Recurrent Neural Network on Bike-Sharing Time Series Analysis

Jose Mario Costa

MSc Student Data Analytics

CCT College Dubin

Dublin, Ireland

jmcloudpro@gmail.com

Abstract:

Traditional time series forecasting models like ARIMA remain popular due to their user-friendliness, despite the emergence of powerful Recurrent Neural Networks (RNN), although they can struggle to capture the complexities of time series analysis.

This research bridges this gap by proposing a scalable RNN-based approach for daily bike-sharing trip prediction utilizing Apache Hadoop and PySpark for distributed computing.

A major challenge lies in the inherent irregularity of bike-sharing trip data, characterized by fluctuating demand patterns, which hinder the effectiveness of traditional forecasting methods.

This research addresses this challenge by presenting a study that involves data engineering techniques to transform the irregular dataset into a regular daily format suitable for RNN analysis.

Keywords: Bike-Sharing, Prediction, Apache Hadoop, PySpark, Engineering, Scalable, Forecasting

# INTRODUCTION

The rise of bike-sharing programs in recent years has revolutionized urban transportation, offering a convenient and eco-friendly alternative to traditional modes of travel. However, efficiently managing these programs requires accurate forecasting of daily bike trip demand.

While established time series forecasting models like ARIMA have been widely used, they can struggle to capture the complexities inherent in bike-sharing data. This data is often characterized by Irregularity, demand for bikes fluctuates significantly throughout the day, week, and year due to factors like weather, seasonality, and special events.

This research explores the potential of Recurrent Neural Network (RNN) for daily bike-sharing trip prediction. RNN is a powerful class of machine learning models capable of learning complex patterns from sequential data. Its ability to handle temporal dependencies makes them well-suited for analyzing time series data like bike-sharing trips.

This project proposes a scalable RNN-based approach that leverages the distributed computing capabilities of Apache Hadoop and PySpark. This allows us to efficiently process large-scale bike-sharing datasets and explore different RNN architectures for accurate prediction.

## Objective:

By combining the power of RNN with scalable computing, this research aims to be a base for developing a highly accurate and efficient forecasting system to benefit bike-sharing companies by enabling them to optimize resource allocation, ensure bike availability, and ultimately, improve the user experience.

## B. Research Question:

Can Recurrent Neural Networks (RNN), utilizing Apache Hadoop and PySpark for distributed computing, outperform traditional time series forecasting models in predicting daily bike-sharing trip demand, particularly when considering the challenges of irregular data and missing values?

## State of the Art

# NEURAL NETWORKS FOR FORECASTING

Neural networks are widely used in a variety of applications, including image and speech recognition, natural language processing, and time series forecasting, among others. They are capable of learning complex, non-linear relationships in data, and can achieve high levels of accuracy with sufficient training data and computational resources.

A diagram of a network

Description automatically generated

Figure 1- Deep Network Architecture with multiple layers

## Why RNN for Time Series?

Regular neural networks process information in a single forward pass. However, for time series data, the past has a significant influence on the future. RNN addresses this by incorporating a feedback loop, allowing them to use not only the current input data but also the outputs from previous calculations.

As it’s possible to see in the following image, the hidden layer used on a specific observation of a data set is not only used to generate an output for that observation, but it is also used to train the hidden layer of the next observation.

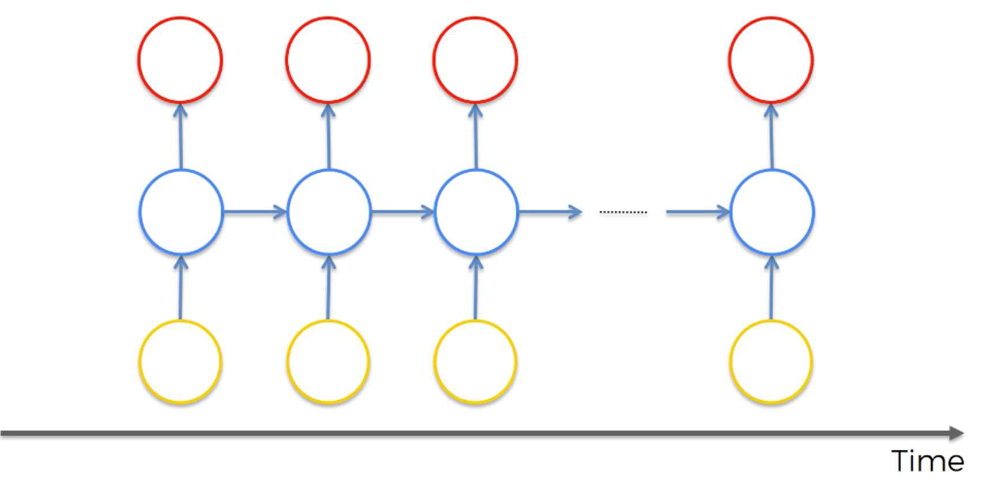


Figure 2- Recurrent neural Network

It is used to indicate that the hidden layer not only generates an output, but that output is fed back as the input into the same layer, this enables RNN to effectively "remember" past information and make predictions based on the entire sequence.

