mathematics

MDPI

Article

Web Traffic Time Series Forecasting Using LSTM Neural Networks with Distributed Asynchronous Training

Roberto Casado-Vara [1,*], Angel Martin del Rey [2], Daniel Pérez-Palau [3] and Luis de-la-Fuente-Valentín [3] and Juan M. Corchado [1]

[1] BISITE Research Group, University of Salamanca, 37008 Salamanca, Spain; corchado@usal.es
[2] Department of Applied Mathematics, Institute of Fundamental Physics and Mathematics, University of Salamanca, 37008 Salamanca, Spain; delrey@usal.es
[3] Escuela Superior de Ingeniería y Tecnología, Universidad Internacional de La Rioja, Av. La Paz 137, 26006 Logroño, Spain; daniel.perez@unir.net (D.P.-P.); luis.delafuente@unir.net (L.d.-l.-F.-V.)
* Correspondence: rober@usal.es

Abstract: Evaluating web traffic on a web server is highly critical for web service providers since, without a proper demand forecast, customers could have lengthy waiting times and abandon that website. However, this is a challenging task since it requires making reliable predictions based on the arbitrary nature of human behavior. We introduce an architecture that collects source data and in a supervised way performs the forecasting of the time series of the page views. Based on the Wikipedia page views dataset proposed in a competition by Kaggle in 2017, we created an updated version of it for the years 2018–2020. This dataset is processed and the features and hidden patterns in data are obtained for later designing an advanced version of a recurrent neural network called Long Short-Term Memory. This AI model is distributed training, according to the paradigm called data parallelism and using the Downpour training strategy. Predictions made for the seven dominant languages in the dataset are accurate with loss function and measurement error in reasonable ranges. Despite the fact that the analyzed time series have fairly bad patterns of seasonality and trend, the predictions have been quite good, evidencing that an analysis of the hidden patterns and the features extraction before the design of the AI model enhances the model accuracy. In addition, the improvement of the accuracy of the model with the distributed training is remarkable. Since the task of predicting web traffic in as precise quantities as possible requires large datasets, we designed a forecasting system to be accurate despite having limited data in the dataset. We tested the proposed model on the new Wikipedia page views dataset we created and obtained a highly accurate prediction; actually, the mean absolute error of predictions regarding the original one on average is below 30. This represents a significant step forward in the field of time series prediction for web traffic forecasting.

Keywords: web traffic forecast; time series forecast; LSTM; parameter averaging; Downpour strategy; pattern extraction

Citation: Casado-Vara, R.; Martin del Rey, A.; Pérez-Palau, D.; de-la-Fuente-Valentín, L.; Corchado, J.M. Web Traffic Time Series Forecasting Using LSTM Neural Networks with Distributed Asynchronous Training. *Mathematics* **2021**, *9*, 421. https://doi.org/10.3390/math9040421

Academic Editor: Zeev Volkovich, Oleg Granichin, Dvora Toledano-Kitai and Paolo Crippa

Received: 30 December 2020
Accepted: 17 February 2021
Published: 21 February 2021

1. Introduction

The majority of web resources provide customers with transactions of various kinds. As an example, the network communication which is forwarded at the user's request creates a phone conversation, the game provider addresses the player's request, the web application handles the HTTP GET request as a consequence of the website's response, media applications spread content according to the user's demand, etc. The request time will strongly influence on the quality perceived by the end-user. Due to the high response time, numerous platforms have lost their users. However, the response time is the time between when the application receives the request and the time when the reply is sent. This cannot be removed. It is mainly affected by the following sources: cache, disk, geometry data, network bandwidth, network congestion, change in request arrival data, request

queuing, unexpected usage and flow patterns. In the case of web services, the response time is too long to be assumed by customers, and developers have been able to diagnose situations where the response time is too long. Better service is often associated with higher customer satisfaction [1].

This problem, web congestion, can be mathematically modeled with the time series in which the values of the series are the dates with the required granularity (usually daily) and the number of page views that the web server has had. Analyzing these time series will provide information about web traffic features such as trend and seasonality for the daily dataset. Time series forecasting (TSF) is an important field of application and covers many different fields, ranging from economic trend indicators and weather forecasting to demand driven power plant construction. This topic has a strong research precedent and has received the attention of several scientists throughout the world [2,3]. Not only are there numerous academic papers on this topic but also many competitions. These activities encouraged the developments of forecasting methods for several real world time series. In this study, we checked the Kaggle dataset web traffic forecasting competition to analyze the visit pages in the Wikipedia categorized by language [4–6]. Once this was done, we created a new current version of the Wikipedia web traffic dataset with webpages and their page views on the most visited pages for each language from 1 January 2018 to 31 December 2020.

The purpose of this research is to design and develop a deep learning-based platform to analyze and forecast online web traffic of a considered web-server. Emphasis is placed in feature extraction and pattern detection in the analysis, which allows the design of a Long Short-Term Memory (LSTM) to forecast the flow of page views on websites in the short and medium term. The hypothesis of this research was the possibility of enhancing the operation of the web servers by identifying the demand for views of the web pages. Time-series being an important concept in statistics and machine learning is often less explored by data scientist enthusiasts including us. To change the winds, we decided to work on one of the most burning time series problem of today, "predicting web traffic". Actually, we strongly consider that this web traffic forecasting approach will help website servers to effectively address disruptions. Our implemented technique can be widely applied to various fields such as financial markets, weather forecasting, audio and video processing. Furthermore, figuring out the traffic patterns of a website will increase customer satisfaction and open up new potential business opportunities. Although time series analysis and forecasting have been well researched, the number of research papers on applying LSTM on real-time real data to this area is still limited. We intuitively asked if this extraction of time series patterns such as seasonality or trend would improve the design of TSF methods. However, according to our research, we found that the number of papers on this and similar topics is far less fewer those dealing with images, audio, etc. On the other hand, the web traffic data could also be modeled as time series, so we decided to investigate the possibilities of applying pattern extraction to the design of a LSTM for TSF.

To provide an optimal solution to this problem, an architecture was designed to collect data through a scrapper that gathers the pages' view data. The architecture performs the pre-processing of these data, discovering the features and finding the patterns embedded in these data. Finally, applying all the knowledge discovered in the analysis step, a LSTM was designed to fit the problem of forecasting the web traffic of the chosen page. To validate the architecture and the artificial intelligence model based on deep learning, the wikipedia web traffic dataset competition suggested by Kaggle in 2017 was retrofitted with data from 2018 to 2020. In this case, it was done for Wikipedia website, however with minimum changes it could be used on other websites or computer networks as well.

We made the forecast of each of the Wikipedia page views difference by language where we applied pattern detection techniques to design a distributed architecture with several LSTM, which were trained asynchronously using the strategy called Downpour, which make a TSF with a relatively low error for each of the languages that have been taken into consideration in the work.

The paper's contributions are summarized as follows:

- Effectively captured seasonality patterns and long-term trends for supporting the design of the LSTM
- Architecture for supervised web traffic forecasting using advanced AI models
- Training systems with a parameter server that enhance the asynchronously LSTM train using the Downpour strategy
- Good quality forecasts in long time horizon achieved in the seven dominant languages of the dataset

The remainder of the paper is divided as follows. Section 2 provides the state of the art of Artificial Intelligence (AI) models for web traffic forecasting. The proposed distributed architecture and the LSTM model are presented in Section 3. Results and discussion are placed in Section 4. Finally, the conclusions and the future work are shown in Section 5.

2. Related Work

Over the years, the forecasting area has been affected by the fact that experts have dismissed neural networks (NNs) as being non-competitive, and NN enthusiasts have introduced several new and sophisticated NN architectures, mostly without strong empirical evaluations as compared to simplified univariate statistical methods. Especially, this notion was supported by many time series prediction contests such as the M3, NN3 and NN5 competitions [7–9]. Subsequently, NNs have been classified as not convenient for forecasting. There are several possible reasons for the weak performance of NNs in the past, one of them being that the individual time series involved were often simply too short to be modeled through sophisticated methods. Alternatively, the time series features may have evolved over time, so that even longer time series may not contain enough relevant data to fit a complicated model [10,11]. Thus, for modeling series through complex methods, it is critical that they are of appropriate length, as well as produced from a relatively robust system. Furthermore, NNs are widely criticized for their black box nature. Thus, forecasting experts have historically preferred to use easier statistics methods [12].

However, at present, we are experiencing a big data world. Companies have collected a great deal of data over the years, which holds important insight into their business patterns. Big data in the time series context do not always mean that an individual time series carries significant amounts of detail. Rather, they usually indicate that there are many associated time series in a given field. Within that context, univariate prediction methods that consider individual time series separately can fail to provide reliable predictions. They become unavailable in the context of big data where a single model could learn simultaneously from many similar time-series. Moreover, even more sophisticated models such as the NNs profit as much as possible from the access to massive quantities of data [13,14]. It is in this new field of growth of scientific interest in the NN that the Recurrent Neural Networks (RNN) come into play. With this new type of neural networks specialized in the sequence prediction problem, results never seen before in the field of language and time series analysis begin to be achieved [15]. However, the RNNs have serious memory problems, which were solved with the inclusion in the research world of the LSTM. This new type of RNN has a new internal memory (cell state) in addition to the usual hidden state of the RNN. This makes it easier for the training of LSTMs to avoid the problems of vanishing or exploding gradients [16].

