs excel at processing image data with geometric attributes. However, it is not ideal for sequence problems because of the filter size limit and failure of capturing long dependence information. Although CNNs are much less sequential than RNNs, the number of steps required to combine information from distant parts of the input still grows with increasing distance.

TCNs have the advantages of parallelism, low memory requirements for training and flexible receptive field size. They outperformed LSTM architectures on a variety of tasks. This architecture can map a sequence of any length to an output sequence of the same length and have longer memory than RNN. The disadvantages of TCN adaptiveness in transfer learning and the amount of history information might limit its development. Recent research boldly claimed that RNN could be even replaced by TCN, which motivates us to verify this proposition.

GAN models have evolved into several hundred variants since its inception. However, the unstable training processes and the lack of evaluation metrics remained unresolved. Although the GAN framework has remarkable performances in area of image, video, etc., the number of academic papers applying GAN to the area of time series forecasting and classification is still limited.

The paper's contributions are summarized as follows:

1. LSTM with autoencoder and with attention mechanism is proposed and applied to the time series forecast modeling. Gaussian sliding window is proposed for the weights initialization in LSTM attention model;
2. Performances of TCN-based NN to model for time series forecasting and classification are evaluated. The proposed models outperform the classification methods such as 1NN-DTW, BOSS and WEASEL;
3. A GAN model with LSTM as the G and MLP as the D for time series forecasting task is proposed and the performances are evaluated. Comparisons are made with the statistical ARIMA models.

The organization of the paper is as follows. In Section 1, we briefly introduce the background information and related research studies. Methodologies are presented in Section 2. Proposed models are tested and results are given in Section 3. Conclusions and possible future developments are discussed in Sections 4 and 5, respectively.

2. Methodology

2.1. Basics for the Proposed Models

RNN/LSTM models were used for time series forecasting modeling in the past due to its recurrent and autoregressive structure. The basic LSTM cell [39] is illustrated in Figure 1a and Equation (1). The LSTM cells are building blocks of the input layer, hidden layer and output layer of the deep NN. Autoencoders employ the unsupervised learning method to extract features from high-dimensional data and to reconstruct them from feature representations. They consist of encoding and decoding processes, where encoding maps the input to the feature spaces and then decodes back to the original spaces.

$$\begin{aligned}
 o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{output gate}) \\
 h_t &= o_t * \tanh(C_t) \\
 C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \quad (\text{new state } C_t) \\
 \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (\text{candidate state } \tilde{C}_t) \\
 i_t &= \sigma(w_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{input gate}) \\
 f_t &= \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{forget gate})
 \end{aligned} \tag{1}$$

Recent study suggested that RNNs could be replaced by models such as attention-based NN. In practice, LSTM-based neural machine translation (NMT) has been replaced by Google, Salesforce, etc.

Wave-net proposed by Google for speech synthesis and CNN-based models for machine translation by Facebook outperformed other models. The additive and multiplicative attention functions were introduced in [18]. The attention mechanisms map vectors input (query, key, values) to a vector output, which is computed as a weighted sum. The scaled dot-product attention detailed on the computation of attention function where queries were packed into a matrix Q , keys and values into matrices K and V . Query vector $q^i = w^q a^i$, thus $(q^1, q^2, \dots, q^n) = w^q(a^1, a^2, \dots, a^n)$, Key vector $k^i = w^k a^i$ ($k^1, k^2, \dots, k^n) = w^k(a^1, a^2, \dots, a^n)$, and Value vector $v^i = w^v a^i$, ($v^1, v^2, \dots, v^n) = w^v(a^1, a^2, \dots, a^n)$, $a_{1,1} = k^1 q^1$, $a_{1,2} = k^2 q^1 \dots$ We gave the A matrix computation (the computation for K and V were similar) in Equation (2) and Figure 2. $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$.

$$A = \begin{bmatrix} a_{1,1} & a_{2,1} \dots & a_{n,1} \\ a_{1,2} & a_{2,2} \dots & a_{n,2} \\ \dots & \dots & \dots \\ a_{n,1} & a_{n,1} \dots & a_{n,n} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ \dots \\ k^n \end{bmatrix} \begin{bmatrix} q^1 & q^2 \dots & q^n \end{bmatrix} = K^T Q \tag{2}$$

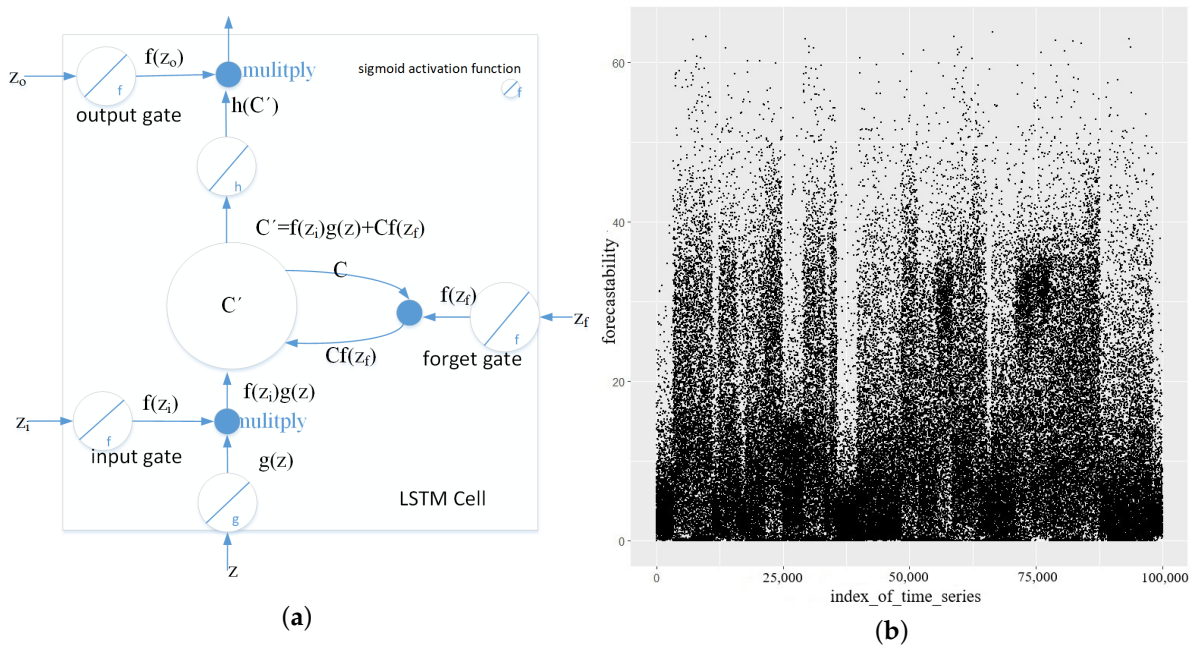


Figure 1. (a): Basic LSTM cell. (b): Overall Kaggle web traffic dataset forecastability.

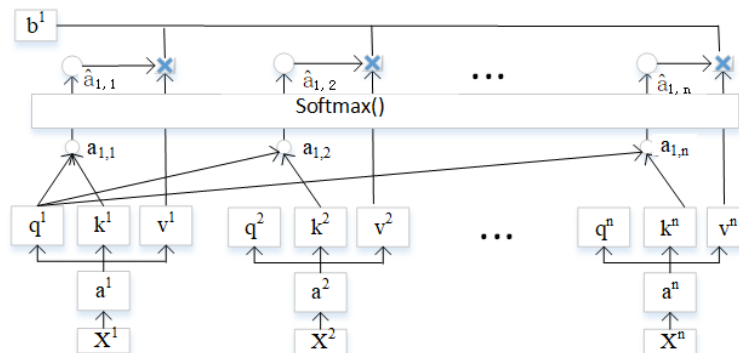


Figure 2. Basic attention mechanism.

