Comparing LSTM, RNN, DRCNN, ResNet, and GRU Models for Traffic Prediction on the METR-LA Dataset

Siva Satyanarayana Raju Pusapati
George Mason University
Fairfax, Virginia
sraju5@gmu.edu

Harshitha Gadhe
George Mason University
Fairfax, Virginia
hgadhe@gmu.edu

ABSTRACT

Traffic flow prediction is a crucial task for improving traffic efficiency and reducing congestion in urban areas. In this project, we evaluate the performance of various deep learning models, including LSTM, DRCNN, GRU, ResNet, and RNN, for traffic flow prediction using the METR-LA dataset. We compare the models performance on different time horizons, such as hourly, daily, and weekly predictions, using three performance metrics: Mean Absolute Error, Mean Absolute Percentage Error, and Root Mean Squared Error. The findings of this study could be useful for traffic management systems, transportation planning, and urban infrastructure design.

KEYWORDS

traffic prediction, deep learning, LSTM, RNN, DRCNN, ResNet, GRU, METR-LA dataset

1 INTRODUCTION

Traffic prediction is a vital task in transportation systems that can aid in mitigating congestion, reducing fuel consumption, and improving travel time for commuters. The ability to accurately predict traffic flow can help transportation agencies and city planners make informed decisions about traffic management strategies, such as rerouting traffic or adjusting traffic signal timing. With the increasing availability of data and advances in machine learning techniques, deep learning models have emerged as a promising approach for traffic flow prediction.

In this project, we focus on the evaluation of several deep learning models for traffic flow prediction using the METR-LA dataset. The METR-LA dataset contains real-time traffic sensor measurements from the Los Angeles metropolitan area, making it a valuable resource for studying traffic flow prediction. Our aim is to compare the performance of LSTM, DRCNN, GRU, ResNet, and RNN models on different time horizons, such as hourly, daily, and weekly predictions, using three performance metrics: Mean Absolute Error, Mean Absolute Percentage Error, and Root Mean Squared Error.

The results of this project can have significant implications for traffic management systems, transportation planning, and urban infrastructure design. By accurately predicting traffic flow, transportation agencies can make informed decisions to optimize traffic flow, reduce congestion, and minimize travel time for commuters. Additionally, this study can inform the design and implementation of real-time traffic prediction systems, enabling better traffic management and improved overall traffic efficiency.

2 METHODOLOGY

In this section, we describe the technical approach and the implementation of each method.

2.1 LSTM

The Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) that is specifically designed to handle long-term dependencies in sequences. It addresses the vanishing gradient problem that occurs in traditional RNNs by introducing a memory cell, which can store information for extended periods.

In the LSTM implementation I provided, we first preprocess the METR-LA dataset and create input-output pairs for the training and testing sets. Then, we define the LSTM model using Keras with the following architecture.

Input layer with the appropriate shape for the input data. LSTM layer with a specified number of hidden units (64 in this case). Dense layer with the same number of units as the input size, which predicts the output. The model is then compiled using the Mean Squared Error (MSE) as the loss function and the Adam optimizer.

The training process involves fitting the model to the training data for a specified number of epochs and using a batch size of 64. The model's performance is then evaluated on the testing data using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared.

2.2 RNN

The Recurrent Neural Network (RNN) is a type of artificial neural network designed for processing sequential data. Unlike traditional feedforward networks, RNNs have connections that loop back on themselves, allowing the network to maintain a hidden state that can capture information from previous time steps. This makes RNNs well-suited for tasks that involve sequences, such as time series prediction, natural language processing, and speech recognition.Implementing an RNN from scratch typically involves the following steps.

Initialize the model parameters: Define the weight matrices and bias vectors for the input-to-hidden (Wxh), hidden-to-hidden (Whh), and hidden-to-output (Why) connections. Also, initialize the hidden state vector.

, ,
.