Since time series have components of seasonality, LSTM can be used in a predictive context. For example, if a given monthly time series has a yearly seasonality, the prediction of the value for the immediate next month benefits more from the value of the same exact month of the previous year. Suilin et al. did an outstanding work for the Kaggle challenge of Wikipedia's web traffic forecast encompassing this idea [17]. Although this dataset has been widely used for the prediction of time series related to web traffic, it has not been deepened in the design of the LSTM with few data, since it is assumed by the researchers that other models such as ARIMA are more efficient in these cases. Other research work at

TSF has led to more complex schemes that do not take into account the seasonal component for prediction [18]. Qin et al. proposed an RNN for multivariate forecasting problems. In this model, different weights are assigned to the different driving series according to their importance in contributing to the forecast at each time stage. This model was widely validated by the authors comparing it with ARIMA, NARX RNN, Encoder Decoder, Attention RNN, Input Attention RNN and the Dual Stage Attention RNN [19].

More recent work has delved into the model proposed by Qin et al. In [20], the authors claimed that their model can handle the inherent characteristics of the spatial-temporal series. In recent years, the tendency of researchers is to stack RNN or stack LSTM to achieve the required result in FTS problems. However, we found a gap in the literature with FTS prediction models in time series with limited data [21]. Due to the gap detected in the state of the art in the prediction of time series with little data, we proposed a supervised architecture based on LSTM that is trained through distributed data parallelism and following the Downpour strategy.

3. Proposed Model

In this section, we describe the proposed architecture for network traffic forecasting. This architecture is modularly designed, distributed and scalable such that with minimal modifications needed it can be easily adapted and used for network traffic predictions, regardless of whether it is a closed computer network or a website. This architecture follows the design pattern from bottom to top, as can be seen in Figure 1. It is mainly divided into three major layers: data extraction, its transformation by extracting its features and data loading as a time series in the deep learning layer to make predictions. In this study, we focused on the prediction of the Wikipedia web traffic stream has worldwide on some selected topics by the authors. For designing and testing our data analysis and forecasting model of the network traffic page views, we used the dataset 'Wikipedia web traffic' used in other recent works [22,23]. even though the Wikipedia web traffic helps us in the design of our model, we created a new version of this dataset since we developed our own wikipedia scrapper in order to build our own Wikipedia's top 1000 page views dataset from all the language available in the List of ISO 639 − 1 codes.

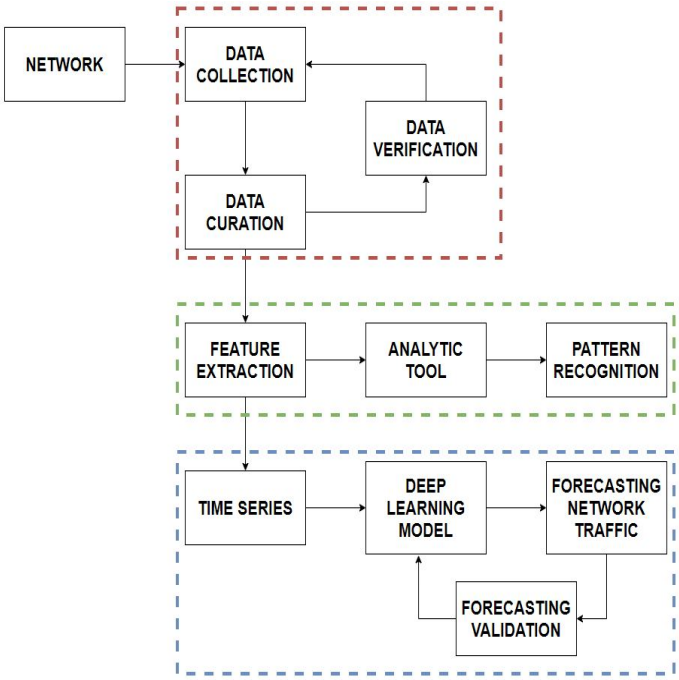


Figure 1. Proposed architecture for supervised web traffic forecasting.

3.1. Data Collection, Cleansing and Curation

The first step is to develop a scrapper to collect network traffic data from the network APIs. Figure 2 shows the diagram of this scrapper. These data are stored in CSV format by the crawler, which makes it available for preparing data within the architecture. The crawler takes information from wikipedia's API, requesting for each of the projects, such as en.wikipedia.org, ja.wikipedia.org, de.wikipedia.org, fr.wikipedia.org, zh.wikipedia.org, ru.wikipedia.org and es.wikipedia.org, the top 1000 websites during each month of the year 2020. Then, for each project, a list of popular websites is created, removing duplicate ones. Finally, for each of the recovered websites, the daily number of visits from 2015 to 2020 is extracted, both inclusive, taking into account all agents (human and bot accesses) and types of access (web, mobile or desktop). For enhancing the process, data from each project difference by language are obtained in a separate thread, which are backed up periodically to avoid loss of information in case of error, and then combined with all the data at the end of the process. It should be pointed out that this improvement does not mean a substantial increase in the consumption of computer resources, since most of the time the threads are waiting for the responses to the HTTP requests they perform.

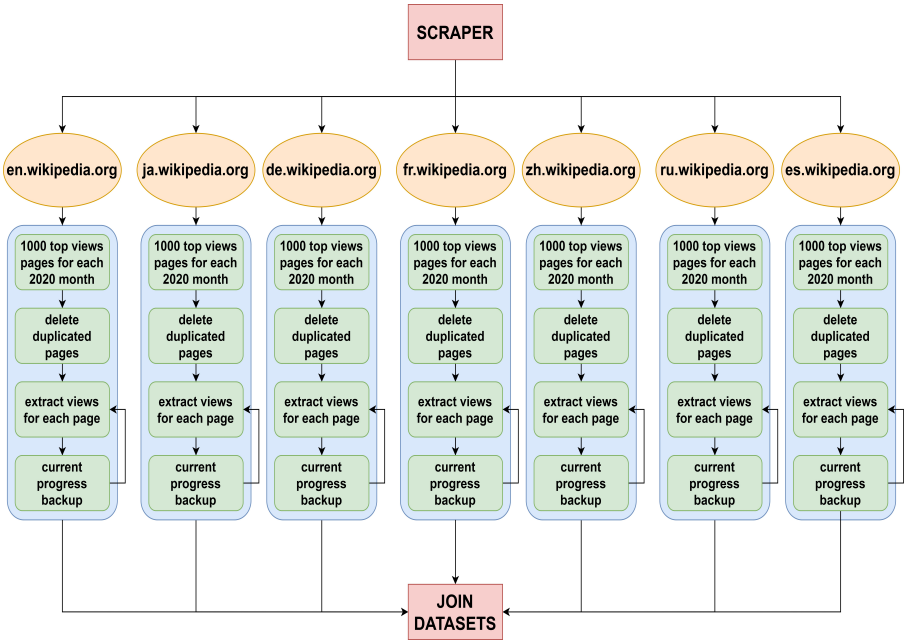


Figure 2. Architecture for dataset generation via concurrent scraping techniques. In the architecture, the scraper creates one thread per Wikipedia's project. Then, in each thread, the most relevant websites for the targeted project are extracted, and subsequently the views for each website (in the selected time window) are extracted. It is worth highlighting that, after downloading the views for each article, the current progress of data is stored to ensure the data are saved in case of failure. Finally, when all the views data are extracted, they are merged into a single dataset.

In addition, in this layer of the architecture, we developed a Python script to pre-process these data, removing the wrong values or filling the empty positions with 0 s. This process is known as data curation and involves cleaning data and providing the appropriate structure to submit them to the next layers of the architecture.

3.2. Feature Extraction and Pattern Recognition

As a previous step to the design of an AI model, it is strongly suggested to perform an analysis of data for an in-depth understanding, extracting the features and identifying the hidden patterns. This may help to define the core layers of the AI model such that its data and outcomes are significantly better than a trial-and-error designed neural network. This section explains the methods of analysis and pattern recognition that have been performed.

For this purpose, the Python data analysis libraries such as numpy, pandas and seaborn were used.

3.2.1. Top Page Views in Different Languages

During the first part of the exploratory analysis, the page views were shown in an aggregated form, separated by language. Figure 3 shows the form of the time series as well as many other data such as some peaks that indicate a large number of visits even in several languages. The most viewed pages are the main portal pages of Wikipedia, and each page has its unique trend features. In addition, there are some weird spikes as well. Thus, we here asked ourselves if the traffic is influenced by page language, which is clearly shown in Figure 3.