The TCN model combining dilated and causal convolutional layers that was presented ensured that the previous time step will not use future information because the output of time step T_t was

obtained based on T_{t-1} . TCN can be viewed as 1D FCN plus one-dimensional causal convolutions, and each two convolutional layers and identity mapping are encapsulated into a residual module. The residual module then stacks the deep network, and uses fully convolutional layers in the last few layers. Dilated convolution enlarges the receptive field exponentially [38]. We simplified the TCN model's highlights using Equation (3). Dilated convolution operation F on elements of the sequence X is defined as:

$$F(s) = (X *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot X_{s-d \cdot i} \quad (3)$$

where $X \in R^n$, $f : \{0, \dots, k-1\} \rightarrow R$, d stands for dilation rate and f for filter size. For the GAN structure, there are two NNs, Generator (G) and Discriminator (D). G and D contest with each other in which G generates candidates while D evaluates them. G 's training objective is to increase the error rate of D . The training could be expressed by the classical value function [26] demonstrated in Equation (4).

$$\underset{G}{\text{Min}} \underset{D}{\text{Max}} = E_{x \sim p_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (4)$$

where G 's distribution is represented by p_g over data x , a priori on input noise variables is $p_z(z)$, G is a differentiable function represented by a multilayer perceptron (MLP) and $D(x)$ represents the probability that x comes from the data rather than p_g . The maximizing of assigning correct labels for D and minimizing of $\log(1 - D(G(z)))$ are simultaneously trained.

2.2. Datasets

Public datasets were collected for the proposed models. Kaggle web traffic competition was used for the time series forecasting task [40]. We evaluated the performance of the proposed LSTM, TCN and GAN models using this dataset. ECG, a multivariate and classified dataset, was used to test the performance of time series classification task with the TCN model. The dataset has 19 features, 150 time steps and 20 classes—that is, 19 features were collected for 150 time steps and they were classified into 20 classes. The VPN-nonVPN dataset from the Canadian Institute of Cybersecurity was used for time series classification, which can be found at [41].

The adding problem has been used repeatedly as a stress test for sequence models [39]. MNIST images are presented to the model as a $784 * 1$ sequence for digit classification [42]. The memory-copying task is to generate an output of the same length as the input sequence with predefined restraints [43].

The dataset should generally be preprocessed before entry into the models. Typical preprocessing consists of center and scale, impute for missing value, the dealing of imbalanced classification problems (many more samples for one class and less for other), or the wrong labels, etc. There was some missing value for the whole web traffic dataset and we deleted those missing value datasets and used 100,000 for the tests. The ECG dataset was perfectly labeled and no extra preprocessing was needed, and for the VPN-nonVPN dataset, the scaling and centering was used.

The forecastability computation was defined using spectral density in Equation (5). We calculated the forecastability of the Kaggle web traffic dataset of 100,000 time series and plot them in Figure 1b. Each dot in Figure 1b represents one time series' forecastability. The larger value of forecastability, the easier to forecast. For example, the value of white noise nearly reaches 0, which means it's hard to forecast. We obtained the whole picture of dataset forecastability from reading Figure 1b.

$$\omega(x_t) = 1 + \frac{\int_{-\pi}^{\pi} f_x(\theta) \log f_x(\theta) d\theta}{\log 2\pi} \quad (5)$$

where x_t stands for stationary process, $f_x(\theta)$ for normalized spectral density of the time series. In particular $\int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1$.

2.3. Long Short-Term Memory with Autoencoder and Attention

The LSTM with autoencoder was proposed to forecast the web traffic. The input was preprocessed as time series data and the output were the 500 training and the forecasting horizon of 50 steps forward (90% of the dataset was used for training and the remaining for test). We deleted the time series with missing values. We designed one encoder and one decoder as illustrated in Figure 3. The left branch from the first LSTM layer up was the encoder and the right was the decoder structure. Input was repeated after the first LSTM layer to enter the encoder and decoder layer, respectively. We defined a predict layer and tied all these together to establish the models. The basic LSTM layer was used for each encoder and decoder with 100 neurons and RELU activation. We compared the performances of losses, and the training time with five different optimizers (“Adadelta”, “Adagrad”, “Adam”, “RMSprop” and “SGD”) using the same loss function (mean_squared_error) and learning rate. The test was repeated fifty times with the same parameters to avoid occasional bias.

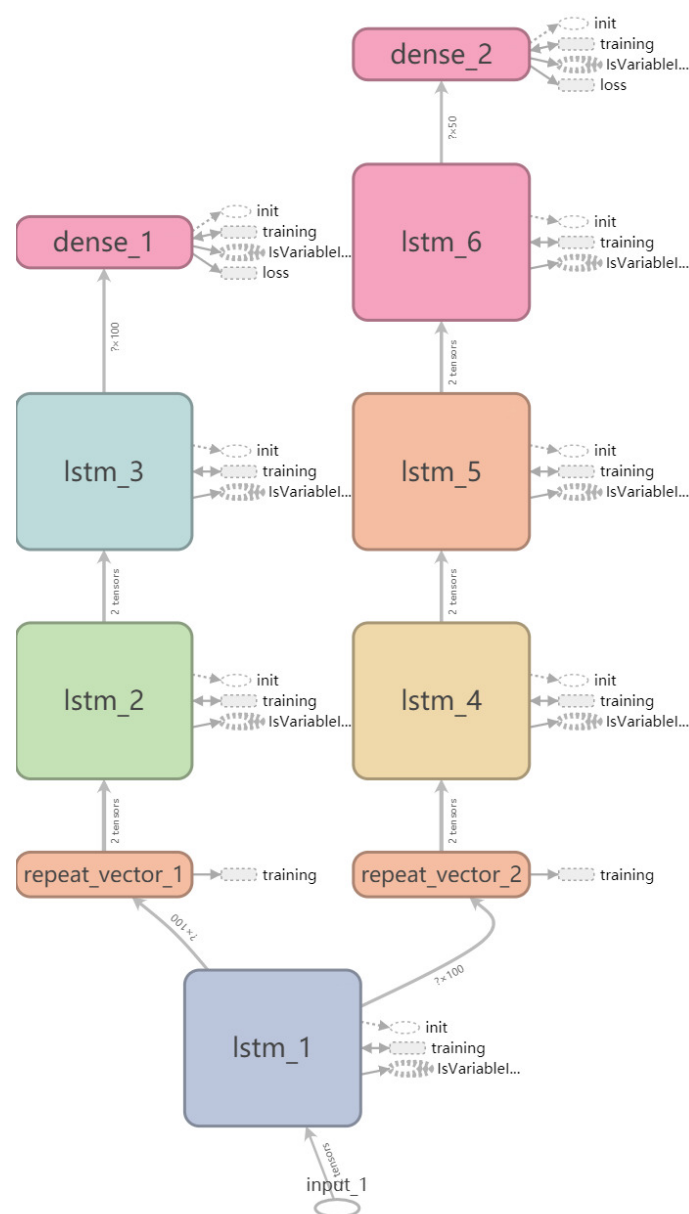


Figure 3. Architecture for LSTM with autoencoder.

Inspired by the “sandwich transformer” of reordering the sublayers of more self-attention (S) toward the bottom and more feedforward (F) toward the top perform better [44], we designed the balanced architecture, which consist of four self-attention sublayers, four feed forward sublayers and reordered them from “FSFSFSFS” to “SSFSFSFF”. The proposed architecture of attention (Figure 4) was based on autoencoder framework with S layers and F layers as the encoder, and masked S, F, and encoder-attention layer as the decoder.

We proposed using the sliding Gaussian windows as the initial weights for the attention layer (Figure 5). The length of input vectors was n and the weights were randomly allocated for basic attention. Gaussian distribution for the weights was based on that distant part of the sequence might play a decreasing role. The parameters of a_k , b_k and c_k were optimized gradually using back propagation through time during training. The Gaussian windows vectors were then input into the LSTM with attention network demonstrated in Figure 5. Layer normalization reduced the “covariance shift” problems faced by batch normalization through fixing the mean and variance of the summed inputs within each layer [45]. We designed one RNN layer normalization between S and F layer.