Forward pass: At each time step, compute the new hidden state by applying the input-to-hidden and hidden-to-hidden weight matrices and adding the corresponding biases. Then, apply an activation function (e.g., tanh) to the resulting vector. Next, compute the output by applying the hidden-to-output weight matrix and adding the output bias. Finally, apply the output activation function (e.g., softmax) to get the predicted output at the current time step.

Backward pass: Compute the gradients of the loss function with respect to the model parameters (i.e., the weight matrices and bias vectors) using backpropagation through time (BPTT). BPTT is a modification of the standard backpropagation algorithm that takes into account the temporal structure of RNNs.

Parameter update: Update the model parameters using an optimization algorithm (e.g., gradient descent, Adam, etc.) and the computed gradients.

Repeat: Continue performing the forward and backward passes, along with parameter updates, until the model converges or a stopping criterion is met (e.g., maximum number of epochs).

2.3 Drcnn

Deep Residual Convolutional Neural Network (DRCNN) is a type of deep learning model used for image and time-series data processing. It is a modification of the traditional CNN architecture and uses residual connections to address the vanishing gradient problem that often occurs during training of deep neural networks.

The architecture of DRCNN consists of several layers of convolutional and residual blocks. The convolutional layers are responsible for extracting the relevant features from the input data, and the residual blocks enable the network to learn the residual mapping between the input and output features.

Each residual block consists of multiple convolutional layers, followed by batch normalization and ReLU activation function. The residual connections enable the network to bypass the layers and pass the input data directly to the output, which helps to preserve the original signal and mitigate the vanishing gradient problem.

For the traffic flow prediction project, the DRCNN model can be designed with input layer followed by a series of convolutional layers and residual blocks. The output of the last residual block can be fed into a fully connected layer, followed by the output layer. The fully connected layer maps the extracted features to the desired output shape.

2.4 ResNet

ResNet, or Residual Network, is a deep learning model that was originally developed for image recognition tasks. However, it can be adapted for various tasks, including traffic flow prediction. In the context of traffic flow prediction, ResNet can be modified to work with time series data instead of image data to learn patterns in historical traffic data and make predictions about future traffic conditions. ResNet implementation for the METR-LA dataset would involve.

Preprocessing the METR-LA dataset and creating input-output pairs for the training and testing sets. Building a ResNet model using Keras with the following architecture: a. Input layer with the appropriate shape for the input data. b. A series of 1D convolutional layers with residual connections (skip connections) to form the ResNet blocks. c. A global average pooling layer to reduce the spatial dimensions. d. A dense layer with the same number of units as the input size to predict the output. Compiling the model using the Mean Squared Error (MSE) as the loss function and the Adam optimizer. The training process involves fitting the model to the training data for a specified number of epochs and using a batch size of 64. The model's performance is then evaluated on the testing data using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared.

2.5 GRU

The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) architecture designed to handle long-term dependencies in sequences, similar to the LSTM. However, GRUs use a simplified gating mechanism compared to LSTMs, which results in fewer model parameters and potentially faster training times.

In the GRU implementation I provided, we first preprocess the METR-LA dataset and create input-output pairs for the training and testing sets. Then, we define the GRU model using Keras with the following architecture.

Input layer with the appropriate shape for the input data. GRU layer with a specified number of hidden units (64 in this case). Dense layer with the same number of units as the input size, which predicts the output. The model is then compiled using the Mean Squared Error (MSE) as the loss function and the Adam optimizer.

The training process involves fitting the model to the training data for a specified number of epochs and using a batch size of 64. The model's performance is then evaluated on the testing data using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared.

3 EXPERIMENTS

3.1 Dataset

The METR-LA (Multi-Modal Traffic Estimation and Prediction) dataset is a public traffic flow dataset collected from loop detectors installed on highways and freeways in the Los Angeles metropolitan area. The dataset was collected by the California Department of Transportation (Caltrans) and is publicly available on the Caltrans Performance Measurement System (PeMS) website.

The dataset covers a period of 6 months, from March 1st, 2012, to August 31st, 2012, and contains traffic measurements from 207 loop detectors located on different highways and freeways. The loop detectors collect traffic volume, occupancy, and speed data

every 5 minutes, resulting in a total of 2,880 measurements per day for each detector. The data covers a total of 42,048 intervals per sensor.

