TRR
JOURNAL OF THE TRANSPORTATION RESEARCH BOARD

Research Article

Graph-Partitioning-Based Diffusion Convolutional Recurrent Neural Network for Large-Scale Traffic Forecasting

Transportation Research Record
2020, Vol. 2674(9) 473–488

DOI: 10.1177/0361198120930010
journals.sagepub.com/home/trr
SAGE

Tanwi Mallick[1], Prasanna Balaprakash[1,2], Eric Rask[3], and Jane Macfarlane[4]

Abstract

Traffic forecasting approaches are critical to developing adaptive strategies for mobility. Traffic patterns have complex spatial and temporal dependencies that make accurate forecasting on large highway networks a challenging task. Recently, diffusion convolutional recurrent neural networks (DCRNNs) have achieved state-of-the-art results in traffic forecasting by capturing the spatiotemporal dynamics of the traffic. Despite the promising results, however, applying DCRNNs for large highway networks still remains elusive because of computational and memory bottlenecks. This paper presents an approach for implementing a DCRNN for a large highway network that overcomes these limitations. This approach uses a graph-partitioning method to decompose a large highway network into smaller networks and trains them independently. The efficacy of the graph-partitioning-based DCRNN approach to model the traffic on a large California highway network with 11,160 sensor locations is demonstrated. An overlapping-nodes approach for the graph-partitioning-based DCRNN is developed to include sensor locations from partitions that are geographically close to a given partition. Furthermore, it is demonstrated that the DCRNN model can be used to forecast the speed and flow simultaneously and that the forecasted values preserve fundamental traffic flow dynamics. This approach to developing DCRNN models that represent large highway networks can be a potential core capability in advanced highway traffic monitoring systems, where a trained DCRNN model forecasting traffic at all sensor locations can be used to adjust traffic management strategies proactively based on anticipated future conditions.

In the United States (U.S.) alone, traffic congestion accounts for billions of dollars of economic loss because of productivity loss from additional travel time and additional inefficiencies and energy required for vehicle operation. To address these issues, intelligent transportation system (ITS) strategies seek to better manage and mitigate congestion and other traffic-related issues via a range of data-informed strategies and highway traffic monitoring systems (*1*). Near-term traffic forecasting is a foundational component of these strategies. Accurate forecasting across a range of normal, elevated, and extreme levels of congestion is critical for improved traffic control, routing optimization, and identification of novel approaches for handling emerging patterns of congestion (*2*,*3*). Furthermore, predictions and models from machine learning methods can be used to delve more deeply into the dynamics of a particular transportation network to identify additional areas of improvement beyond those enabled by improved prediction and control (*4*–*6*). Forecasting methodologies are also expected to enable new forms of ITS strategies as they become integrated into larger optimization and control

[1]Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL
[2]Leadership Computing Facility, Argonne National Laboratory, Lemont, IL
[3]Energy Systems Division, Argonne National Laboratory, Lemont, IL
[4]Sustainable Energy Systems Group, Lawrence Berkeley National Laboratory, Berkeley, CA

Corresponding Author:
Tanwi Mallick, tmallick@anl.gov

approaches and highway traffic monitoring systems (*7*,*8*). For example, the benefits of highly dynamic route guidance and alternative transit mode pricing in real time would be greatly aided by improved traffic forecasting.

Traffic forecasting is a challenging problem. Key traffic metrics, such as flow and speed, exhibit complex spatial and temporal correlations that are difficult to model with classical forecasting approaches (*9–12*). From the spatial perspective, locations that are close geographically in the Euclidean sense (e.g., two locations located in opposite directions of the same highway) often do not exhibit a similar traffic pattern, whereas locations in the highway network that are relatively far apart (e.g., two locations separated by a mile in the same direction of the same highway) can show strong correlations. Many traditional predictive modeling approaches cannot handle these types of correlation. From the temporal perspective, because traffic conditions vary across different locations (e.g., diverse peak hour patterns, varying traffic flow and volume, highway capacity, incidents, and interdependencies), the time series data becomes nonlinear and nonstationary, rendering many statistical time series modeling approaches ineffective.