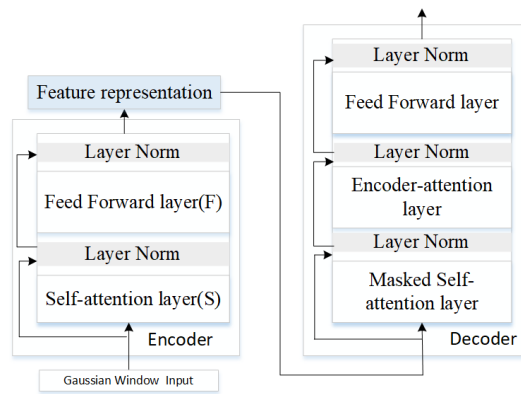


Figure 4. Transformers with attention mechanisms.

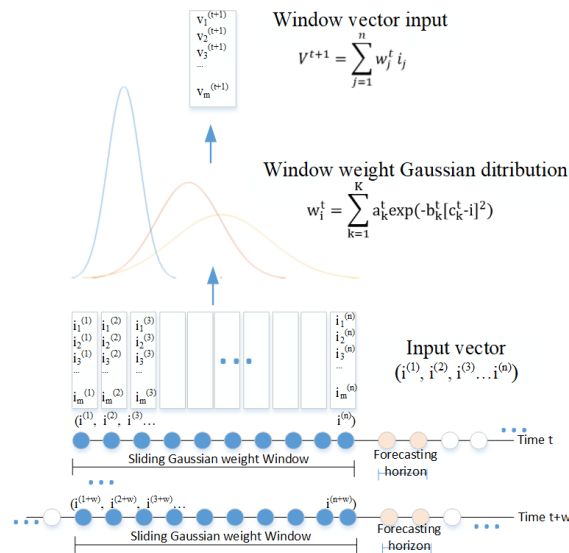


Figure 5. Gaussian sliding windows for initial weights in attention NN.

Activation function of “Sigmoid” and “Tanh” were used in the basic LSTM cells (see Equation (1)). The basic function $\mathbf{softmax}(x_i) = e^{x_i} / \sum_j e^{x_j}$ might suffer from computational underflow (the value of numerator was very small) or overflow (the value of denominator was approaching zero). We proposed using the following equation $\mathbf{softmax}(\tilde{x}_i) = e^{x_i - \max(x_i)} / \sum_j e^{x_j - \max(x_j)}$. The value of denominator was at least more than $e^0 = 1$ and the numerator becomes larger so this problem was avoided. The optimizers

selection influenced the training processes and the results, so five candidate optimizers were tested to find the most suitable one for the proposed models and dataset. Performances of losses, mean squared error rate and the training time were reported.

2.4. Temporal Convolutional Network Architectures for Forecasting and Classification

TCN model with dilations was proposed for time series forecasting and classification task. TCN uses a 1D-FCN architecture where each hidden layer is the same length as the input layer, and zero padding of length is added to keep subsequent layers the same length. Causal convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer. Those features differentiate TCN architectures from FCN and CNN models. We illustrated the TCN models with dilation rate set to 4 and filter size to 3 in Figure 6. The number of CNN layers (convolutional 1D + activation + max_pooling) defined the TCN model architecture. To make the training process stable and achieve better performance fine-tuning the parameters of filter size, kernel size, pooling_size and dropout rate were crucial. The number of hidden CNN layers depends on the characteristic of the targeted data, which has no golden-rules and trial and error method were often used. We observed the dataset and constructed the TCN model with the number of hidden CNN layers set to 2, 3, 4, 5 which correlated with the filter size length. The flatten and dense layers were then added for the convolutional layers. The filter size, kernel size, pooling_size and drop out rate are all hyperparameters, which play important roles for the models' performances. We proposed the grid search method and using the RMSE score to find the best fit parameters' values. These four parameters, namely, dilation rate of [8,16,32,64], filter size of [16,32,64,128,256], kernel size of [2,3,4,5], pooling_size of [2,3,4,5] and dropout rate of [10%,20%,40%,60%], were set for the tests, which added up to 1280 ($4 \times 5 \times 4 \times 4 \times 4$) possible combinations. The candidate parameters which led to the unstable training process was deleted. For each of the combinations five optimizers were tested. The RMSE score was used to measure the performance. The combination of dilation rate of 64, filter size of 128, kernel size to 5, pooling_size to 2 and dropout rate to 20% for the regularization was chose for the model's parameters.

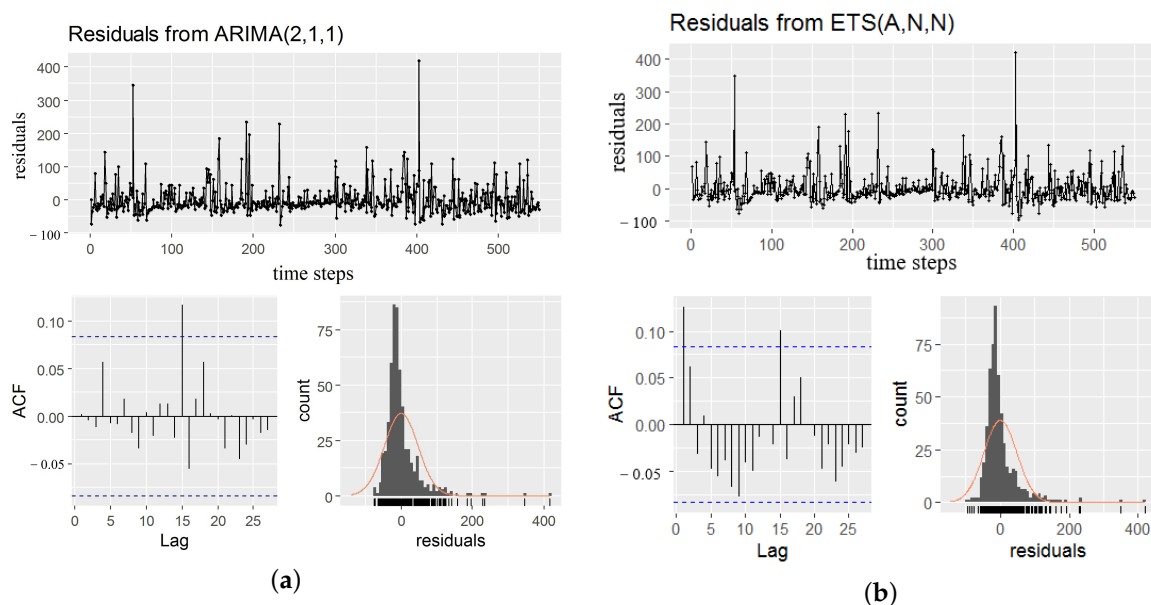


Figure 6. Comparisons between ARIMA and ETS models (a): ARIMA model, (b): ETS model.

2.5. Comparisons with Statistical and Machine Learning Methods

We constructed the statistical ARIMA and Exponential Smoothing (ETS) model to compare the performance with the proposed TCN models. Parameter p for AR, q for MA and d for difference orders determine the ARIMA model. For example ARIMA(2,1,3) means after processing of one order differencing, combines AR(2) model and MA(3). There are best practices for setting the values of these parameters such as observing the (partial) autocorrelation of the dataset, identifying the stationarity with differencing and the model selection (AR, MA, ARMA or ARIMA model), etc. After the model was selected, the parameters estimation process began. We followed the steps to construct two candidate models, compared the accuracy using two different parameter of p,d,q set to (2,1,2) and (3,0,0) for the same Kaggle web traffic dataset. Metrics showed ARIMA(2,1,1) was slightly better than ARIMA(3,0,0). ETS can be simplified as additive (A) or multiplicative (M) model. As for the target dataset, the additive model was more appropriate. We modeled the time series with ETS (A,N,N) (simple exponential smoothing with additive errors) and calculated the criteria. The analysis results of these two kinds of models were illustrated in Figure 7.