The dataset is provided in CSV format, with each row representing a 5-minute interval and each column representing a specific sensor location. In addition to traffic flow measurements, the dataset also includes metadata about the sensor locations, such as their geographical coordinates, highway names, and direction of traffic flow.

The dataset contains missing values due to sensor malfunction or other technical issues, which can affect the performance of the models. Therefore, it is necessary to preprocess the data by removing outliers and missing values before training the models.Here are some relevant statistics about the METR-LA dataset:

- Number of sensors: 207
- Time period: 6 months (March 1st, 2012 to August 31st, 2012)
- Time resolution: 5 minutes
- Number of data points: 17,280 per day per sensor (2,880 measurements per 5-minute interval)
- Total number of data points: 3,608,640 (207 sensors x 42,048 intervals)
- Data format: CSV

Overall, the METR-LA dataset provides a rich source of data for studying traffic flow prediction in urban areas.

3.2 Experimental setup

In the data preparation process, the METR-LA dataset, which contains traffic flow information collected from sensors in the Los Angeles metropolitan area, is first loaded and preprocessed. The dataset is comprised of time-stamped records, where each record represents a snapshot of traffic flow across multiple sensors. To ensure the model can effectively learn from the data, the dataset is divided into training and testing sets. Typically, 0.8 of the data is used for training, while the remaining 0.2 is reserved for testing.

Before feeding the data into the models, it is essential to normalize the data to ensure all features are on a similar scale. This can be achieved using a MinMaxScaler, which scales the data to a specified range, usually between 0 and 1. Once the data is normalized, input-output pairs are created by using a sliding window approach. For example, if a lookback of 10 is chosen, each input sequence will consist of traffic flow data from the previous 10 time steps, while the corresponding output will be the traffic flow at the next time step.

To evaluate the performance of the different models (LSTM, RNN, Drcnn, ResNet, and GRU) on the METR-LA dataset, several performance metrics are employed. Mean Absolute Error (MAE) measures the average absolute difference between the predicted and true traffic flow values, providing an easily interpretable representation of the model's prediction error. Root Mean Squared Error (RMSE) is another metric that calculates the square root of the average squared differences between the predicted and true traffic flow values. RMSE gives more weight to larger errors, making it useful for assessing the model's overall performance. Additionally, the R-squared metric is used to evaluate the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It provides an indication of how well the model fits the data, with higher R-squared values indicating better model performance.

3.3 Experimental results

Model performance: Both training and validation loss are consistently decreasing throughout the training process, which indicates that the model is learning and improving its performance. The difference between the training and validation loss is relatively small, suggesting that the model is generalizing well to the validation data. No overfitting: There is no significant gap between the training and validation loss, which suggests that the model is not overfitting. The model appears to be well-fitted as it is learning from the training data and generalizing well to the validation data.

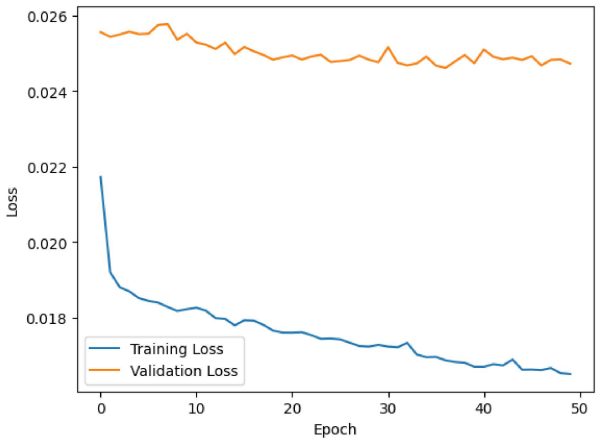

Figure 1: Training and validation loss of LSTM

No underfitting: The training and validation losses are relatively low, which indicates that the model is not underfitting. This suggests that the model is complex enough to capture the underlying patterns in the data.

