Distributed Deep Learning for Big Data Time Series Analysis

Chapter · August 2022

DOI: 10.1007/978-3-031-15063-0_31

5 authors, including:

Quang Hoang Dinh
Ho Chi Minh City University of Information Technology
4 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Trong-Hop Do
University of Information Technology VNU-HCM, Ho Chi Minh City, Vietnam
72 PUBLICATIONS **816** CITATIONS
SEE PROFILE

Distributed deep learning for big data time series analysis

Dinh-Quang Hoang[1,2], Dang-Khoa Tran[1,2], Viet-Thang Le[1,2], The-Manh Nguyen[1,2] and Trong-Hop Do[1,2]

[1] University of Information Technology, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam
{18521294,18520936,18520356,18521084}@gm.uit.edu.vn
Correspondence: hopdt@uit.edu.vn

Abstract. This paper deals with the problem of traffic flow prediction in large scale, which has three major challenges. First, the prediction must be performed in large scale that requires lots of computation. Second, achieving good accuracy of time series prediction is challenging for most traditional machine learning algorithms. Third, the amount of training data is so big that makes it very difficult to train on a single machine. This paper solves all these problem by proposing a big data time series analysis system using distributed deep learning. The entire system runs on Apache Spark big data framework, which allows the prediction to be performed in large scale. A Temporal Convolutional Networks (TCN) model is proposed and applied to achieve high accuracy of traffic flow prediction. The model are trained on a huge amount of training data using a distributed deep learning framework namely BigDL. The experimental results show that the system can predict the traffic flow in large scale at accuracy levels that are much higher than that of traditional machine learning models.

Keywords: big data · time series analysis · distributed deep learning.

1 Introduction

The achievements and contributions of Artificial Intelligence (AI) to our world are increasing day by day. One of the important factors contributing to the success of AI is deep learning. Applications of deep learning have supported people with many things from work to life. In the healthcare field, disease diagnosis systems can diagnose many times faster and more accurately than humans. Sentiment analysis or fraud detection systems used by Google, Twitter, Facebook, etc. have brought them huge profits. In transportation, many traffic problems such as traffic jams, traffic accidents, etc. are also significantly solved thanks to intelligent traffic management systems, smart cars or self-driving cars are also one of the outstanding achievements of AI today. In order to get quality deep learning models that can be applied in practice as above, a good algorithm is not enough, data is the core to create a good model. In recent years, the amount

of data has increased rapidly due to the development of science and technology, especially social networks, e-commerce, sensors, etc.led to a data explosion and the emergence of Big Data. This large and variety amount of data is both an advantage and a challenge for current deep learning. Some fields in Big Data such as Computer Vision, Voice Recognition have applied Deep Learning to improve model results. Deep learning algorithms extract meaningful abstract representations of the raw data through the use of an hierarchical multi-level learning approach, where in a higher-level more abstract and complex representations are learnt based on the less abstract concepts and representations in the lower levels of the learning hierarchy [1]. However, in addition to the advantages, the characteristics of big data also created many challenges for deep learning algorithms such as dealing with streaming input data, huge amount of data, high-dimensional data, large-scale models, etc.

Time series data is one of the most widely used data types today, even real time series data. Time series data analysis is widely used in forecasting tasks for management, monitoring, etc. Algorithms or models for time series problems are often more complex than other problems because they do not only exploit relationships and correlations between attributes, but also need to exploit relationships among time points to get the best performance. Current deep learning algorithms such as Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), etc. have given high performance for real-time problems. But to do that, the training time for these models is quite long, it can be many days, many weeks, even many months. Not only that, the cost of system resources consumed for training is very expensive. And when the volume of data is constantly increasing, up to hundreds of millions, billions of data points, models with billions of parameters, the current server and machine resources may not be enough to train the model.

To contribute to solving the above problems, in this paper, a method for big data time series analysis by deep learning is proposed. This method based on distributed computing principles to build deep learning models for big data time series analysis on Apache Spark platform. This deep learning model uses the BigDL framework, which is a distributed deep learning framework for big data. BigDL implements distributed, data parallel training directly on top of the functional compute model of Spark [2]. The experiment was conducted on the prestigious large traffic time series dataset published on IEEE DataPort. The experimental results is the comparison of the performance of the deep learning model trained by the traditional method with the deep learning model trained by the proposed method and between the models trained by the proposed method and the proposed method through evaluation measures and training time. Experimental results will be presented in detail in the following sections of the paper.