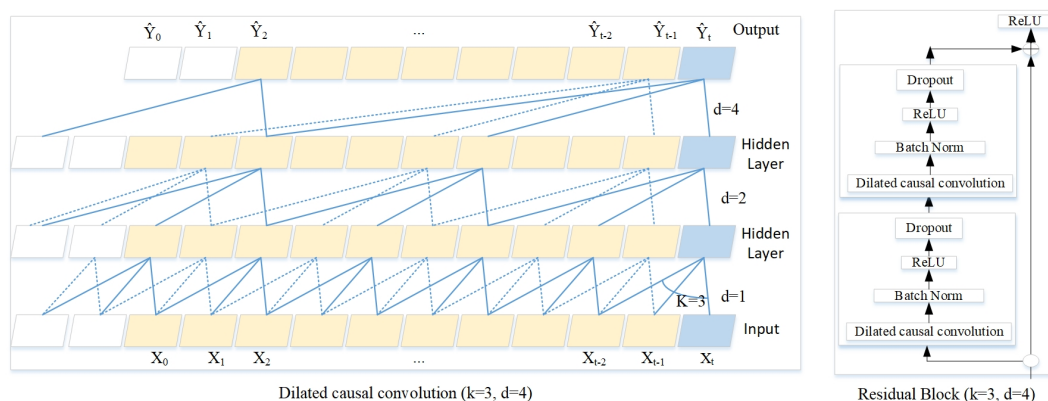


Figure 7. Dilated causal convolution with dilation rate $d = 4$, filter size $k = 3$ and residual block.

We compared performances of four classical machine learning classification algorithms which are Random Forest (RF), Gradient-Boosting tree (GB), Extra tree (ET) and Bagging methods (BA) with the proposed TCN model using the VPN-nonVPN dataset with the Brier scores in Figure 8. The Brier score (vertical axis in Figure 8) measures the accuracy of a probability forecast ranging from 0 to 1. $BS = \frac{1}{N} \sum_{t=1}^N (f_t - o_t)^2$, where f_t is the forecast probability, o_t is the outcome. The score of 0 stands for complete accuracy and 1 means the forecast was totally inaccurate. The results were illustrated using box-plot from which we concluded TCN model outperformed the others. Although GB method might perform better than TCN in some cases, the mean and the range of the scores was below the TCN model.

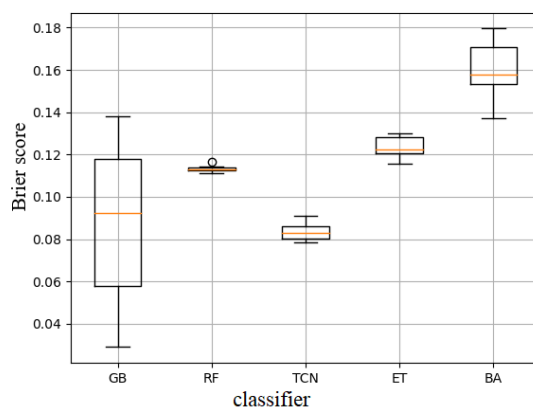


Figure 8. Performances comparisons among five models for time series classification task.

2.6. Generative Adversarial Architecture for Forecasting

The proposed GAN model was designed with the MLP as the G and the stacked LSTM as the D was for the time series forecasting task. The structures of G and D were given in Figure 9. We trained the model on the same web traffic dataset for 10,000 epochs with batch size set to 500 using the five optimizers ('Adadelta', 'Adagrad', 'Adam', 'RMSprop', and 'SGD'), respectively.

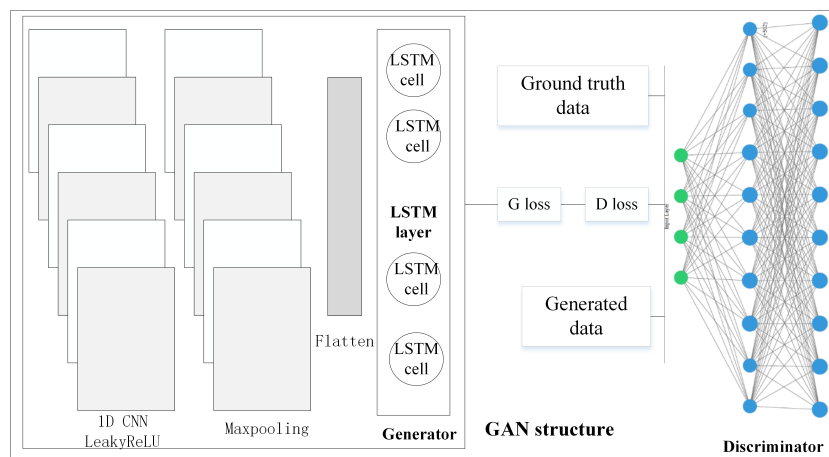


Figure 9. GAN structure for time series forecasting task.

3. Results and Discussion

The metric of loss we used for all the experiments was Mean Squared Error. Forecasting error for data i : $e_i = x_i - \hat{x}_i$, where \hat{x}_i represents the forecast data i . SSE stands for sum of squared error and RMSE root mean squared error. The common metrics were defined in Equation (6) and used for TCN model designed for time series classification task. P, N, TP, FP, TN, FN stand for positive, negative, true positive, false positive, true negative, and false negative, respectively. For the optimizer of Adam, we set the parameters [46] of the algorithm: step size $\alpha = 0.01$, exponential decay rates for the first and second moment estimates $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-6}$.

$$SSE = \sum_{i=1}^n e_i^2$$

$$RMSE = \sqrt{\frac{1}{n} SSE}$$

$$Accuracy = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN} \quad (6)$$

3.1. Long Short-Term Memory with Autoencoder

We used the same web traffic dataset and metric to evaluate the performances for this model.

We trained 100 epochs for each of five optimizers for fifty times. The results suggested that around 10 min per optimizer and the optimizer of 'Adam' and 'Adagrad' performed best with nearly the same RMSE score. Although the autoencoder model was more complex than the vanilla LSTM, we found the time required for training was only increased by less than 2 min and the RMSE scores slightly decreased.

3.2. Attention-Based Long Short-Term Memory

Kaggle web traffic dataset was used to evaluate the performances for this model. We trained 100 epochs for each optimizer for fifty times and found that training time was around 2 minutes per optimizer and the best performed optimizer was Adam. This model improved the performances by cutting training time by around 80 percent compared to vanilla LSTM and the error rate of RMSE also decreased (See Figure 10 and Table 1 for the results).