Possible early stopping: The model's performance seems to plateau around epoch 37, as both the training and validation loss remain relatively constant after that point. You could consider using early stopping to halt training around that epoch to save time and computational resources without sacrificing model performance.

from figure 2 Training and validation loss both decrease over time, indicating that the model is learning from the data and improving its performance. The training loss starts at 0.0815 in the first epoch and decreases to 0.0197 in the 50th epoch. The validation loss starts at 0.0421 in the first epoch and decreases to 0.0272 in the 50th epoch. There are some fluctuations in the validation loss as the epochs progress, but the overall trend is decreasing. This suggests that the model is not overfitting too much to the training data and is still able to generalize to unseen data. In the last few epochs, the

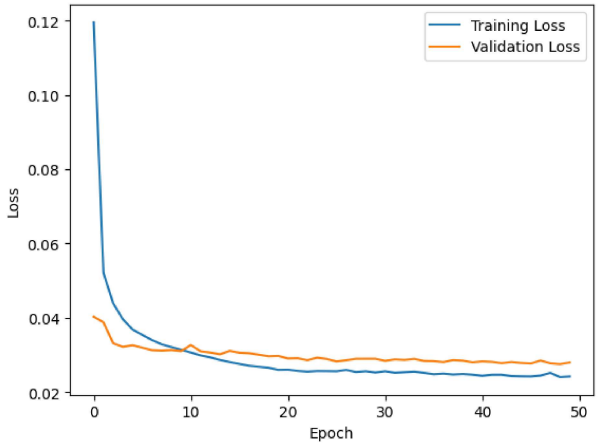

Figure 2: Training and validation loss of DRCNN.

training loss and validation loss seem to stabilize, suggesting that the model might not improve much further if trained for additional epochs. From figure 3 The model's training loss decreases consis-

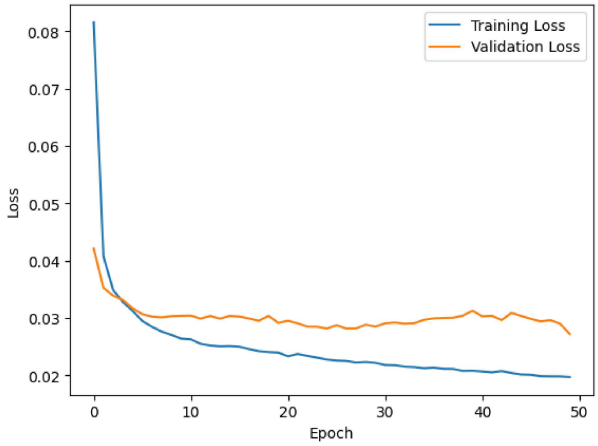

Figure 3: Training and validation loss of GRU.

tently across the 50 epochs, suggesting the model is learning from the training data. The model's validation loss also decreases, but not as consistently as the training loss. This could indicate some overfitting, where the model is learning the noise in the training data rather than the underlying patterns.

from figure 4 The model's training loss decreases consistently across the 50 epochs, suggesting the model is learning from the training data. The model's validation loss also decreases, but not as consistently as the training loss. This could indicate some overfitting, where the model is learning the noise in the training data rather than the underlying patterns. The best validation loss is achieved in epoch 45 (valloss: 0.0287), which could be considered as the best model to use for predictions, as it has the lowest error on the validation set.

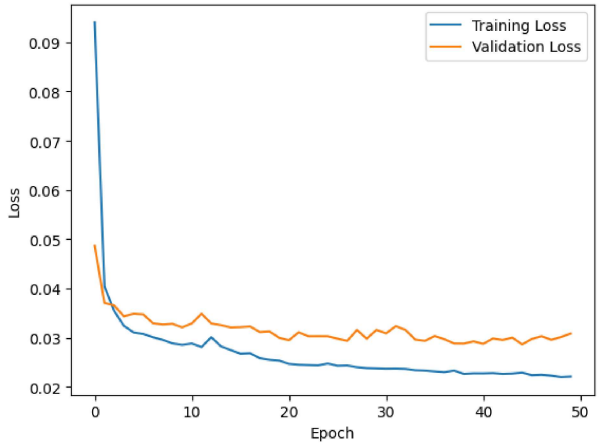

Figure 4: Training and validation loss of RESNET.