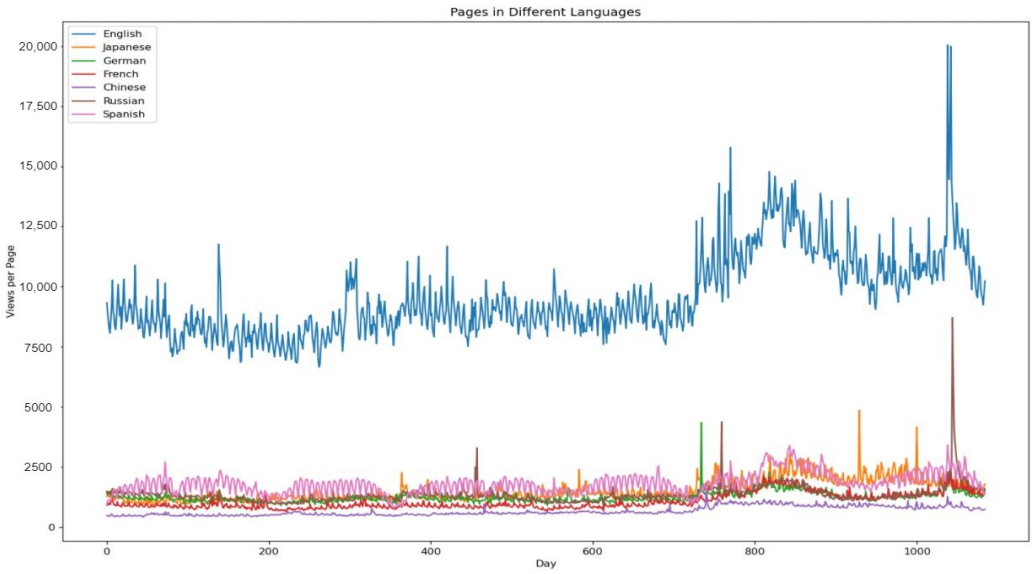


Figure 3. Wikipedia page views web traffic time series difference by languages.

3.2.2. Periodic Structure of Page Views Time Series

Once we have the time series with the processed data, we usually analyze its periodical structure. This analysis is usually done by the Fast Fourier Transform Python function. In this analysis, we move from the time domain to the frequency domain in which we can see the harmonics of the time series.

In Figure 4, there are clear peaks at $1/7$, $2/7$ and $3/7$. They are likely to be the weekly trends as we have seven days per week. In addition, there are trends in longer terms (smaller frequency) depend on the language.

3.2.3. Distribution of Web Traffic by Languages

A probability distribution is a statistical function that describes all the possible values and likelihoods that a random variable can take within a given range. Perhaps the most common probability distribution is the normal distribution, or "bell curve", although several distributions exist that are commonly used. Typically, the data generating process of some phenomenon will dictate its probability distribution. This process is called the probability density function. This analysis is useful to find the ranges of congestion that the web server will have to handle for stress test designs and its daily work. Figure 5 shows the distribution function difference by language.

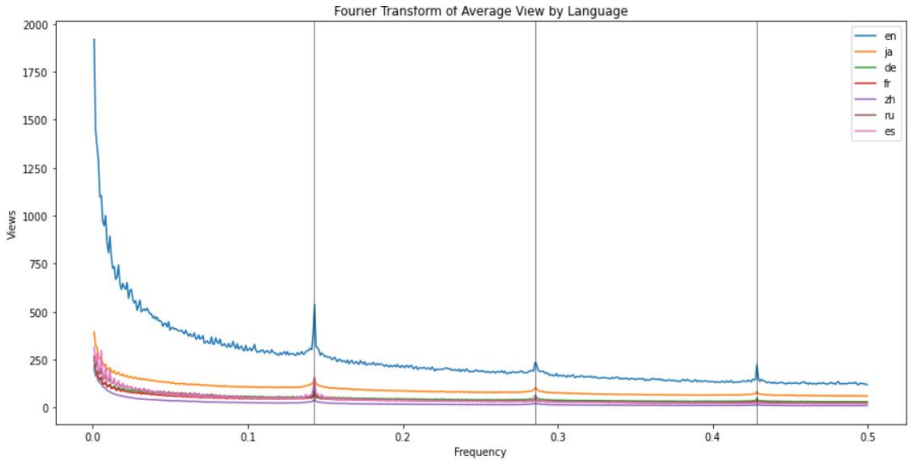


Figure 4. FFT applied to Wikipedia page views web traffic time series difference by languages.

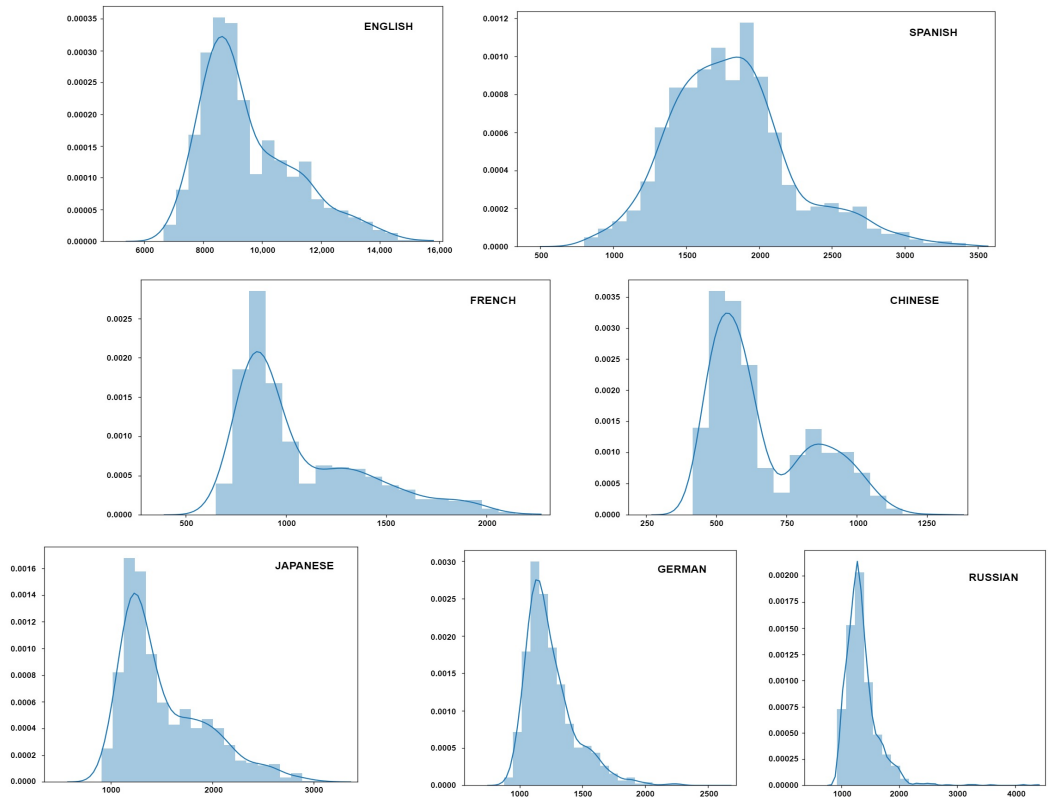


Figure 5. Density distribution of the page views values of the updated Wikipedia dataset difference by languages.

3.2.4. Testing for Seasonality

For testing the seasonality, we use the following two functions: ACF is an autocorrelation (full) function which provides us with autocorrelation results of any series with its delayed values. It describes, quite simply, exactly how closely the present series value is related to its previous values. A time series has components such as trend and seasonality. The ACF addresses all of these factors while searching for correlations, which makes it a "fully autocorrelated graph".PACF is a partial autocorrelation function. Essentially, rather than locating present correlations with time delays similar to ACF, the residuals are correlated with the next time delay value, thus it is "partial" and not "complete" since we remove the fluctuations previously found until the next correlation is found. Consequently, if there is some hidden knowledge in the residue that can be modeled by the next gap, we could obtain a strong correlation and we will maintain that next gap as a feature while

modeling. Keep in mind that while modeling we do not want to keep too many features that are correlated since that can create multicollinearity problems. Therefore, we need to maintain only the outstanding features. The seasonality analysis is shown in Figure 6.