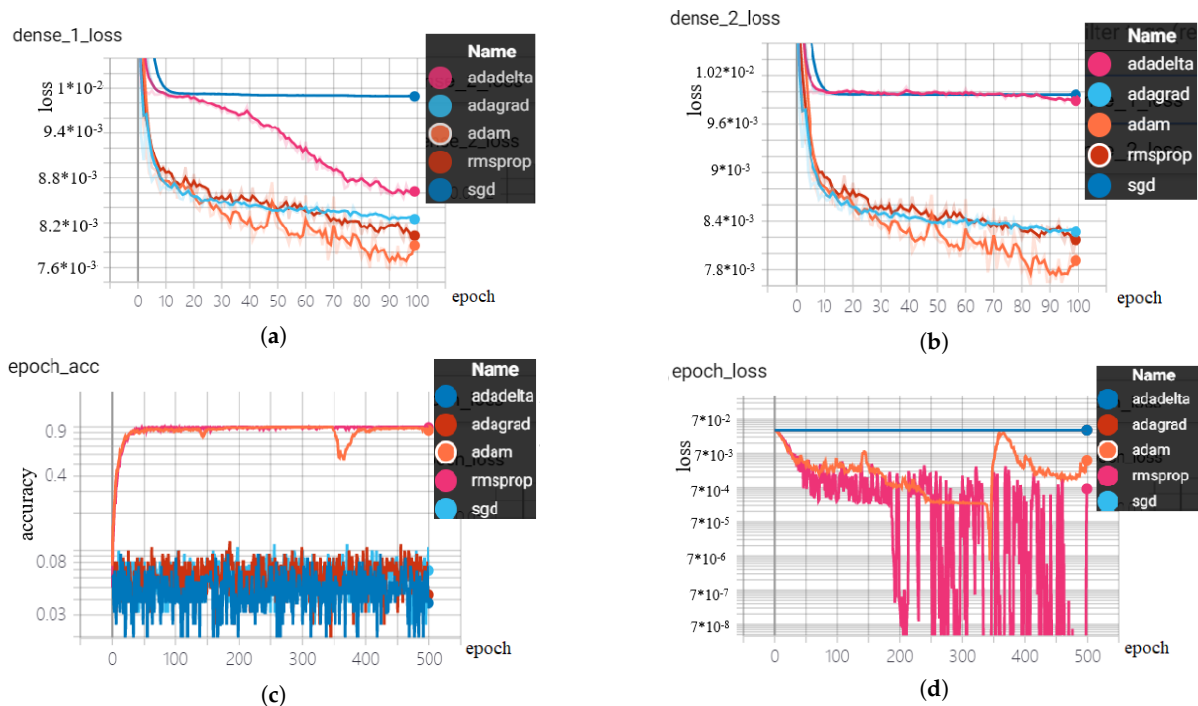


Figure 10. LSTM with autoencoder test results for time series forecasting task (a): encoder's loss, (b): decoder's loss, LSTM with Attention test (c): accuracy, (d): loss.

Table 1. Test results of web traffic forecasting with different optimizers 'training/test losses (time)'.

Optimizer	LSTM	LSTM-auto	LSTM-att	TCN
Adadelta	57.98/49.23 (10 m 34 s)	51.57/39.34 (12 m 30 s)	46.77/40.12 (1 m 50 s)	49.34/40.15 (41 s)
Adagrad	59.21/51.34 (11 m 02 s)	45.74/40.83 (12 m 29 s)	46.77/41.29 (1 m 48 s)	46.24/39.94 (42 s)
Adam	51.20/46.91 (10 m 29 s)	45.21/40.60 (11 m 53 s)	45.12/40.40 (1 m 29 s)	45.82/39.61 (31 s)
Rmsprop	55.23/51.29 (11 m 38 s)	45.81/40.38 (13 m 42 s)	46.34/40.52 (1 m 39 s)	45.85/40.27 (45 s)
SGD	56.54/52.38 (11 m 44 s)	51.79/39.47 (13 m 36 s)	50.32/40.54 (1 m 52 s)	52.92/41.76 (44 s)

Table 2. Test results for different tasks with the proposed models.

Sequence Modeling	Model Size	LSTM	LSTM-auto	LSTM-Att	TCN
Adding problem (loss)	70 K	0.175	6.2×10^{-3}	6.1×10^{-3}	5.9×10^{-3}
MNIST (accuracy)	70 K	87.2	89.7	95.2	98.7
Music MIDI data (loss)	500 K	0.0822	0.0755	0.0671	0.0635
Copying memory (loss)	16 K	0.0301	0.0204	0.0198	0.0182
Kaggle web traffic (RMSE)	10 K	49.83	48.81	46.92	47.12
ECG classification (accuracy)	10 K	95.8	98.6	98.2	99.5

3.3. Temporal Convolutional Network

The TCN model for time series forecasting task used the same web traffic dataset as we did in the LSTM autoencoder and attention mechanism experiments. Training time for TCN was the least among all the models with less than 1 min per 100 epochs, and the accuracy also improved compared to vanilla LSTM (see Figure 11). The second TCN model was for time series classification task of classifying of multi-variate, multi-class time series using the ECG dataset which achieved 99.5 percent of accuracy (see Table 2).

The tests showed different optimizers impacted significantly in the performances of accuracy and loss. The overall training time for each optimizer was around 3 min for 500 epochs (less than one minute for 100 epochs in Table 1).

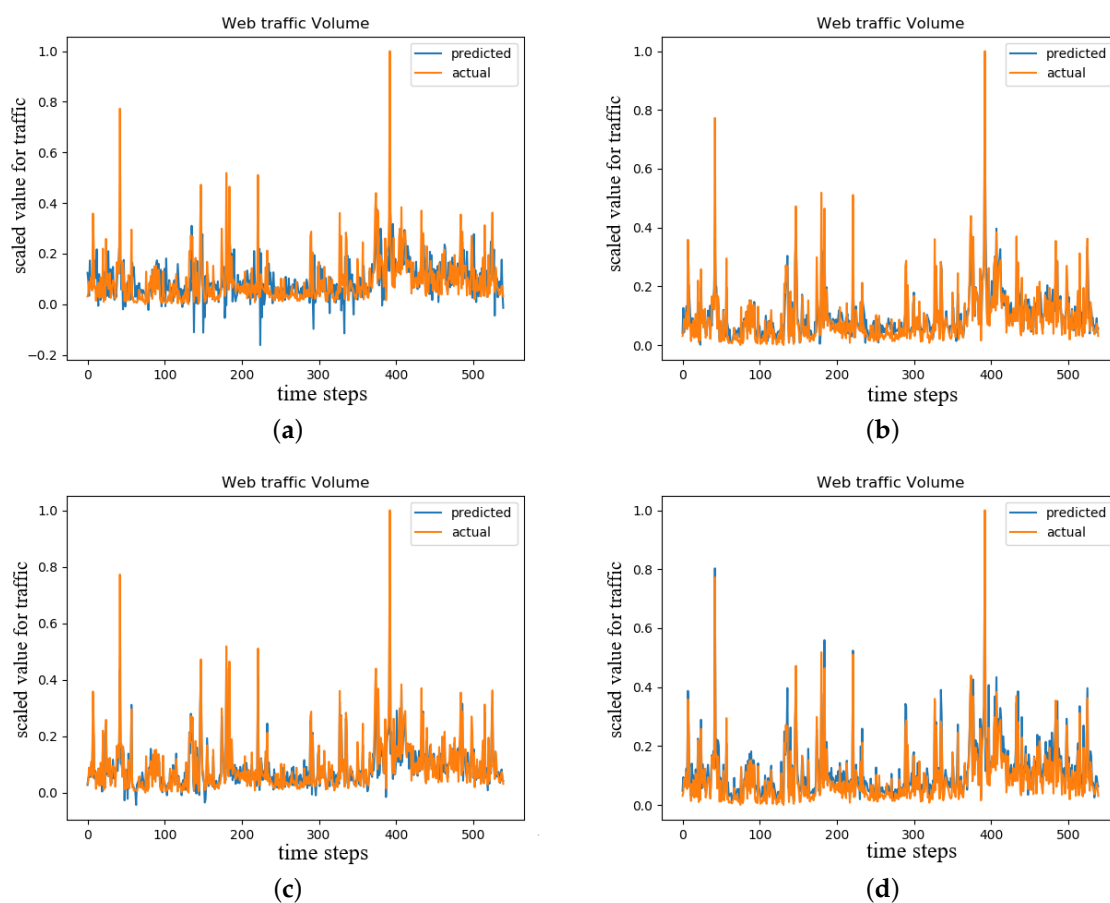


Figure 11. TCN with dilations test results for time series forecasting task (a) adadelta, (b) adagrad, (c) adam, (d) rmsprop.