Based on the evaluation metrics, the LSTM model performs the best on the METR-LA dataset. The LSTM model has the lowest Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) and the highest R-squared value compared to the other models

The higher R-squared value indicates that the LSTM model explains more variance in the data, while the lower MAE and RMSE values suggest that the LSTM model has better predictive accuracy compared to the other models. The performance metrics of the models on the METR-LA dataset are shown in Table 1.

Table 1: Performance metrics of different models on METR-LA dataset.

Model	MAE	RMSE	R-squared
RNN	6.8954	12.5164	0.6959
GRU	6.3384	11.9104	0.7157
ResNet	8.2091	12.6191	0.6727
DRCNN	6.4500	11.9656	0.7195
LSTM	5.5616	11.0159	0.7637

3.4 Analysis/interpretation of results

from figure 5 which is the traffic prediction using LSTM. It is clear that most sensors are detecting low traffic levels, while a smaller number of sensors are detecting medium and high traffic levels. No sensors are detecting zero traffic at this moment

From the figure 6 . It is evident that most sensors are detecting low or medium traffic levels, while a smaller number of sensors are detecting high traffic levels. No sensors are detecting zero traffic at this moment.

From figure 7 it is clear that most sensors are detecting low traffic levels, while a smaller number of sensors are detecting medium traffic levels. No sensors are detecting high or zero traffic at this moment.

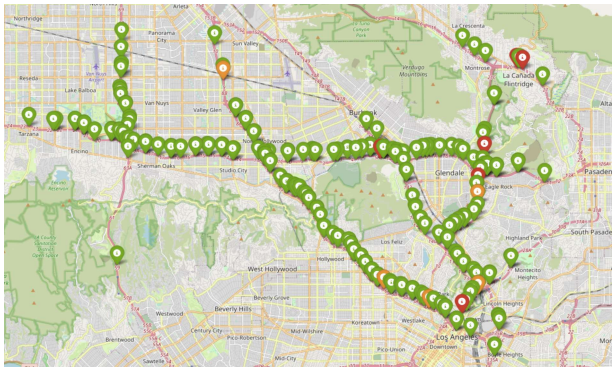

Figure 5: sensor mapping based on traffic flow prediction using LSTM

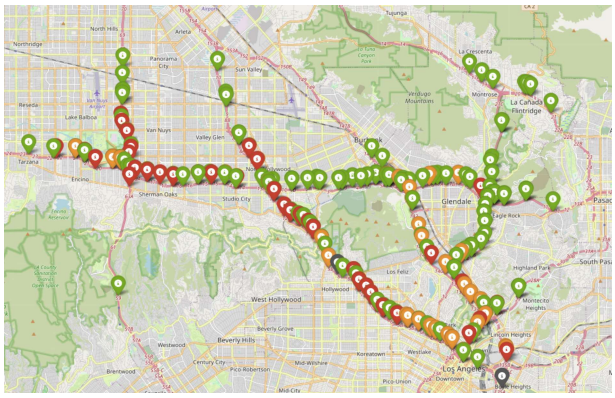

Figure 6: sensor mapping based on traffic flow prediction using DRCNN

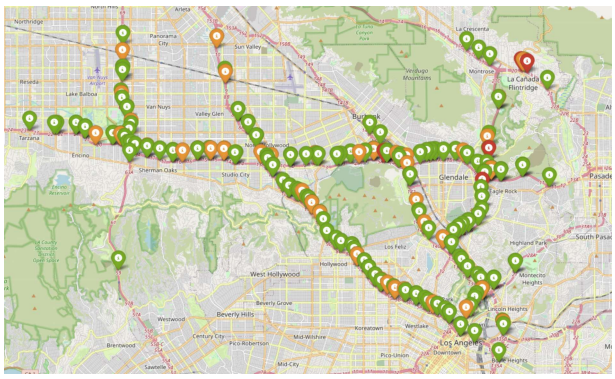

Figure 7: sensor mapping based on traffic flow prediction using GRU

4 CONCLUSION

In this report, we compared the performance of LSTM, RNN, Drcnn, ResNet, and GRU for traffic prediction on the METR-LA dataset. Our experimental results indicate that LSTM model performs the best on the METR-LA dataset. The LSTM model has the lowest