2 Fundamental of big time series analysis and distributed deep learning

2.1 Big data processing

Big Data is a term that refers to very large and complex data sets including structured, semi-structured or unstructured data, which makes data processing software system's inability to collect, manage and process data within a reasonable period of time. This data comes from many different sources such as from sensors that collect traffic information, weather forecasts, updates, images, videos on social networking sites, transactions on exchanges. e-commerce, etc. Currently, there is still no exact answer to the question of what data is called Big Data. People often judge whether data is Big Data or not based on 5V's: Volume, Velocity, Variety, Veracity and Value Volume is a very large volume of data, measured in Terabytes, Exabytes, and more. Velocity is the growth of the data in terms of speed. This growth rate is extremely fast. Variety is the variety of data. Traditionally we used to be organized as structured data, today most of the stored data is created unstructured (video, images, audio, data from sensors, etc.). Veracity is the reliability or accuracy of data and Value is the value of the data. Data is only really useful when it is collected from a reliable data source and the data must bring some value to the individual, business or organization that owns it. Big data processing includes all tasks from collection, storage, preprocessing, feature extraction, model training, etc. For example, big data must use cloud technology to store because the volume is very large. Big data analysis also needs to reach real-time speeds and handle a variety of data types.

To serve tasks related to big data processing, Apache Spark was born and became an extremely useful platform. Apache Spark is an open source framework used mainly for big data analysis, machine learning and real-time processing based on distributed processing. Apache Spark is faster, easier to use, and more flexible than traditional platforms like Hadoop. By performing computations on the cluster that enables high-speed data analysis when reading and writing data. This processing speed is due to the fact that Spark's computations are performed at the same time on many different machines and are performed either in-memory or entirely in RAM. Apache includes many libraries for various tasks from SQL to machine learning, real-time data processing, and support for many programming languages like Scala, Python, Java, and more.

2.2 Time series analysis

Time Series Analysis or Sequence Analysis is the next prediction in a given input sequence based on what was previously observed. Predictions can be anything that might happen next: a symbol, a number, the next day's weather, the next term in the speech, and so on. Time series analysis can be very useful in applications such as stock market analysis, weather forecasting, and product recommendations. There are many approaches to the problem of time series

analysis such as based on probabilistic statistical models, machine learning, deep learning, etc. Auto Regressive Integrated Moving Average (ARIMA) is a traditional time series forecasting model based on the hypothesis of stationary series and constant error variance. The model will represent multiple linear regression equations of the input variables (dependant variables) as 3 components, Auto Regression (AR), Intergrated (I) and Moving Average (MA).

Prophet is also a powerful model for time series forecasting. Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well [3]. Prophet model is built based on 3 main components: the trend function g(t), seasonality function s(t) and holidays function h(t). Prophet's output is calculated according to the formula shown in (1).

$$y(t) = g(t) + s(t) + h(t) + \epsilon_l. \tag{1}$$

In addition, in deep learning, Long-Short Term Memory (LSTM) is a commonly used model for time series analysis. LSTM is designed as a special form of RNN to avoid long-term dependency. The predictions of the LSTM network ar:e always adjusted based on past experience. The core components of the LSTM model are the cell states and the task of controlling information flows in the network is performed by gates, the LSTM model includes update gate controls the retrieval of information of the previous class (or unit), output gate decides how much information to output and forget gate decides what information to keep and which information is close to being ignored. Suppose x_t is the input at time t, h_t is the output of which cell state C_t. The LSTM algorithm is shown in the order of equations from (2) to (6).