The best performed optimizer was 'RMSprop' which improved the loss rate to an order of magnitude. TCN's efficiency was the best among the proposed models. The performance of TCN model for time series classification task was compared with the classical models of 1NN-DTW, WEASEL and BOSS using the VPN-nonVPN dataset. From the calibration curve in Figure 12 we could see that TCN was closest to the perfectly calibrated dashed line which indicated the proposed TCN model outperformed the aforementioned models.

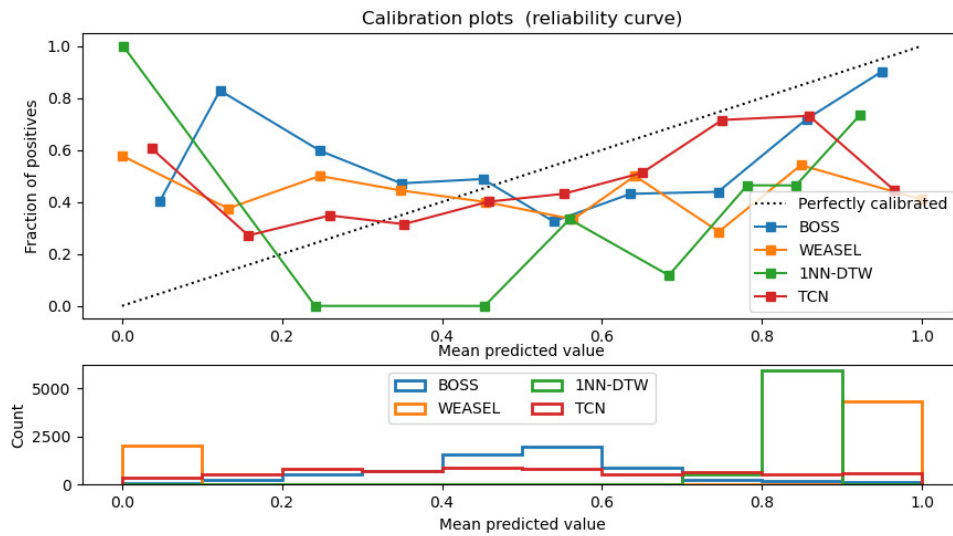


Figure 12. models calibration comparisons among 1DNN-DTW, BOSS, WEASEL and TCN.

3.4. Generative Adversarial Network

A GAN model for time series forecasting was proposed to model the Kaggle web traffic. The 500-time steps were used for training and the remaining 50-time steps for test. We set the same parameters (epochs, learning rate) for the models and loss functions as mean_squared_error for all the time series forecasting experiments. The detailed parameters were shown in Figure 13.

The same web traffic was used as the input to this GAN model. After 2000 epochs of training we plotted the results of the generator loss (blue curve), discriminator loss (green curve) and the discriminator accuracy (red curve) for different optimizers in Figure 14. The discriminator accuracy converged for optimizer of “Adadelata” and “Adam”. When using optimizer of “RMSprop” (c) and “SGD” (d), the discriminator accuracy arrived at around 50%, which meant that it had a relative high success rate of distinguishing the real dataset from generated ones.

This finding confirmed that training processes of GAN sometimes cannot converge to the satisfying minimum. Carefully choosing optimizers such as “Adam” and fine-tuned the corresponding parameters, the accuracy of discriminator converged to the minimum (see Figure 14).

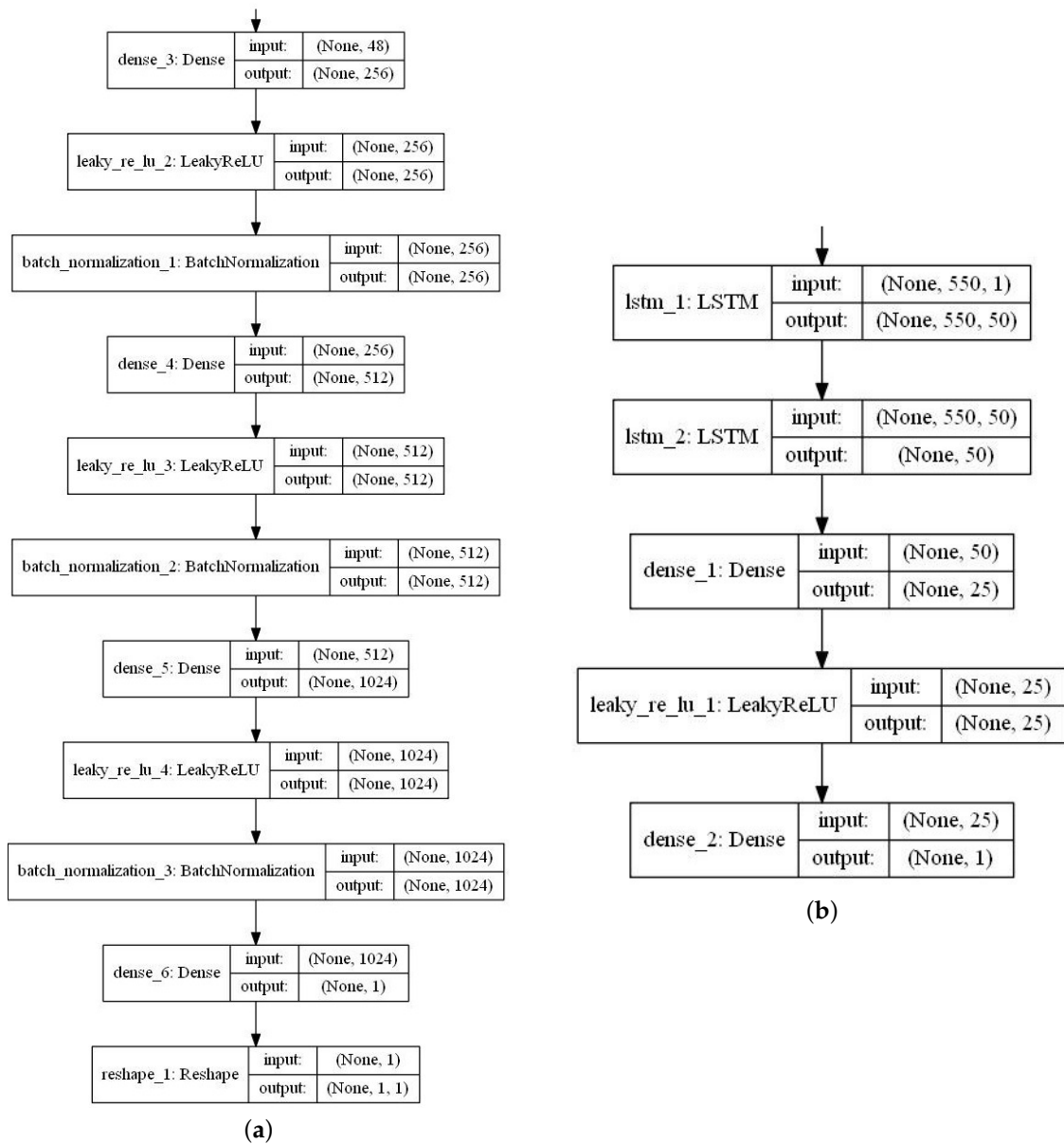


Figure 13. GAN structure for time series forecasting task (a): Generator (b): discriminator.