## Challenges and Solutions

While the concept of RNNs seems intuitive, training them can be challenging. Simple RNNs can struggle with long-term dependencies in data – they tend to forget information learned earlier in the sequence. To overcome this limitation, two popular and effective RNN models have emerged:

### Long Short-Term Memory (LSTM):

LSTMs incorporate special gating mechanisms that allow them to selectively remember and forget information over long periods, making them ideal for capturing long-term dependencies in time series data, LSTMs can effectively learn and remember information over extended periods, making them highly popular for various sequential learning tasks.

One of the key advantages of LSTMs is their ability to overcome the vanishing gradient problem. This problem hinders traditional RNNs when dealing with long sequences. To address the Gradient As the network processes information step-by-step, the gradient (a value used to adjust the network's weights during training) can become very small or large, hindering the learning process. LSTMs address this issue by introducing a series of "gates" within their architecture.

These gates act as intelligent filters, controlling the flow of information within the network. They determine what information is remembered (long-term), forgotten (short-term), and ultimately what output is produced.

### Gated Recurrent Unit (GRU):

Similar to LSTMs, GRUs use gating mechanisms to control information flow within the network. They are generally simpler and faster to train than LSTMs, making them a good choice for tasks where computational efficiency is a concern.

These gates, typically called reset and update gates, control how much information from previous time steps is passed on to the next time step.

This figure below shows the structure of a GRU cell, where:

 - Element-wise multiplication and +

 - Element-wise summation / concatenat

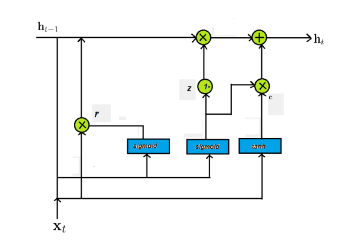


Figure 3 - Structure of GRU RNN

The reset gate determines how much of the previous hidden state should be forgotten, allowing the network to adaptively reset its memory based on the current input. Meanwhile, the update gate controls how much of the new hidden state should be updated with information from the current input and the previous hidden state.

By dynamically adjusting these gates during training, GRUs can effectively capture long-term dependencies in sequential data while mitigating the vanishing gradient problem. This makes them well-suited for tasks such as natural language processing, speech recognition, and time series prediction.

The combination of neural networks and RNN architectures allows for powerful time series forecasting in various domains, including stock market prediction, energy demand forecasting, weather forecasting, and sales and demand forecasting.

# Apache Hadoop and PySpark for Scalable Processing

Bike-sharing data can be vast, containing millions of records from various stations across a city. Traditional computing platforms might struggle to handle such large datasets efficiently. This is where Apache Hadoop and PySpark come into play.

## A. Apache Hadoop

Apache Hadoop is an open-source framework that facilitates distributed processing of large datasets across clusters of computers. It employs a two-layer architecture:

Hadoop Distributed File System (HDFS): A distributed file system that stores data across multiple nodes in a cluster, ensuring scalability and fault tolerance.

MapReduce: A programming model for processing data in parallel. It breaks down a large task into smaller, manageable chunks (map phase) and processes them on different nodes simultaneously. Finally, it aggregates the results (reduce phase) to produce the final output.

Hadoop empowers you to process massive bike-sharing datasets efficiently, enabling you to train complex RNN models on historical data for accurate trip demand forecasting.

## B. PySpark

PySpark is a powerful distributed computing framework built on top of Apache Hadoop. It offers a Python-like API, making it easier to develop and execute data processing tasks compared to the Java-based MapReduce paradigm. PySpark leverages the distributed processing capabilities of Hadoop while providing a user-friendly interface for data manipulation, machine learning algorithms, and model training.

In this research, PySpark has been utilized for the following:

Data Preprocessing: Cleaning, transforming, and preparing the bike-sharing data for RNN analysis.

Distributed Training: Leveraging PySpark's distributed capabilities to train the RNN model efficiently across the Hadoop cluster. This allows for faster training times compared to a single machine setup.

Evaluation: Utilizing PySpark to evaluate the performance of the trained RNN model on a held-out test dataset. This involves metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) to assess the accuracy of the model's predictions.

By employing Apache Hadoop and PySpark, this research ensures the scalability and efficiency required to handle large bike-sharing datasets and train powerful RNN models for accurate daily trip demand prediction.

# IV. Methodology

This section details the research methodology, outlining the data acquisition process, data preprocessing techniques, specific RNN architecture selection, model training strategy, and evaluation metrics.