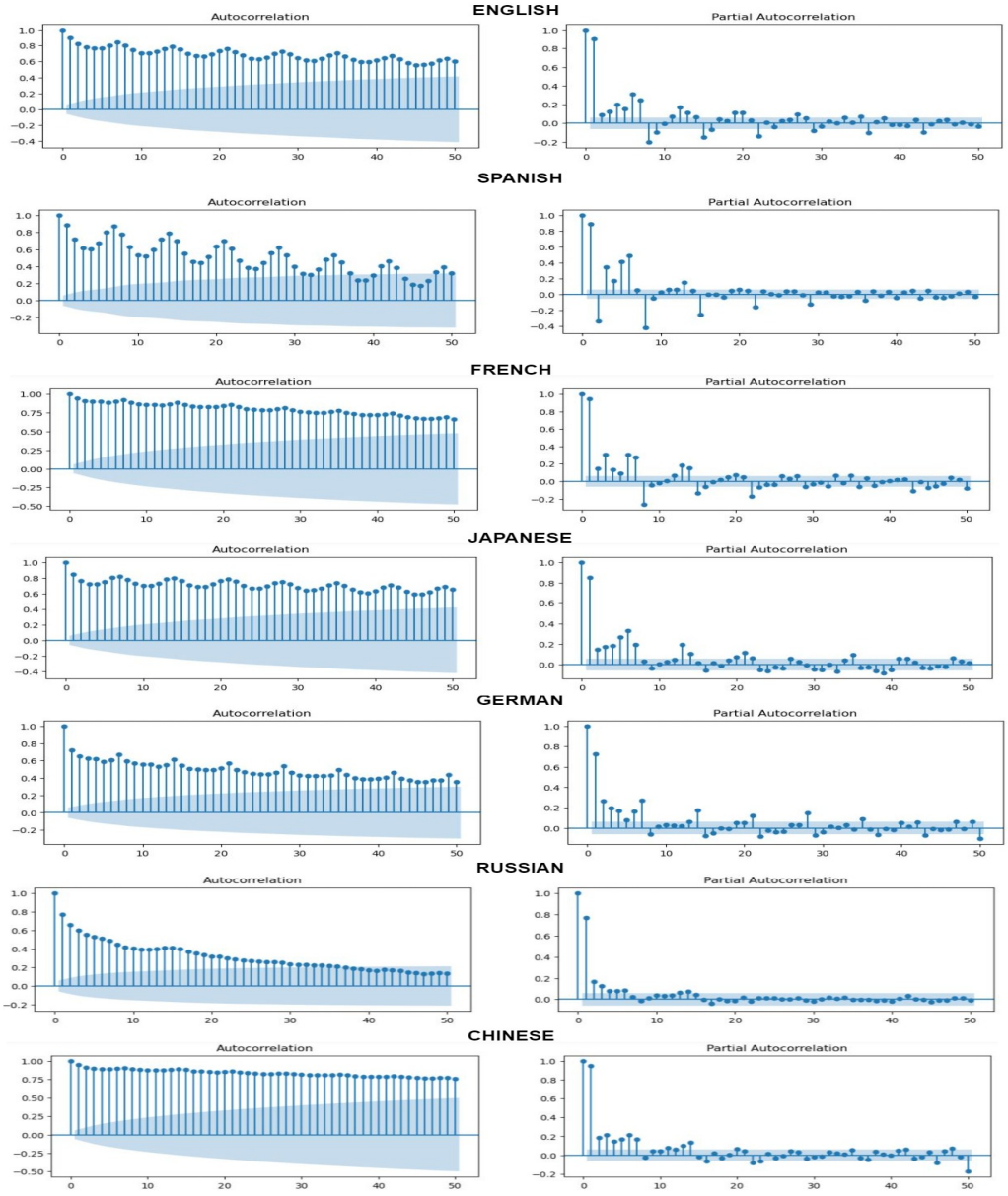


Figure 6. Auto-correlation (**left**) and partial auto-correlation (**right**) of the time series of the Wikipedia dataset difference by languages.

3.3. Deep Learning for Network Traffic Forecast

Since we are working with time series, we have to design an AI model for forecasting the new expected values. To accomplish this task, the following possible options are available: RNN, LSTM, GRU and TCN. Recurrent architectures are designed to process input sequences quite efficiently. Since the memory problems of RNNs and the problems with vanishing or exploding gradient computations were demonstrated, LSTMs and Gated Recurrent Units (GRU) became highly popular among researchers. For this reason, at the beginning of our research, we ruled out using an RNN model in our model for forecasting web traffic on Internet servers. LSTM iterates a sequence element by element to learn which sequence of elements leads to which type of result. To control the operation of LSTMs, they have two internal states: cell state (which transmits intermediate results

from one iteration step to another, finally presenting the final result) and hidden state (which provides a snapshot of the result of the current iteration step). GRUs are a model of recurrent networks that were designed as a lighter and faster alternative to LSTMs. To achieve this, the cell state and the hidden state are merged into a single hidden state for each iteration. However, GRU could not achieve the same performance as LSTMs unless they are enhanced with residual connections, which would mean using the ResNet neural network. For this reason, at the beginning of our research, we decided not to use the GRU, since it implied the need to use the ResNet to achieve good results and that meant having to have greater resources to achieve similar efficiency to LSTMs.

Temporal convolutional networks (TCNs) are a new type of neural network for working with sequences. TCN employs techniques such as multiple layers of dilated convolutions and padding of input sequences to handle different sequence lengths and detect dependencies between elements that are not next to each other, but are located at different places in a sequence. TCN researchers have shown that their implementation is capable of outperforming standard RNNs in the analysis of long time series. However, Bai et al. [24] showed the data storage problems of TCNs during training and validation. LSTMs only maintain their two internal states (cell state and hidden state) and receive the input to generate the prediction. In contrast, TCNs need to take the raw sequence up to the effective length of the history, thus requiring larger memory during evaluation. For practical purposes, this means that TCNs are very effective for extremely long time series, while they perform not as well on short time series as they do on long time series. In our research, as previously outlined, the time series in our dataset have a length of 1000 days, which implies that they are quite short. Considering the above reasons, we decided that our time series forecasting model should be based on LSTM. In this subsection, we detail how a LSTM works and then design and develop our LSTM model.

3.3.1. Long Short-Term Memory: An Overview

The LSTM emerged as an architecture aimed at solving the "memory" problems of the RNN vanilla. In practice, RNN present problems to learn relationships with distant time step elements (i.e., not close to the current time step). This causes most of the theoretical potential of RNNs being lost. LSTMs are explicitly designed to try to solve this problem. To do this, they have an internal cell state (ct) as well as the conventional hidden state (ht), which represents a kind of "information highway" over time, as shown in Figure 7.

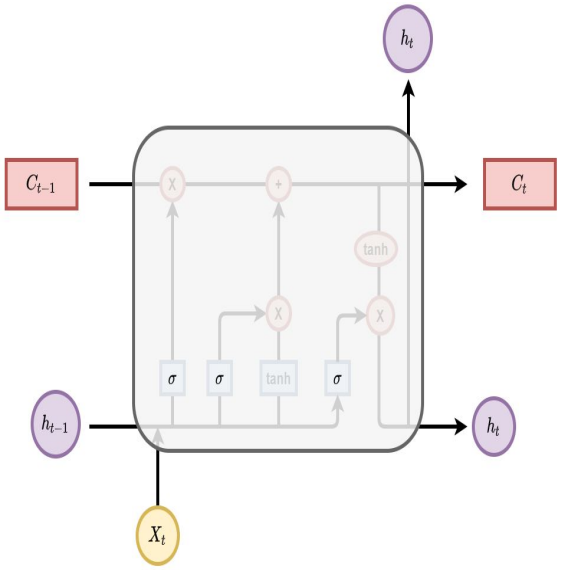


Figure 7. Single LSTM cell with the "information highway" highlighted.

In Figure 8, a diagram of the operation of a LSTM can be seen considering the current state, the previous one and the next state. This kind of RNN is an upgraded version of the

vanilla RNNs, despite the complexity of the LSTM architecture, one can find the differences with RNN in the mathematical formulation which allow us for developing LSTM with just only a few changes with the RNN code. RNN involve the following equation for finding their hidden state in each new iteration of the RNN is shown in Equation (1).

$$h_t = tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \tag{1}$$

where h_t is the current hidden state, W is the parameter matrix with the embedding weights, h_{t-1} is the last hidden state and x_t is the new input of the RNN. On the other hand, LSTMs have by far the most complex equation regarding the present RNN models. They have a hidden state (h_t) as well as their own new hidden state called cell state (c_t). Both hidden states allow the LSTM for having a good operation such as language model, text prediction and time series forecasting. The equation to calculate these two hidden states is shown in Equation (2).

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \tag{2}$$

Looking at the formula of the LSTM, we see that we have many new elements. Instead of computing directly "h", we now get four different vectors called gates, namely i, f, o and g, which are then combined to obtain the cell state c_t and the hidden state h_t. Products that are seen within the vectors in the formula are elemental matrix products. The values of i, f and o are applied after a sigmoid activation function, which means that they have values ranging from 0 to 1. Therefore, they act as gates that retain (values close to 1) or eliminate (values close to 0) information. They are detailed as follows:

- f is called forget gate and, once it is multiplied by the previous cell state (c_{t-1}), it represents how much we have to forget about the values stored in it.
- i is called input gate and it represents how much we have to write in c_t. The values to write in c are given by g, which comes from applying $tanh$ in the a vanilla RNN.
- c_t is obtained as a combination of "how much we remember from the past" $(f \cdot c_{t-1})$ and "how much new information we want to add" $(i \cdot g)$; this relationship can be found in Equation (3).

$$c_t = (f \cdot c_{t-1}) + (i \cdot g) \tag{3}$$

- o is called output gate and reveals how much of our internal c_t state we have to show in the new hidden state h_{t+1}.

The operation of a LSTM is slightly confusing and complex. However, its outstanding performance in practice has been essential to the popularity of the LSTM in the machine learning community.

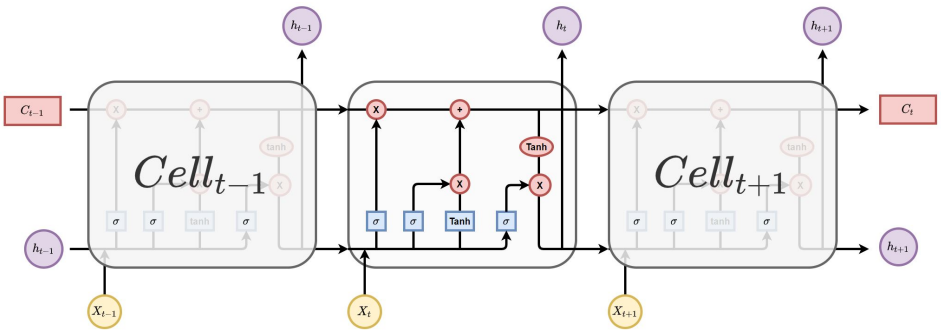


Figure 8. LSMT sequence cells. In this figure, a detailed scheme of a single LSTM cell is shown. In addition, the information flow and the hidden state and cell state flows are highlighted.