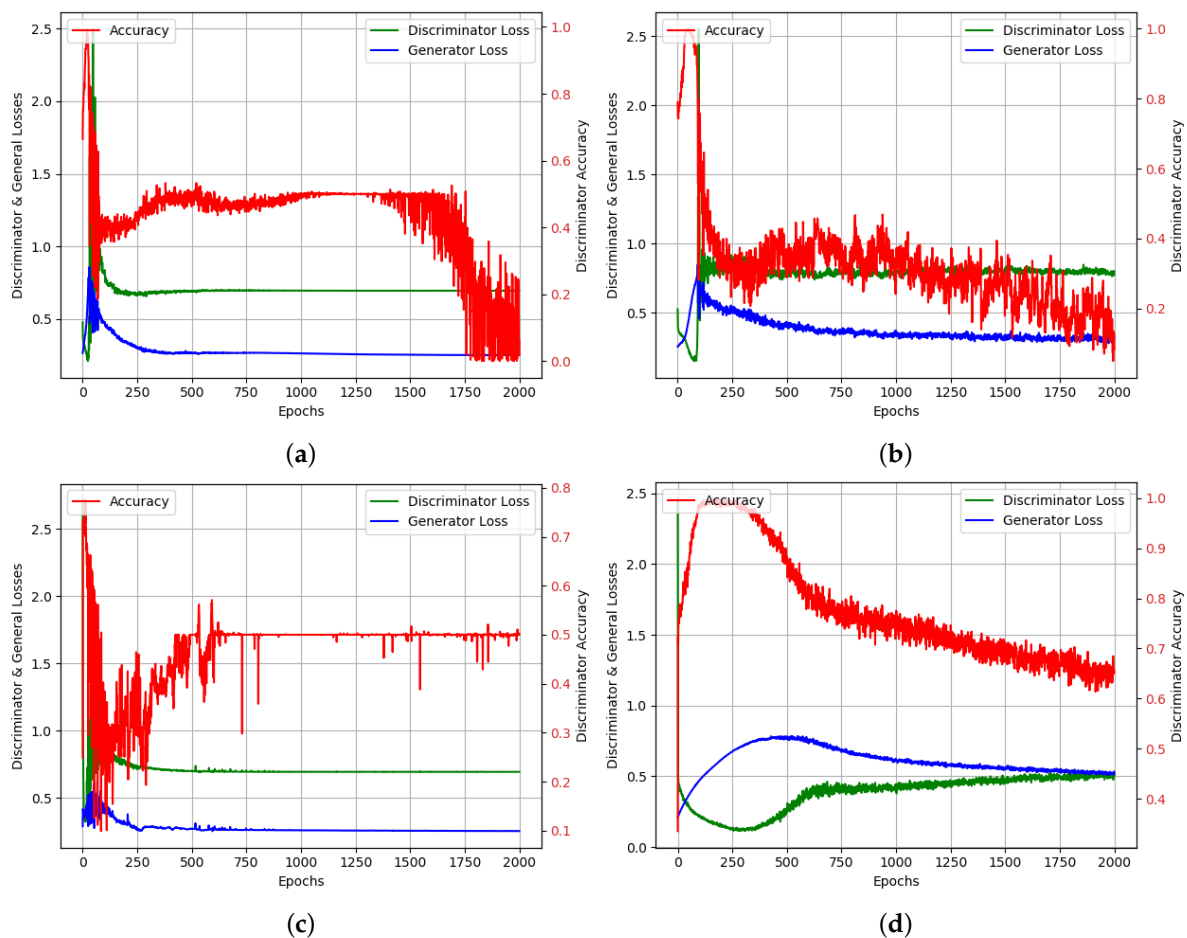


Figure 14. GAN test results for time series forecasting task loss and accuracy for Discriminator and Generator, (a) 'adadelta', (b) 'adam', (c) 'rmsprop', (d) 'sgd'.

4. Discussions

A challenge faced by LSTM is that it is unable to parallel. RNNs once became the dominant network architecture for translation and language modeling. However, one word at a time makes RNNs unable to parallel, so performance improvement became unavailable. This sequential nature also makes it hard to take advantage of fast computing devices such as TPU and GPU. Simply stacking the FCN, CNN and RNN or tweaking their parameters to get better performance for sequence were not enough and sometimes even failed. New architectures such as attention-based models (TCN and transformer) were then invented to increase performance, extract better features from data, generalization capability and reduce parameters.

TCN models with attention mechanisms have achieved remarkable successes, which makes us rethink RNN models' situation in sequence modeling. Time complexity was analyzed to explain the experiment results why TCN outperformed others in terms of efficiency. In sequence models, the hidden state matrix is of size d^2 where d is the dimension of hidden state and n is the length of the input sentence. The cost of computing an input sentence of length n equals $n \times d^2$. The complexity of a sequence model where d and n are equal to 1000 and 50, respectively, is $50 \times 1000 \times 1000$. For models based on attention mechanisms, the complexity is $n^2 \times d$. This is $50 \times 50 \times 1000$. This is the key reason that attention-based models are superior to the sequence model family (RNNs / LSTMs)—the n^2 can be learned faster than the d^2 matrix- thus no need to sequentially back-propagate the errors through time. The experiments also confirmed the conclusions that TCN model has greatly improved the efficiencies. Attention models were fast growing field and many improvement solutions were put forward such as Sandwich transformer [44], Universal transformer [47], and the Residual shuffle

exchange network [48] which requires less parameter compared to other models for the same task. Applying state of the art language models such as BERT, ELMO, GPT, etc. to the time series forecasting and classification areas worth researching.

The unstable training processes and lack of evaluation metrics hinder further development of GAN. Many academic papers on how to enable the stable training process were published, however the problem still needs further research. The evaluation metrics are still scarce (FID and IS dedicated for images quality) and lack of objectivity (judged by humans). Furthermore the majority of influential GAN papers were devoted their GAN models in the specific area of images, several for video generation, however the metrics for generation and forecasting of times series with GAN are much less (most cases visually, or RMSE metrics).

5. Conclusions

We proposed LSTM with autoencoder and attention, TCN with attention and GAN model to time series forecasting and classification, and all of these models accomplished the tasks. The performances of models using five different optimizers with the public datasets of Kaggle web traffic, VPN, ECG and the stress tests were conducted. For the time series classification task, the TCN model outperformed the classical algorithms such as random forest, gradient boosting and extra trees and bagging, and the GAN achieved comparable performances with statistical models such as ARIMA. We proposed the Gaussian sliding window weights to the attention mechanisms which showed reducing the training time greatly at least 80% while remained the same accuracy. “Adam” performed best among all five optimizers most of the time and the discretionary chosen parameters could result in the failure of convergence, especially for GAN models. The grid search method was employed in the experiments to the parameters tweaking, which helped the models’ reaching convergence and performed better.

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

Funding: This research was funded by Institute of Computer Application, China Academy of Engineering Physics(CAEP): SJ2019A05.

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

Abbreviations

The following abbreviations are used in this manuscript:

MLP	Multiple Layer Perceptron
DNN	Deep Neural Network
TCN	Temporal Convolutional Networks
LSTM	Long Short-Term Memory
GAN	Generative Adversarial Network
RNN	Recurrent Neural Network
NLP	Natural Language Processing
ARIMA	Auto Regressive Integrated Moving Average
ETS	Exponential Smoothing
CNN	Covolutional Neural Network
ECG	Electrocardiogram
VPN	Virtual Private Network
FCN	Fully Connected Network
SSE	Sum of Squared Error
RMSE	Root Mean Squared Error
RF	Random Forest

GB	Gradient Boosting
ET	Extra Tree
BA	BAGging method
BS	Brier Score
RF	Generative Adversarial Network