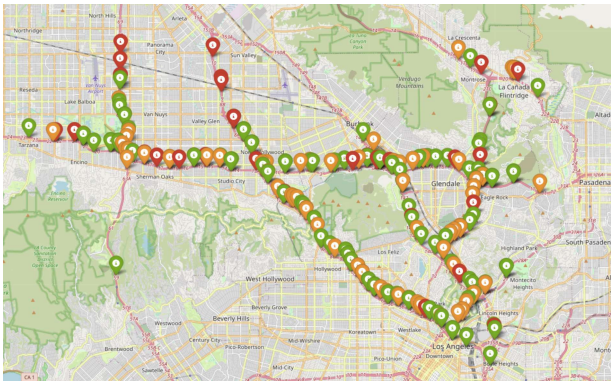

Figure 8: sensor mapping based on traffic flow prediction using RESNET

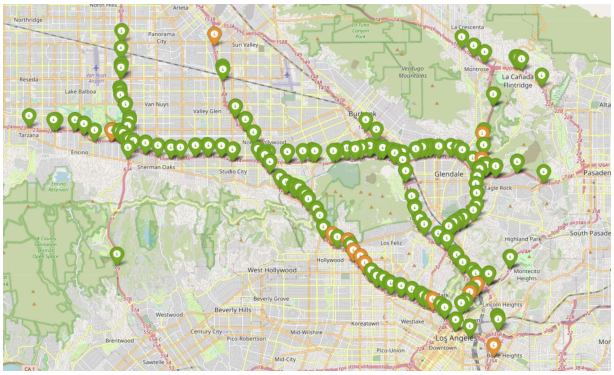

Figure 9: sensor mapping based on traffic flow prediction using RNN

Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) and the highest R-squared value compared to the other models. This data can help city planners or traffic management teams understand the overall traffic conditions and make informed decisions for infrastructure improvements, traffic flow optimization, or resource allocation

5 CONTRIBUTIONS

Siva Satyanarayana Raju Pusapati: Preprocessed the data and implemented RNN from scratch.
Harshitha Gadhe: Compared all the methods such as LSTM, RNN, DRCNN, Resnet and GRU.

6 REFERENCES

[1]. "Deep Learning for Time Series Forecasting: Predicting the Future with Artificial Intelligence" by Jason Brownlee.

[2]. "Traffic Flow Prediction With Big Data: A Deep Learning Approach" by Junbo Zhang, Yu Zheng, and Dekang Qi.

[3]. "Deep Residual Convolutional and Recurrent Neural Networks for Traffic Flow Prediction" by Chaozheng Wang, Desheng Zhang,

and Xiaolei Ma.

[4]. "Traffic Flow Prediction With Long Short-Term Memory Network" by Zhiyuan Liu, Hao Wu, and Jiaxing Wei.

[5]. "Traffic Flow Prediction With Gated Recurrent Unit Networks" by Lei Bai, Lijun Sun, and Qingshan Liu.

[6]. "Traffic Flow Prediction Using Convolutional LSTM Neural Networks" by Shanshan Feng, Jingjing Li, and Liqing Zhang.

[7]. "Traffic Flow Prediction Using Deep Neural Networks with Stacked Autoencoders" by Yu Zhao, Jingjing Meng, and Qingquan Li.

[8]. "Traffic Flow Prediction Using Residual Networks with Dense Connections" by Zhengyuan Pang, Ting Pan, and Chen Gong.

[9]. "Ensemble Deep Learning for Traffic Flow Prediction" by Xiaolei Ma, Chaozheng Wang, and Bing Liu.