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{3}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

$$\text{“}o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{5}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{6}$$

2.3 Distributed deep learning

Distributed machine learning refers to multi-node machine learning algorithms and systems designed to improve performance, increase accuracy, and scale to

larger input data sizes [4]. Distributed deep learning is a field in distributed machine learning and is applied in many different fields because of its effectiveness. Distributed deep learning includes two main types, data parallelism and model parallelism. In data parallelism, the input data is divided into partitions equal to the number of worker nodes (machines) and processed in parallel. Model weights are replicated for all worker nodes and trained on the partitioned data for that node. The resulting weights or gradients at the nodes after each iteration will be aggregated at the Parameter Server and shared back to the nodes to calculate for the next iteration. The results at Parameter Server can be aggregated in many different ways, the simplest example is parameter averaging. Instead of partitioning the data for worker nodes, in model parallelism, the model itself is divided into parts that are trained simultaneously across different worker nodes. All worker nodes use the same data set and they also send the resulting model weights to the Parameter Server for aggregation and the Parameter Server returns the latest model. Model parallelism is much more difficult to implement and only works well in models with naturally parallel architectures. Therefore, the model proposed in this paper uses data parallelism.

BigDL is one of the distributed deep learning frameworks for Apache Spark, which has been used by a variety of users in the industry for building deep learning applications on production big data platforms released in 2019 [5]. Recently, BigDL released version 2.0 with the combination of the original BigDL and Analytics Zoo. With powerful features like DLlib, Orca, RayOnSpark, Chronos, etc. BigDL is really powerful for Big Data analysis tasks.

3 Proposed system

3.1 Proposed TCN model

The model used in the experiment is a basic TCN model (Temporal Convolutional Network) with an architecture of 3 temporal blocks. Each temporal block consists of two 1D Causal Convolutional layers and alternately Normalization, Dropout and ReLU activation layers as described in the architecture in Figure 1. The model is designed with an input sequence length of 288 corresponding to one day's speed information. The number of input features is 5, each of which contains speed information for a road in the dataset. The number of outputs equal to the input is 5, containing the forecasting results for the next time. Dropout ratio used is 0.1

3.2 Distributed deep learning training

From the model architectures that have been designed, the paper trains deep learning models by distributed method, in the form of data parallelism. Distributed training process is carried out according to the procedure Figure 2. The entire process is done on the Spark platform First, historical data on traffic speed is collected from the data source. Then, the entire data is passed through