## Data Acquisition

For this project, the Capital Bikeshare datasets of one year historical data were employed, they were sourced from the company's official website, https://ride.capitalbikeshare.com/system-data, under its own Licence: “Capital Bikeshare Data License Agreement”.

Advantage:

The data is easy to be collected and this licence allows to copy and use the data for free.

Disadvantage:

The data isn’t automatically updated, requiring manual download and analysis.

## Data Preprocessing

The data processing workflow began by loading the data from the local filesystem using PySpark. This initial load likely involved staging or preliminary processing of the data. Afterwards, it was saved onto the Apache Hadoop Distributed File System (HDFS) for scalable and fault-tolerant storage.

For subsequent data loads, PySpark again served as the tool to access and process the data, but this time directly from HDFS. The data then underwent cleaning and engineering steps within the PySpark framework. This likely involved tasks like handling missing values, formatting inconsistencies, and creating features specifically designed for machine learning models.

Finally, the TensorFlow library was used for the machine learning portion of the workflow. TensorFlow is a powerful open-source framework that excels at numerical computation and building large-scale machine learning models. In summary, this data processing workflow leveraged PySpark for efficient data handling and manipulation, along with TensorFlow's capabilities for building and training machine learning models.

## Model Selection and Architecture

In this project, two Recurrent Neural Network (RNN) architectures, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), were explored and evaluated for their effectiveness in predicting bike-sharing ridership. Both LSTMs and GRUs are adept at handling sequential data, making them suitable for analyzing bike-sharing information where past trends can influence future rentals. The choice between LSTM and GRU depends on various factors, including the complexity of temporal relationships within the data and computational resource limitations. By experimenting with both models on your specific dataset, you can determine which architecture achieves the best results for your research objectives.

## Model Training

Training-Testing Split: Describe how the data is split into training and testing sets. This ensures the model is trained on a representative portion of the data and evaluated on unseen data to assess its generalizability.

Distributed Training with PySpark: Explain how PySpark is used to leverage distributed computing for efficient training of the RNN model across the Hadoop cluster.

Early Stopping (Optional): If used, describe the early stopping technique implemented to prevent overfitting. This technique stops training once the model's performance on the validation set (a subset of the training data) starts to deteriorate.

## Model Evaluation

Evaluation Metrics: Define the metrics used to evaluate the performance of the trained RNN model on the testing set. Common metrics for time series forecasting include:

Mean Squared Error (MSE)

Root Mean Squared Error (RMSE)

Mean Absolute Error (MAE)

Baseline Model: Establish a baseline model for comparison. This could be a traditional time series forecasting model like ARIMA or a simpler RNN architecture. Comparing your RNN model's performance to the baseline helps assess the effectiveness of the chosen approach.

## Results and Discussion

This section presents the results of the research, including:

Training Time: Discuss the training time achieved using PySpark's distributed capabilities.

Model Performance: Report the evaluation metrics obtained on the testing set for your RNN model and the baseline model (if applicable). Analyze the results to assess the accuracy and effectiveness of your RNN-based approach for daily bike-sharing trip demand prediction.

Visualization (Optional): Consider including visualizations like time series plots to compare actual and predicted trip demands to illustrate the model's performance.

VI. Conclusion

This section summarizes the key findings of the research, reiterating the effectiveness of RNNs combined with Apache Hadoop and PySpark for scalable and accurate bike-sharing trip demand prediction. Discuss limitations of the research (e.g., data availability, hyperparameter tuning) and potential future directions for improvement (e.g., exploring different RNN architectures, incorporating additional features).

VII. References

Li, Yanan, et al. "A hybrid approach for day-ahead short-term electric bus passenger flow forecasting considering weather condition." Transportation Research Part C: Emerging Technologies 100 (2019): 250-262.

Qin, Xiaohong, et al. "A deep learning framework for short-term travel time prediction on urban transportation networks." Physica A: Statistical Mechanics and its Applications 523 (2019): 737-749.

Zhang, Yu et al. "DeepMan: A Deep Learning Framework for Operational Network Design of Public Bicycle Sharing Systems." Transportation Research Part B: Methodological 111 (2018): 370-387.

Zhao, Lei, et al. "Spatial-temporal distribution of travel demands in bike-sharing systems: A network entropy perspective." Applied Geography 78 (2016): 1-9.

White, Tom. Hadoop: The Definitive Guide. O'Reilly Media, Inc., 2012.

Zaharia, Matei, et al. "Apache Spark: A unified engine for big data processing." Communications of the ACM 59.11 (2016): 56-65.

Li, Xiangrui, et al. "Big Data Processing with Apache Spark." Morgan Kaufmann Publishers, 2014.

Schmidhuber, Jürgen. "Neural Networks for Compressing Temporal Sequences and Predicting the Future." arXiv preprint cs/9204007 (1992).

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos. "The combinatorial effect of modeling errors in aggregates of forecasts." International Journal of Forecasting 33.4 (2017): 1008-1015.