3.3.2. Downpour Strategy for Distributed Training of the LSTM Model

Since our dataset for page views of the Wikipedia from 2018 to 2020 has seven dominant languages (Spanish, 'es'; English, 'en'; German, 'de'; French, 'fr'; Russian, 'ru'; Chinese, 'zh'; Japanese, 'jp'), we decided the best way to train our LSTM is through distributed training. We designed a LSTM with the web traffic dataset from Wikipedia for the competition proposed by Kaggle. Once we finished the design of the LSTM, we made a copy of the LSTM on seven virtual machines and another parameter server node within an AWS g3s.xlarge environment. There are two remarkable agents in this architecture:

- Parameter server: It is in charge of maintaining the most current version of the LSTM model. Its main purpose is to receive the hyper-parameters of each of the workers, update them and send them back for retraining. The operation of this parameter server is ruled by Equation (4).

$$W_{i+1} = \frac{1}{7}\sum_{j=1}^{7} W_{i+1,j} \tag{4}$$

- Workers: Virtual machines that have a copy of the LSTM model and train a part of the entire dataset. When they finish each iteration of the training, they send the hyper-parameters to the parameter server and wait for the response with the new ones for training again the LSTM.

The artificial intelligence layer of the architecture was designed such that the LSTM could be trained in an asynchronous way with as many copies as necessary (one for each of the dominant languages in the dataset). Figure 9 shows a diagram of the operation of the Parameter Averaging method.

The operation of the parameter averaging model, which is embedded in our LSTM, to train an instance of the LSTM for the specific problem of the TSF distinguished by each language is summarized in the following steps:

1. The parameters in the workers' LSTM are initialized with a normal distribution of mean 0 and standard deviation 0.1.
2. The parameter server spreads a copy of the current parameters to each worker.
3. Each worker performs training on its subset of data.
4. Each worker sends the new parameters of the model it has obtained by training to the parameter server.
5. The parameter server waits to receive all the parameters from all the workers. Once received, the new current parameters of the model are set as the average of all received parameters.
6. Return to Step 2 and repeat the process for the selected number of epochs.

Parameter averaging has the problem that one has to wait until all the workers finish their iteration to update the parameters and make an update of the model. To solve this problem, we propose to use the distributed training strategy of parameter averaging called Downpour. In this strategy, the parameter server receives the parameter values of each worker asynchronously and uses them to make an immediate update of the stored parameters of the model. The new parameters are returned to the workers. Intuitively, one may think that this strategy is not optimal, since the parameters become outdated as soon as a new worker sends its own, but in practice it has proven to be a highly efficient distributed training system [25–27].

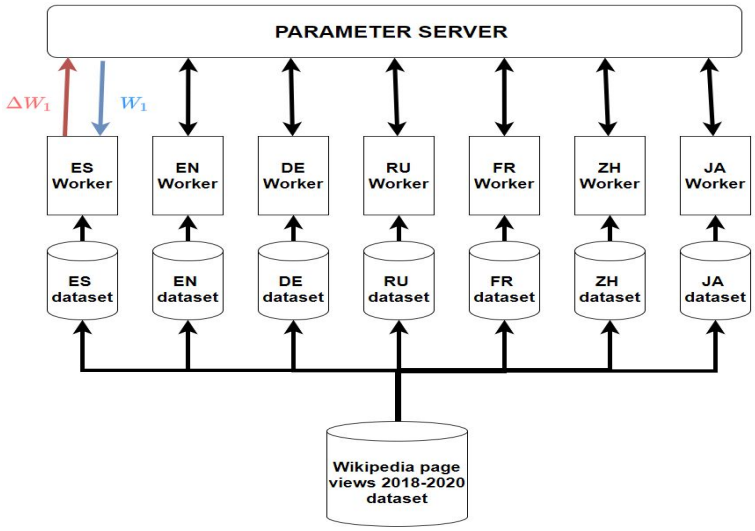


Figure 9. Distributed training of the seven workers with the parameter server updating the hyper-parameters of these LSTM difference by language.

3.3.3. LSTM Designed Model

Designing the LSTM was a challenge due to the limited data available. We had page views of 1085 days during 2018–2020. Furthermore, in the exploration of the dataset, it was seen that there were seven dominant languages in the dataset. Despite what one might intuitively think, each of the time series built for each wikipedia page view separated by language can be trained with the same LSTM, although it is important to keep in mind that each of the seven LSTMs could have different training parameters, which would mean a high training cost. To prevent this from being a problem, different techniques were used to improve the efficiency of the LSTM, although the result is that the LSTM is very deep and this is reflected in very high training times. For this reason, it was decided to use asynchronous distributed training using the Downpour strategy. A detailed scheme of the LSTM can be seen in Figure 10.

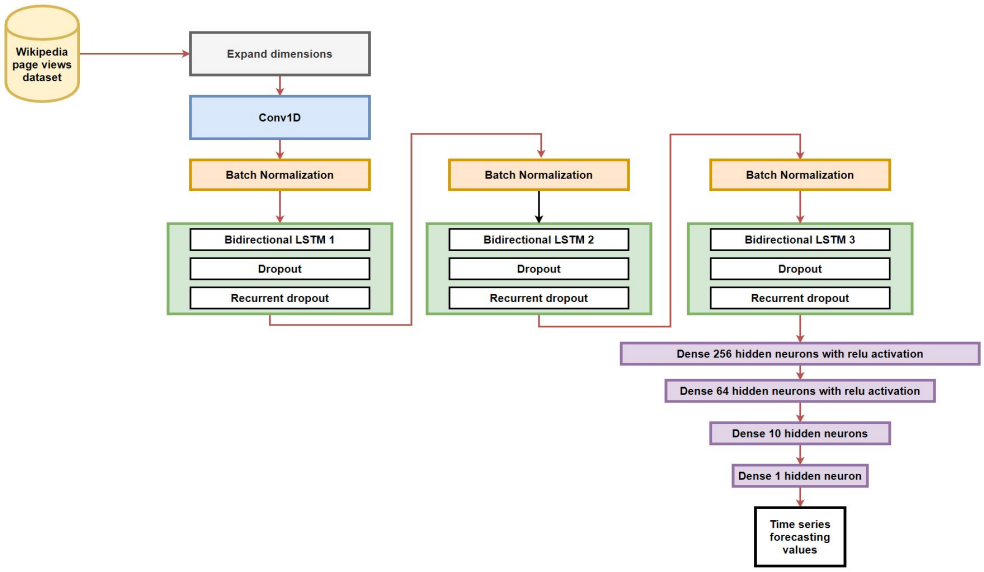


Figure 10. LSTM hidden layers. Data come from the scrapper, and then we expand the dimension to fit with the input of the Conv1D layer. Three attached LSTM layer are placed after the Conv1D layer but with batch normalization function before data input in the LSTM layer. Finally, there is a fully connected neural network to comply the design of our AI model.

We used TensorFlow and keras Python libraries for machine learning with additional connected supportive libraries to implement the proposed framework and to ensure their prediction performance. We also designed a Python function called "windowed_dataset". This is an auxiliary function that was used in some parts of our model for running the time series in small time windows that improve the accuracy for the functions that are being used. Since the time series present in our problem have a time component and a page views component, the first layer of our neural network is a dimensional expansion layer to transform that array into a new element suitable as input to our neural network. We used the TensorFlow function called "expand_dim". This function will not add or reduce elements in a tensor, it just changes the shape by adding 1 to dimensions. For example, if we are performing TensorFlow's Conv1D operation on vectors of rank 2, we need to feed them with rank three. Thus, since our data had dimension 2 (time and page views), we had to add one more dimension to make the time series fit with the Conv1D layer input. To improve the learning path of our model, a three-layer LSTM was placed after a Conv1D layer to make a convolution of the time series. After this, there are the three bidirectional LSTMs stacked, with the dropout and recurrent dropout regulator to reduce the overfiting as much as possible. The bidirectional LSTM was chosen because the LSTM scans data in both directions and in this way the learning process is improved in datasets with limited data. A batch normalization layer is used before the input of the stacked LSTMs to force the input data into a normal distribution shape, which helps to reduce the overfitting as well. Finally, there is a fully connected neural network with three hidden layers with Relu activation function, and a last layer with a single neuron in the whole layer is used to provide the result of the neural network. The optimizer that was used is Adam and the loss function is Huber. Our model was trained with a window size of 15 and a batch size of 5. The shuffle buffer was set at 1000 and trained for 200 epochs. Despite the thoroughness of the design of the neural network and all its internal layers, the time series of the page views in English at the end of the training had a high MAE. Although the LSTM did a good forecast, to check if the MAE improved with more epoch, we trained it with 500 and 1000 epochs, having as a result that it improved notably. Since the training is distributed and there are seven time series, the parameter server had to do the training in 200 epochs, since six out of seven time series had an accuracy above 94%, while the time series of the page views in English had 87% accuracy. In future work, we will investigate an asynchronous training strategy that will improve the Downpour strategy. For the training and validation purpose of the proposed AI model, we split the entire dataset into an 80:20 ratio, with 80% utilized for the training and 20% for testing or validating.

3.3.4. LSTM Hyperparameter Tuning

The optimal hyperparameters for the LSTM were determined through one of the upcoming techniques in the field of AI. This technique involves defining the neural networks as functions in Python whose parameters are the hyperparameters of the neural network. In this work, based on our previous experience designing LSTMs for the task of time series prediction, we designed the LSTM architecture as shown in Figure 10. The next step consists in creating lists with the range of values that the hyperparameters of our LSTM can take, such as the number of filters of the Conv1D layer, the number of inner units of each one of the LSTM layers, the dropout rate, etc. Finally, the training and validation loop is created in which all the different versions of our LSTM are executed and as a result a log is obtained with all the loss and MAE of each of the models that have been trained in the loop. Once finished, the most optimal model is searched and thus concludes the process to determine the most optimal hyperparameters of our LSTM model. A summary of this process can be found in Figure 11.