References

1. Kleinbaum, D.G.; Dietz, K.; Gail, M.; Klein, M.; Klein, M. *Logistic Regression*; Springer: Berlin/Heidelberg, Germany, 2002.
2. Rish, I. An empirical study of the naive Bayes classifier. In Proceedings of the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Seattle, WA, USA, 4–6 August 2001; Volume 3, pp. 41–46.
3. Hearst, M.A.; Dumais, S.T.; Osuna, E.; Platt, J.; Scholkopf, B. Support vector machines. *IEEE Intell. Syst. Their Appl.* **1998**, *13*, 18–28. [\[CrossRef\]](#)
4. Peterson, L.E. K-nearest neighbor. *Scholarpedia* **2009**, *4*, 1883. [\[CrossRef\]](#)
5. Adam, S.P.; Alexandropoulos, S.A.N.; Pardalos, P.M.; Vrahatis, M.N. No free lunch theorem: A review. In *Approximation and Optimization*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 57–82.
6. Armstrong, J.S.; Green, K.C.; Graefe, A. Golden rule of forecasting: Be conservative. *J. Bus. Res.* **2015**, *68*, 1717–1731. [\[CrossRef\]](#)
7. Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.* **2019**, *33*, 917–963, doi:10.1007/s10618-019-00619-1. [\[CrossRef\]](#)
8. Wang, Z.; Yan, W.; Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In Proceedings of the International Joint Conference on Neural Networks, Anchorage, AK, USA, 14–19 May 2017; pp. 1578–1585.
9. Fawaz, H.I.; Lucas, B.; Forestier, G.; Pelletier, C.; Schmidt, D.F.; Weber, J.; Webb, G.I.; Idoumghar, L.; Muller, P.A.; Petitjean, F. InceptionTime: Finding AlexNet for Time Series Classification *Data Min. Knowl. Discov.* **2020**, *34*, 1936–1962. [\[CrossRef\]](#)
10. Time Series Classification Repository. Available online: <http://timeseriesclassification.com/index.php> (accessed on 15 December 2020).
11. Zhang, G. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* **2003**, *50*, 159–175. [\[CrossRef\]](#)
12. Koehler, J.; Kuenzer, C. Forecasting Spatio-Temporal Dynamics on the Land Surface Using Earth Observation Data—A Review. *Remote Sens.* **2020**, *12*, 3513. [\[CrossRef\]](#)
13. Ghaderpour, E.; Vujadinovic, T. The Potential of the Least-Squares Spectral and Cross-Wavelet Analyses for Near-Real-Time Disturbance Detection within Unequally Spaced Satellite Image Time Series. *Remote Sens.* **2020**, *12*, 2446. [\[CrossRef\]](#)
14. Cho, K.; Courville, A.; Bengio, Y. Describing Multimedia Content Using Attention-Based Encoder-Decoder Networks. *IEEE Trans. Multimed.* **2015**, *17*, 1875–1886. [\[CrossRef\]](#)
15. Bahdanau, D.; Cho, K.H.; Bengio, Y. Neural machine translation by jointly learning to align and translate. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015; pp. 1–15.
16. Cheng, J.; Dong, L.; Lapata, M. Long short-term memory-networks for machine reading. In Proceedings of the EMNLP 2016—Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016; pp. 551–561.
17. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing—Proceedings, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
18. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5999–6009.

19. Ju, Y.; Sun, G.; Chen, Q.; Zhang, M.; Zhu, H.; Rehman, M.U. A Model Combining Convolutional Neural Network and LightGBM Algorithm for Ultra-Short-Term Wind Power Forecasting. *IEEE Access* **2019**, *7*, 28309–28318. [[CrossRef](#)]
20. Berardi, V.L.; Zhang, G.P. An empirical investigation of bias and variance in time series forecasting: Modeling considerations and error evaluation. *IEEE Trans. Neural Networks* **2003**, *14*, 668–679. [[CrossRef](#)] [[PubMed](#)]
21. Fulcher, B.D.; Jones, N.S. Highly Comparative Feature-Based Time-Series Classification. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 3026–3037. [[CrossRef](#)]
22. Mei, J.; Liu, M.; Wang, Y.; Gao, H. Learning a Mahalanobis Distance-Based Dynamic Time Warping Measure for Multivariate Time Series Classification. *IEEE Trans. Cybern.* **2016**, *46*, 1363–1374. [[CrossRef](#)] [[PubMed](#)]
23. Amin, S.U.; Alsulaiman, M.; Muhammad, G.; Bencherif, M.A.; Hossain, M.S. Multilevel Weighted Feature Fusion Using Convolutional Neural Networks for EEG Motor Imagery Classification. *IEEE Access* **2019**, *7*, 18940–18950. [[CrossRef](#)]
24. Mori, U.; Mendiburu, A.; Lozano, J.A. Similarity Measure Selection for Clustering Time Series Databases. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 181–195. [[CrossRef](#)]
25. Liu, C.; Hsiao, W.; Tu, Y. Time Series Classification With Multivariate Convolutional Neural Network. *IEEE Trans. Ind. Electron.* **2019**, *66*, 4788–4797. [[CrossRef](#)]
26. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* **2014**, *3*, 2672–2680, doi:10.3156/jsoft.29.5_177_2. [[CrossRef](#)]
27. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein generative adversarial networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 214–223.
28. Yu, L.; Zhang, W.; Wang, J.; Yu, Y. SeqGAN: Sequence generative adversarial nets with policy gradient. In Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017), San Francisco, CA, USA, 4–10 February 2017; pp. 2852–2858.
29. Gong, X.; Chang, S.; Jiang, Y.; Wang, Z. Autogan: Neural architecture search for generative adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Korea, 27 October–2 November 2019; pp. 3224–3234.
30. Zhang, H.; Xu, T.; Li, H.; Zhang, S.; Wang, X.; Huang, X.; Metaxas, D.N. StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *41*, 1947–1962, doi:10.1109/TPAMI.2018.2856256. [[CrossRef](#)]
31. Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. Improved training of wasserstein GANs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5768–5778.
32. Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X. Improved techniques for training GANs. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 2234–2242.
33. Che, T.; Li, Y.; Jacob, A.P.; Bengio, Y.; Li, W. Mode regularized generative adversarial networks. *arXiv* **2016**, arXiv:1612.02136.
34. Mescheder, L.; Geiger, A.; Nowozin, S. Which training methods for GANs do actually converge? In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; pp. 5589–5626.
35. Arora, S.; Ge, R.; Liang, Y.; Ma, T.; Zhang, Y. Generalization and equilibrium in generative adversarial nets (GANs). In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017; pp. 322–349.
36. Che, T.; Li, Y.; Zhang, R.; Hjelm, R.D.; Li, W.; Song, Y.; Bengio, Y. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv* **2017**, arXiv:1702.07983.
37. Kusner, M.J.; Hernández-Lobato, J.M. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv* **2016**, arXiv:1611.04051.
38. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
39. Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [[CrossRef](#)]
40. Kaggle Web Traffic Competition. Available online: <https://www.kaggle.com/c/web-traffic-time-series-forecasting> (accessed on 15 December 2020).
41. VPN-nonVPN Dataset from Canadian Institute of Cyber-Security. Available online: <https://www.unb.ca/cic/datasets/vpn.html> (accessed on 15 December 2020).

42. Le, Q.V.; Jaitly, N.; Hinton, G.E. A simple way to initialize recurrent networks of rectified linear units. *arXiv* **2015**, arXiv:1504.00941.
43. Zhang, S.; Wu, Y.; Che, T.; Lin, Z.; Memisevic, R.; Salakhutdinov, R.R.; Bengio, Y. Architectural complexity measures of recurrent neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 1822–1830.
44. Press, O.; Smith, N.A.; Levy, O. Improving Transformer Models by Reordering their Sublayers. *arXiv* **2019**, arXiv:1911.03864.
45. Ba, J.L.; Kiros, J.R.; Hinton, G.E. Layer normalization. *arXiv* **2016**, arXiv:1607.06450.
46. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
47. Dehghani, M.; Gouws, S.; Vinyals, O.; Uszkoreit, J.; Kaiser, Ł. Universal transformers. *arXiv* **2018**, arXiv:1807.03819.
48. Draguns, A.; Ozoliņš, E.; Šostaks, A.; Apinis, M.; Freivalds, K. Residual Shuffle-Exchange Networks for Fast Processing of Long Sequences. *arXiv* **2020**, arXiv:2004.04662.

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).