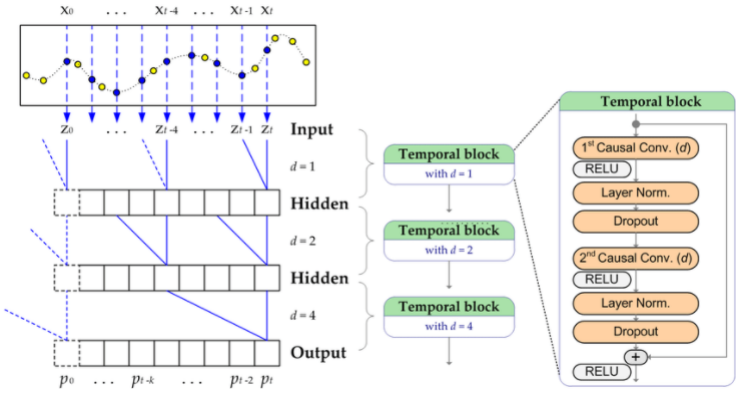

Fig. 1. Proposed TCN model

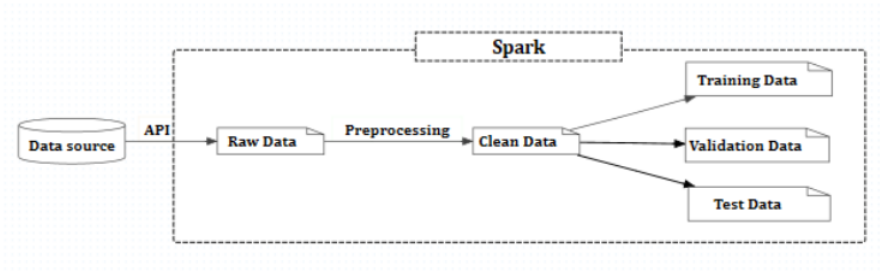

Fig. 2. Data collection and splitting procedure

preprocessing steps on the Apache Spark platform to clean and convert the data back to the correct model training format. The cleaned data is divided into three parts: Training Data, Validation Data and Test Data as shown in Figure 2. After the data preparation step, proceed to use two libraries, Orca and RayOnSpark in BigDL to set up a distributed environment. This distributed environment is a cluster of distributed machines running on the Apache Spark platform. The training and validation data was fed into the newly established distributed environment and converted into the standard time series data format in BigDL TSDataset using the Chronos library. Chronos also supports the implementation of the designed deep learning model on a distributed environment to facilitate the training process. According to the form of data parallelism introduced above, data will be partitioned into equal parts and distributed among machines in Spark's distributed cluster (worker node). The installed deep learning model is also replicated to all machines in the distributed cluster and trained on separate pieces of data on those machines. The details of the training process are shown in Figure 3.

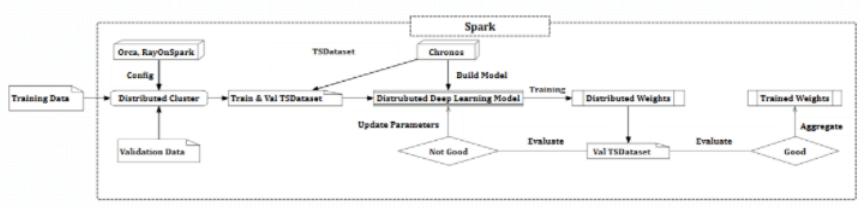

Fig. 3. Training procedure

During distributed training, the model weights are updated after the forward-backward propagation has been completed at the machines in the distributed cluster. After the forward-backward process, the machines will calculate the local gradients and update the weights for the model according to the parameter synchronization method. Specifically, the main model weight is divided into n partitions for calculation, the local gradients are also divided into n parts corresponding to the partitions, the partitions will perform the synthesis of the corresponding local gradients. for that partition by summing and performing a weight update for that partition. Each partition after completing the weight update will send the results to the Parameter Server to synthesize into a complete set of weights for the model, preparing for the next forward-backward step. This process is repeated until the training is complete. After completing the dis-

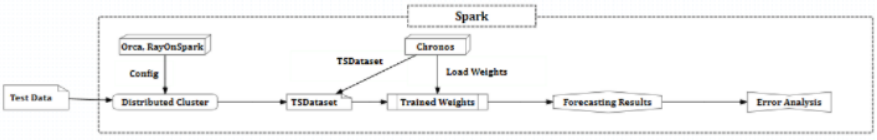

Fig. 4. Test procedure

tributed training for the model, the model is tested on the test set according to the procedure of Figure 4. If the test results are not good, the model will be edited, updated the parameters and re-trained. On the contrary, if the test results are good, the model will be kept and the training phase will end, moving to the evaluation and error analysis phase. The process of evaluating and analyzing model errors on the test dataset is also distributed similarly to the training process, the Chronos library provides the ability to reload the model from the trained weight set to progress. testing, evaluation and fault analysis. The test and evaluation results are presented in detail in the experimental part of the paper.

4 Experiment

4.1 Experimental environment

The whole experiment is done on Ubuntu 20.04 environment with 400GB RAM and 24 CPUs Intel(R) Xeon(R) E5-2670 0 @ 2.60GHz, 24 cores. The training data goes through several preprocessing steps such as deleting unnecessary columns, group by, pivot, interpolation to convert to the correct format of the model input. This entire data preprocessing is done on the Spark platform, written in the Python programming language supported by PySpark.

4.2 Experiment results

After the experimental process, the distributed TCN model shows very good predictive ability on many evaluation levels including several hours, 1 day and 1 week. The model's forecast results are shown in Figures 5, 6 and 7. The resulting graphs show that the deviation between the model's forecast results and the actual speed is very small. There are times when my prediction is almost completely accurate. The model training time is also relatively short at 6939.01 seconds. From all the above results, it can be seen that the distributed trained deep learning model gives the performance not only not inferior to the traditional training methods, but also can scale the model on big data.