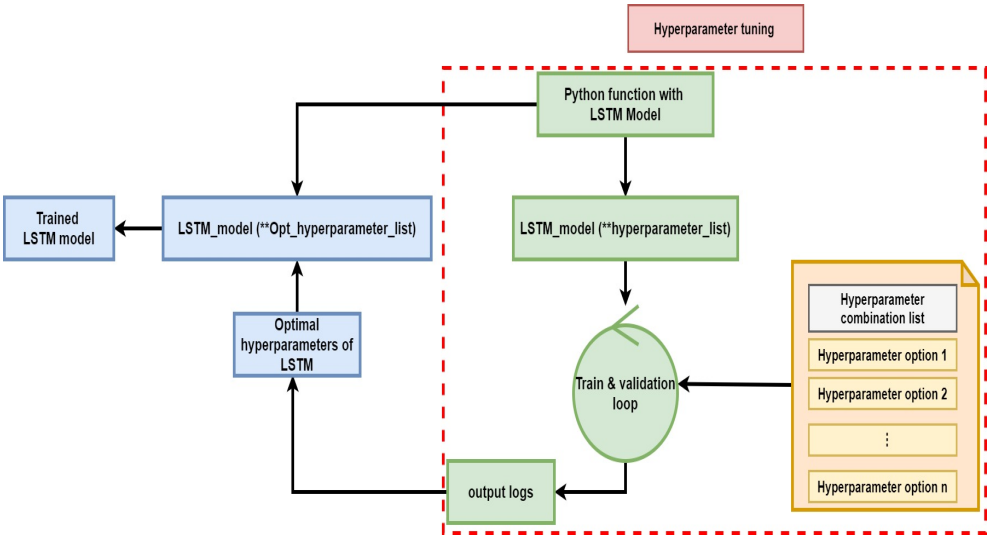


Figure 11. Detailed LSTM hyperparameter tuning process.

Once the hyperparameter tuning process was completed, the optimal values were determined. The hyperparameters we used in our experiment can be found in Table 1.

Table 1. Optimal hyperparameters value list of our LSTM model.

Hyperparameter Name	Hyperparameter Value
Conv1D—filters	32
Conv1D—kernel_size	4
LSTM1—units	180
LSTM1—dropout rate	0.2
LSTM1—recurrent_dropout rate	0.25
LSTM2—units	150
LSTM2—dropout rate	0.25
LSTM2—recurrent_dropout rate	0.25
LSTM3—units	100
LSTM3—dropout rate	0.3
LSTM3—recurrent_dropout rate	0.25

3.4. Model Evaluation Metric

To evaluate the performance of the proposed model, the Mean Absolute Error (MAE) was selected. MAE is used to quantify the difference between two continuous variables such as an original time series and a time series forecasted by our model. Analyzing these two time series relative to Wikipedia web traffic, MAE is used to evaluate the accuracy of a forecasting technique by comparing the predicted values with the observed values. We selected the evaluation method according to Equation (5). MAE is a technique widely used by researchers to evaluate the accuracy of their LSTM-based models [28,29].

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|Yreal_i - Ypred_i| \quad (5)$$

4. Experimental Results and Discussion

In the experiment, we chose Python, TensorFlow and Keras (deep learning framework) for develop our LSTM model. The proposed model was run and evaluated on windows 7 AWS virtual machines with Intel Core i7-7700, 16 G RAM, GTX 1060 GPU, as well as re-evaluated on Google cloud platform (google colab) with TPU.

We set the number of training epochs of the distributed system to 200 for the execution of the simulations of the distributed training system of our LSTM model. The hyperparameters of the layers were initialized following a normal distribution (mean = 0, stdv = 0.1) and the simulations were initialized. As shown in Figure 12, the forecast results (orange line) of the differentiated page views for each language fit quite well with the original time series (blue line). It is remarkable that our model is not able to predict the largest peaks that occur in the time series since they assume anomalous behavior of the time series. This is due to the fact that Wikipedia customers are unpredictable since they are human beings with their own decision-making capacity and freedom of action.

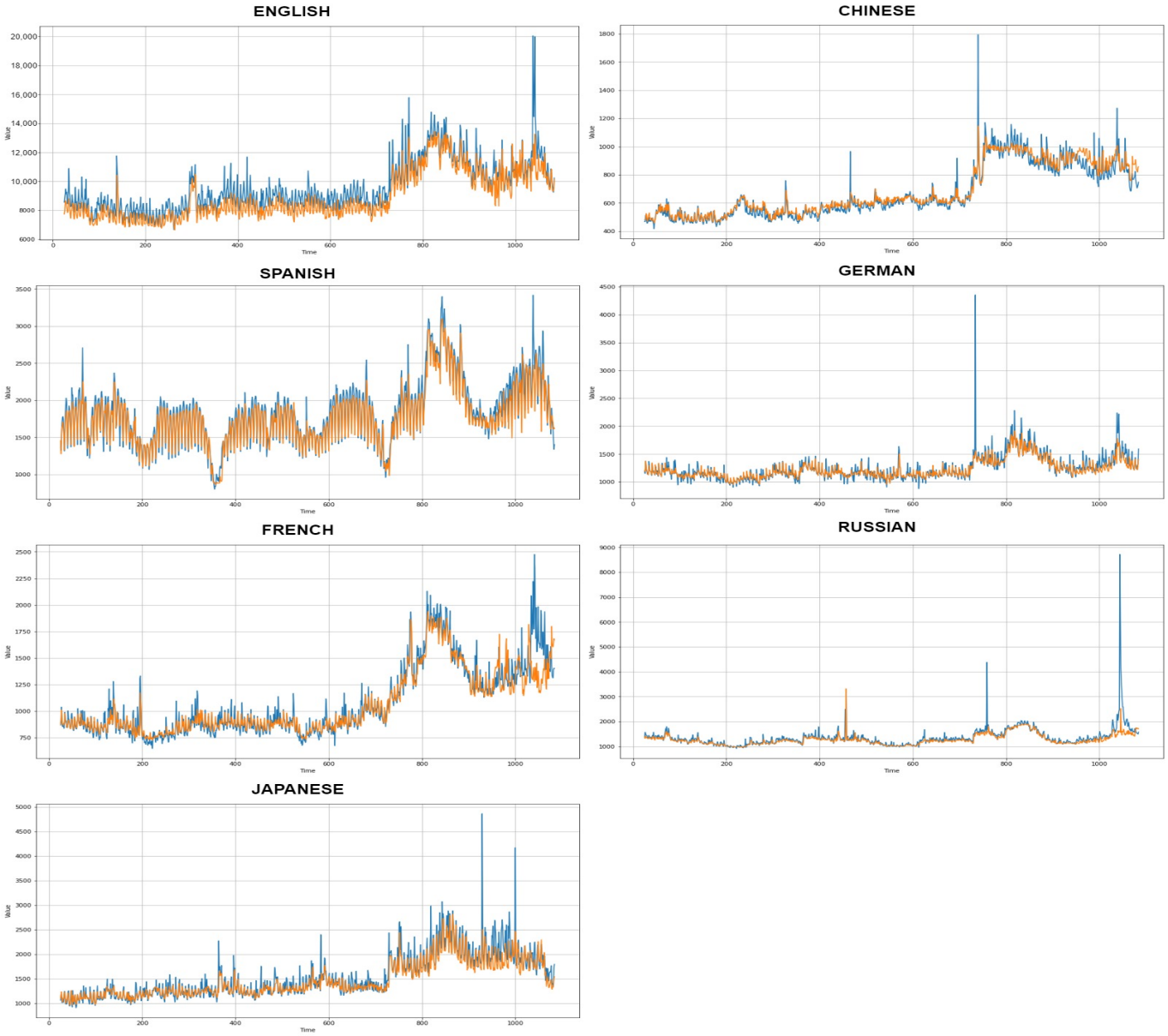


Figure 12. Forecasting (orange) of the Wikipedia web traffic page views difference by language in contrast with the original time series (blue).

For the purpose of validating the forecast of the designed LSTM, dataset was divided with 800 days in the training set and 250 days in the validation set (80:20 ratio). Figure 13 shows the zoomed forecast of each of the time series in those last 250 days. Although the accuracy is good, the model is not able to predict the anomalous behavior of the time series due to an increase of the visits to the Wikipedia due to some important event that occurs

in the world. In the case of the time series of the visits to the French Wikipedia, in the last 50 days, there is a remarkable prediction error that makes us think that those days something out of the ordinary happened that elevated the consultations to the Wikipedia due to some news of current importance in France. It is interesting to note the time series of the Russian Wikipedia; the LSTM was able to predict the final peak but with far fewer page views than actually occurred. Despite the fact that we designed the LSTM based on our findings about the extraction of features and hidden patterns in data, the limitations of LSTM in time series forecasting without proper seasonality and trend is slightly noticeable. Therefore, we think that our model is a bit more accurate than some of the state-of-the-art models for web traffic prediction [30–33]. However, we will investigate in future work the influence of news in social media such as Twitter or Instagram on page views web traffic.

Figure 13. Zoomed forecasting (orange) of the Wikipedia web traffic page views difference by language in contrast with the original time series (blue).

For concluding the results section, we make a discussion of MAE and the loss Huber function to test the accuracy of our model. The graphical values are shown in Figure 14 for the LSTM forecasting after 200 epochs. Intuitively, one can think that the MAE and the loss are good because they decrease at a good rate and end up stabilizing at relatively low values. However, this is not entirely true, as some of them did improve these indicators considerably with 500 and 1000 epochs. Table 2 shows the values of the MAE and loss in the different tests that were done before 200 was selected as the number of training epoch. Although all models perform satisfactorily with 200 epochs, some, such as the English model, may have been more accurate if 500 epochs or more had been used.

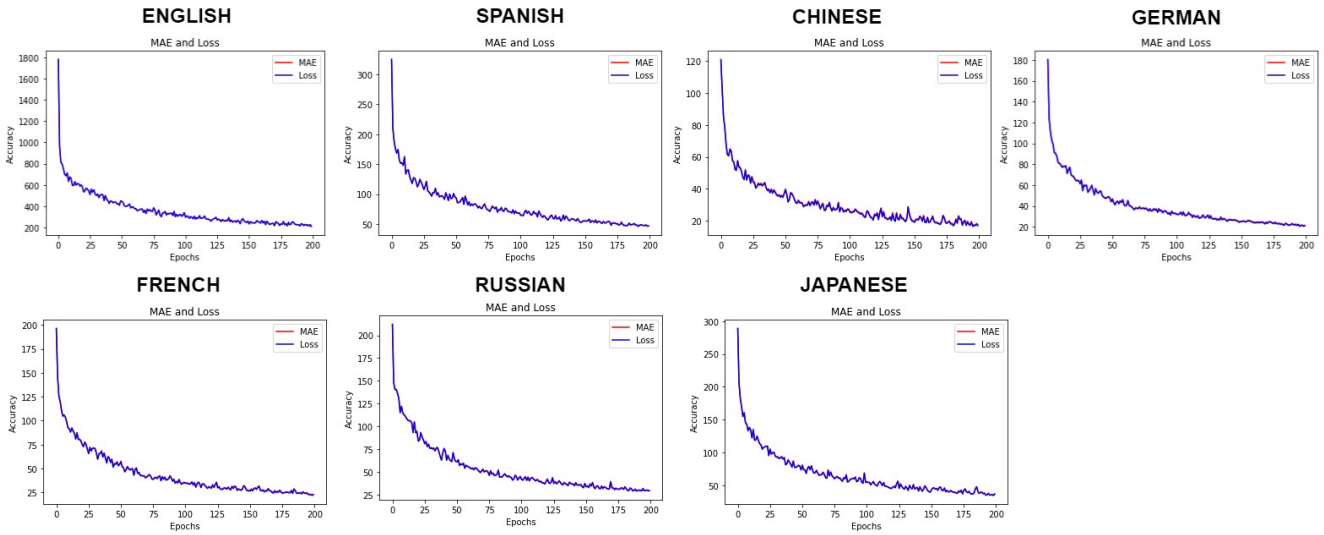
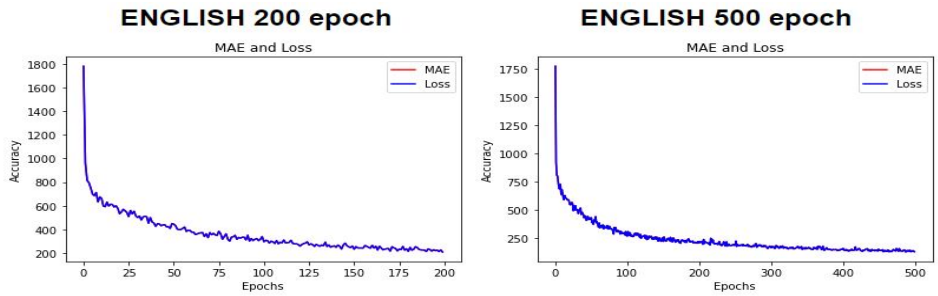


Figure 14. Loss and MAE of our proposed AI model difference by language.

Table 2. MAE in the several train of the LSTM with different epoch. We only performed 500 epochs of training with the English page view time series since it has a high MAE in contrast with the other time series' MAEs.

Language	1 Epoch	50 Epoch	100 Epoch	200 Epoch	500 Epoch
English	1775.5438	433.1056	274.1371	209.2002	132.2613
Spanish	324.9516	95.8202	69.0222	47.1441	-
German	180.6219	44.3626	33.1456	21.5006	-
French	196.4905	54.6633	35.5870	23.0779	-
Russian	212.3684	61.8643	45.2639	29.6669	-
Chinese	121.0158	37.2384	27.3859	17.4330	-
Japanese	289.1507	73.9427	54.6819	37.0835	-

It is possible that by augmenting the epochs and reducing the MAE considerably the time series forecast is becoming more accurate. However, regardless of what one intuitively thinks, the results show that this is not entirely certain. It should be noticed that these time series do not have good seasonality or trends; therefore, as they are based human behaviors, these data have a great influence on the sudden changes in the time series which can be seen in the significant peaks shown in the time series in this section. Figures 15 and 16 show this in deep analysis.



ENGLISH 200 epoch

```
Epoch 195/200
234/234 [==============================] - 31s 131ms/step - loss: 228.2800 - mae: 228.7795
Epoch 196/200
234/234 [==============================] - 30s 130ms/step - loss: 220.2266 - mae: 220.7261
Epoch 197/200
234/234 [==============================] - 31s 134ms/step - loss: 224.7409 - mae: 225.2403
Epoch 198/200
234/234 [==============================] - 31s 133ms/step - loss: 212.5781 - mae: 213.0773
Epoch 199/200
234/234 [==============================] - 32s 136ms/step - loss: 226.8373 - mae: 227.3368
Epoch 200/200
234/234 [==============================] - 30s 130ms/step - loss: 208.5869 - mae: 209.0863
```

ENGLISH 500 epoch

```
Epoch 495/500
234/234 [==============================] - 22s 95ms/step - loss: 131.4485 - mae: 131.9477
Epoch 496/500
234/234 [==============================] - 21s 91ms/step - loss: 134.5521 - mae: 135.0512
Epoch 497/500
234/234 [==============================] - 22s 92ms/step - loss: 135.6791 - mae: 136.1783
Epoch 498/500
234/234 [==============================] - 20s 84ms/step - loss: 142.2581 - mae: 142.7571
Epoch 499/500
234/234 [==============================] - 21s 89ms/step - loss: 138.2433 - mae: 138.7426
Epoch 500/500
234/234 [==============================] - 20s 87ms/step - loss: 131.7624 - mae: 132.2613
```

Figure 15. Loss and MAE for the comparison of 200 and 500 epochs on English Wikipedia page views time series.

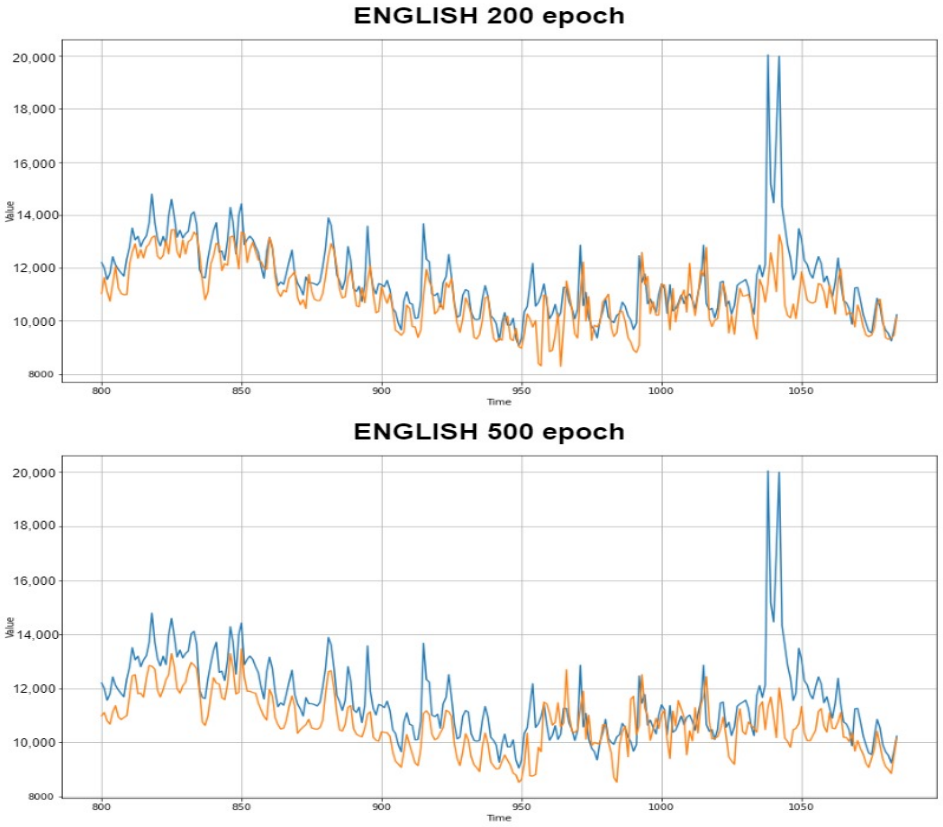


Figure 16. Zoomed version of the comparison of the forecast of the 200 and 500 epochs on English Wikipedia page views time series.