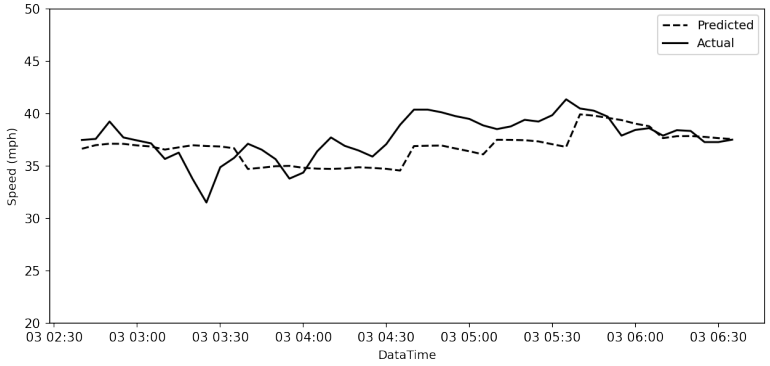

Fig. 5. Forecast results of the model in a few hours

5 Conclusion

Traffic flow prediction is a very challenging problem in practical due to both the required huge computation and the difficulties of time series prediction problem.

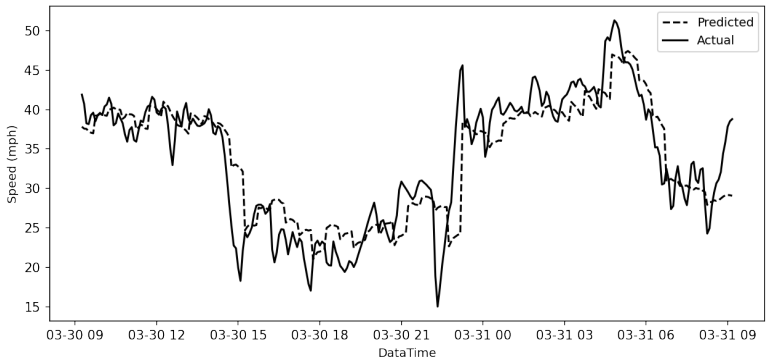

Fig. 6. Forecast results of the model in 1 day

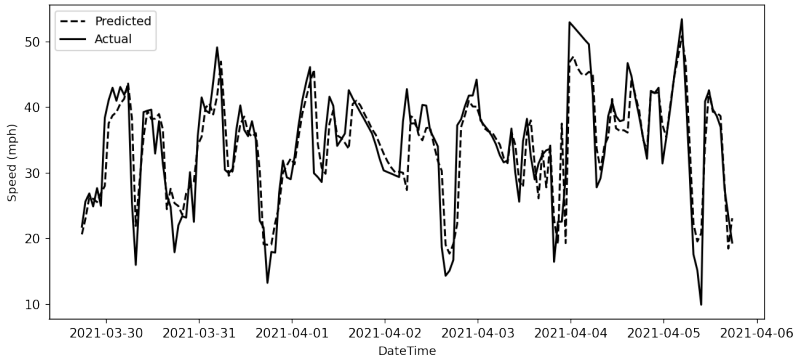

Fig. 7. Forecast results of the model in 1 week

This paper proposed a traffic flow prediction system that run on Apache Spark big data framework to allow predictions to be performed in large scale. A deep learning model namely Temporal Convolutional Networks is proposed for traffic flow prediction and achieve very high performance compared to that achieved with traditional machine learning model. Due to the huge amount of training data, the model is trained on a computer cluster using a distributed deep learning framework namely BigDL. The experimental results show that the proposed system can predict the traffic flow at very high accuracy in large scale.

References

1. Najafabadi, M.M., Villanustre, F., Khoshgoftaar, T.M. et al. Deep learning applications and challenges in big data analytics. Journal of Big Data 2, 1 (2015).
2. Dai, Jason Jinquan, et al. "Bigdl: A distributed deep learning framework for big data." Proceedings of the ACM Symposium on Cloud Computing. 2019.
3. Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2 https://doi.org/10.7287/peerj.preprints.3190v2.
4. Galakatos A., Crotty A., Kraska T. (2018) Distributed Machine Learning. In: Liu L., Özsu M.T. (eds) Encyclopedia of Database Systems. Springer, New York, NY.
5. Dai, J. J., Wang, Y., Qiu, X., Ding, D., Zhang, Y., Wang, Y., ... and Song, G. (2019, November). Bigdl: A distributed deep learning framework for big data. In Proceedings of the ACM Symposium on Cloud Computing (pp. 50-60).