5. Conclusions and Future Work

In this study, we developed an architecture for web traffic forecasting based on artificial intelligence with LSTM for time series forecasting. For this purpose, we created a new dataset to validate our model and extracted the features and hidden patterns to enhance the design of the LSTM. A distributed training system was designed according to the concept of data parallelism along the lines of the Downpour asynchronous training strategy. Despite the limitations of our research such as a dataset with relatively limited data and the unpredictable nature of human behavior, we experimentally verified that the forecasting results of our AI model are pretty accurate and close to the real values. Moreover, we achieved quite good results in the training of the LSTM despite the limited data we had. In future works, the aim is to deepen in hidden pattern extraction for improving the efficiency of the LSTM and to study how human behavior affects the web traffic. To improve the performance of our model, we will investigate the unsupervised model proposed in previous papers (e.g., [34,35]).

Author Contributions: Conceptualization, R.C.-V. and A.M.d.R.; methodology, R.C.-V., A.M.d.R. and D.P.-P.; software, R.C.-V. and L.d.-l.-F.-V.; validation, R.C.-V., D.P.-P. and L.d.-l.-F.-V.; formal analysis, R.C.-V. and A.M.d.R.; investigation, R.C.-V., A.M.d.R., D.P.-P., L.d.-l.-F.-V. and J.M.C.; resources, J.M.C.; data curation, R.C.-V. and D.P.-P.; writing—original draft preparation, R.C.-V., A.M.d.R., D.P.-P., L.d.-l.-F.-V. and J.M.C.; writing—review and editing, R.C.-V., A.M.d.R., D.P.-P., L.d.-l.-F.-V. and J.M.C.; visualization, R.C.-V. and D.P.-P.; supervision, J.M.C.; project administration, J.M.C.; and funding acquisition, J.M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially Supported by the project "Intelligent and sustainable mobility supported by multi-agent systems and edge computing (InEDGEMobility): Towards Sustainable Intelligent Mobility: Blockchain-based framework for IoT Security", Reference: RTI2018-095390-B-C32, financed by the Spanish Ministry of Science, Innovation and Universities (MCIU), the State Research Agency (AEI) and the European Regional Development Fund (FEDER).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Network
TSF	Time Series Forecast
FFT	Fast Fourier Transform
MAE	Mean average error
ACF	Auto-correlation Function
PACF	Partial Auto-correlation Function
AI	Artificial Intelligence

References

1. Chen, D.; Gao, M.; Liu, A.; Chen, M.; Zhang, Z.; Feng, Y. A Recurrent Neural Network Based Approach for Web Service QoS Prediction. In Proceedings of the 2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD), Chengdu, China, 25–28 May 2019; pp. 350–357.
2. Zhou, K.; Wang, W.; Huang, L.; Liu, B. Comparative study on the time series forecasting of web traffic based on statistical model and Generative Adversarial model. *Knowl.-Based Syst.* **2020**, *213*, 106467. [CrossRef]
3. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. The M4 Competition: 100,000 time series and 61 forecasting methods. *Int. J. Forecast.* **2020**, *36*, 54–74. [CrossRef]
4. Yang, Y.; Lu, S.; Zhao, H.; Ju, X. Predicting Monthly Pageview of Wikipedia Pages by Neighbor Pages. In Proceedings of the 2020 3rd International Conference on Big Data Technologies, Qingdao, China, 18–20 September 2020; pp. 112–115.

5. Bojer, C.S.; Meldgaard, J.P. Kaggle forecasting competitions: An overlooked learning opportunity. *Int. J. Forecast.* **2020**. [CrossRef]
6. Fry, C.; Brundage, M. The M4 Forecasting Competition-A Practitioner's View. *Int. J. Forecast.* **2019**. [CrossRef]
7. De Gooijer, J.G.; Hyndman, R.J. 25 years of time series forecasting. *Int. J. Forecast.* **2006**, *22*, 443–473. [CrossRef]
8. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLoS ONE* **2018**, *13*, e0194889. [CrossRef]
9. Montero-Manso, P.; Athanasopoulos, G.; Hyndman, R.J.; Talagala, T.S. Fforma: Featurebased forecast model averaging. *Int. J. Forecast.* **2020**, *36*, 86–92. [CrossRef]
10. Rangapuram, S.S.; Seeger, M.W.; Gasthaus, J.; Stella, L.; Wang, Y.; Januschowski, T. Deep state space models for time series forecasting. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 7785–7794.
11. Tealab, A. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Comput. Inform. J.* **2018**, *3*, 334–340. [CrossRef]
12. Tyralis, H.; Papacharalampous, G. Variable selection in time series forecasting using random forests. *Algorithms* **2017**, *10*, 114. [CrossRef]
13. Chen, W.C.; Chen, W.H.; Yang, S.Y. A big data and time series analysis technology-based multi-agent system for smart tourism. *Appl. Sci.* **2018**, *8*, 947. [CrossRef]
14. Boone, T.; Ganeshan, R.; Jain, A.; Sanders, N.R. Forecasting sales in the supply chain: Consumer analytics in the big data era. *Int. J. Forecast.* **2019**, *35*, 170–180. [CrossRef]
15. Madan, R.; SarathiMangipudi, P. Predicting computer network traffic: A time series forecasting approach using DWT, ARIMA and RNN. In Proceedings of the 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, India, 2–8 August 2018; pp. 1–5.
16. Le, P.; Zuidema, W. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs. *arXiv* **2016**, arXiv:1603.00423.
17. Suilin, A. kaggle-web-traffic. 2017. Available online: https://github.com/Arturus/kaggle-web-traffic/ (accessed on 19 November 2018).
18. Cinar, Y.G.; Mirisaee, H.; Goswami, P.; Gaussier, E.; Aït-Bachir, A.; Strijov, V. Position-based content attention for time series forecasting with sequence-to-sequence rnns. In Proceedings of the International Conference on Neural Information Processing, Guangzhou, China, 14–18 November 2017; Springer: Cham, Switzerland, 2017; pp. 533–544.
19. Qin, Y.; Song, D.; Chen, H.; Cheng, W.; Jiang, G.; Cottrell, G. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv* **2017**, arXiv:1704.02971.
20. Liang, Y.; Ke, S.; Zhang, J.; Yi, X.; Zheng, Y. Geoman: Multi-level attention networks for geo-sensory time series prediction. In Proceedings of the 2018 International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13–19 July 2018; pp. 3428–3434.
21. Smagulova, K.; James, A.P. A survey on LSTM memristive neural network architectures and applications. *Eur. Phys. J. Spec. Top.* **2019**, *228*, 2313–2324. [CrossRef]
22. Miyaguchi, A.; Chakrabarti, S.; Garcia, N. Forecasting Wikipedia Page Views with Graph Embeddings. 2019. Available online: http://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26647399.pdf (accessed on 30 November 2020).
23. Wunnava, V.P. Exploration of Wikipedia traffic data to analyze the relationship between multiple pages. Master's Thesis, University of North Carolina, Chapel Hill, NC, USA, May 2020.
24. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
25. Srinivasan, A.; Jain, A.; Barekatain, P. An analysis of the delayed gradients problem in asynchronous sgd. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
26. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Le, Q.V.; Mao, M.Z.; Ranzato, M.; Senior, A.; et al. Large scale distributed deep networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1223–1231.
27. Talyansky, R.; Kisilev, P.; Melamed, Z.; Peterfreund, N.; Verner, U. Asynchronous SGD without gradient delay for efficient distributed training. In Proceedings of the International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019.
28. Tian, C.; Ma, J.; Zhang, C.; Zhan, P. A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network. *Energies* **2018**, *11*, 3493. [CrossRef]
29. Liu, Y.; Guan, L.; Hou, C.; Han, H.; Liu, Z.; Sun, Y.; Zheng, M. Wind power short-term prediction based on LSTM and discrete wavelet transform. *Appl. Sci.* **2019**, *9*, 1108. [CrossRef]
30. Liu, Z.; Yan, Y.; Hauskrecht, M. A flexible forecasting framework for hierarchical time series with seasonal patterns: A case study of web traffic. In Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, Ann Arbor, MI, USA, 8–12 July 2018; pp. 889–892.
31. Shelatkar, T.; Tondale, S.; Yadav, S.; Ahir, S. Web Traffic Time Series Forecasting using ARIMA and LSTM RNN. In *Proceedings of the ITM Web of Conferences 2020*; EDP Sciences: Ulis, France, 2020; Volume 32, p. 03017.
32. Petluri, N.; Al-Masri, E. Web Traffic Prediction of Wikipedia Pages. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 5427–5429.

33. Du, S., Pandey, M., & Xing, C. Modeling Approaches for Time Series Forecasting and Anomaly Detection. Technical Report. 2017. Available online: http://cs229.stanford.edu/proj2017/final-reports/5244275.pdf (accessed on 30 November 2020).
34. Ragno, R.; Papa, R.; Patsilinakos, A.; Vrenna, G.; Garzoli, S.; Tuccio, V.; Fiscarelli, E.; Selan, L.; Artini, M. Essential oils against bacterial isolates from cystic fibrosis patients by means of antimicrobial and unsupervised machine learning approaches. *Sci. Rep.* **2020**, *10*, 1–11. [CrossRef]
35. Ieracitano, C.; Paviglianiti, A.; Campolo, M.; Hussain, A.; Pasero, E.; Morabito, F.C. A novel automatic classification system based on hybrid unsupervised and supervised machine learning for electrospun nanofibers. *IEEE/CAA J. Autom. Sin.* **2020**, *8*, 64–76.