Recently, machine learning, in particular deep learning (DL) approaches, have emerged as high-performing methods for traffic forecasting (*13*). Among these methods, the diffusion convolutional recurrent neural network (DCRNN) is a state-of-the-art method developed by Li et al. for short-term traffic forecasting (*14*). DCRNN models complex spatial dependencies using a diffusion process on a graph and temporal dependencies using a sequence to sequence recurrent neural network. Despite accurate forecasting results, however, modeling large highway networks with DCRNN still remains challenging because of the computational and memory bottlenecks. This paper focuses on developing and applying DCRNN to a large highway network. This study is motivated by the highway network of a state such as California being $\approx$30 times larger than the size of the highway networks for which DCRNN was originally developed and demonstrated. The two main scaling challenges are that (*1*) the training data size for thousands of locations is too large to fit in a single compute node memory, and (*2*) the time required for training a DCRNN on a large dataset can be prohibitive, rendering the method ineffective for large highway networks. Distributed data-parallel and model-parallel training approaches utilize multiple compute nodes to overcome these methods (*15*). In traditional computational science domains, a common approach for scaling is domain decomposition, wherein the problem is divided into several subproblems that are then distributed over different compute nodes. While domain decomposition approaches are not applicable in scaling typical DL training, such as image and text classification, they are well suited for the traffic forecasting problem with DCRNN because traffic flow in one region of the highway network does not affect another region when the regions are separated by a suitably large driving distance. To that end, a graph-partitioning-based DCRNN approach is proposed that partitions a large highway network into subnetworks and trains a DCRNN for each subnetwork independently. In contrast to distributed data-parallel and model-parallel training approaches, multiple compute nodes are not a prerequisite for this method because the independent DCRNN models can be trained sequentially on a single compute node. Consequently, this approach is more amenable for implementation within traffic management centers (TMCs) without the need to access cloud computing resources. On the other hand, given such multinode cloud/computing access, this method can provide significant benefit with respect to overall model training time. Furthermore, it is shown that the short-term forecasting accuracy can be improved by partitioning the highway network, which results in smaller models that are easier to train. The main contributions of this work are as follows.

1. The efficacy of the graph-partitioning-based DCRNN approach to model the traffic on a large California highway network with 11,160 sensor locations is demonstrated.
2. An overlapping-nodes approach—an improvement strategy for the graph-partitioning-based DCRNN that includes sensor locations from partitions that are geographically close to a given partition—is developed.
3. It is shown that DCRNN can be extended for multi-output learning to forecast both flow and speed simultaneously, as opposed to the previous DCRNN implementation that forecast either speed or flow.

Related Work

Modeling the flow and speed patterns of traffic in a highway network has been studied for decades. Capturing the spatiotemporal dependencies of the highway network is a crucial task for traffic forecasting. The methods for short-term traffic forecasting are broadly classified into two categories: knowledge-driven and data-driven approaches. In transportation and operational research, knowledge-driven methods usually apply queuing

theory, and Petri nets to simulate traffic behaviors (*16–20*). Usually, those approaches estimate the traffic flow of one intersection at a time. Traffic prediction for the full highway system of an entire state has not been attempted to date by using knowledge-driven approaches.

Data-driven approaches have received notable attention in recent years. The methods include statistical techniques such as autoregressive statistics for time series, and Kalman filtering techniques (*9*,*21*). These models are used mostly to forecast at a single sensor location and are based on a stationary assumption about the time series data. Therefore, they often fail to capture nonlinear temporal dependencies and cannot predict overall traffic in a large-scale network (*14*). Recently, machine learning methods on short-term traffic forecasting have emerged. More complex data modeling can be achieved by these models, such as support vector machines (SVMs) and artificial neural networks (ANNs) (*10–12*,*22*). However, SVMs are computationally expensive for large networks, and ANNs cannot capture the spatial dependencies of the traffic network. Furthermore, the shallow architecture of ANNs make the network less efficient compared with a deep learning architecture.

Recently, deep learning models such as deep belief networks and stacked autoencoders have been used to capture effective features for short-term traffic forecasting (*23*,*24*). Recurrent neural networks (RNNs) and their variants, long short-term memory (LSTM) networks and gated recurrent units (GRUs), show effective forecasting because of their ability to capture the temporal dependencies (*25–28*). However, spatial dynamics are often missed my RNN-based methods. To capture the spatial dynamics, researchers have used convolutional neural networks (CNNs). Ma et al. proposed an image-based traffic speed prediction method using CNNs, whereas Yu et al. proposed spatiotemporal recurrent convolutional networks for short-term traffic forecasting (*29*,*30*). Spatial dynamics have been captured by deep CNNs, and temporal dynamics have been learned by LSTM networks. In both, the highway network was represented as an image, and the link was colored by speed. The model was tested on only 278 links of the Beijing transportation network. Zhang et al. also represented the flow of crowds in a traffic network using grid-based Euclidean space (*31*,*32*). The temporal closeness, period, and trend of the traffic were modeled by using a residual neural network framework. The researchers evaluated the model on Beijing and New York City crowd flows. They used two datasets: trajectories of taxicab GPS data of four time intervals, and trajectories of NYC bikes during one interval of time. Trip data included trip duration, starting and ending sensor IDs, and start and end times. The key limitation of these approaches is that they do not capture non-Euclidean spatial connectivity. Du et al. proposed a model with one-dimensional CNNs and GRUs with the attention mechanism used to forecast traffic flow on UK traffic data (33). The contribution of this method is multimodal learning by multiple feature fusion (e.g., flow, speed, events, weather)on time series data of one year. This dataset was composed of 34,876 15-min intervals. However, it was limited to a narrow spatial dimension.

Recently, CNNs have been generalized from a 2D grid-based convolution to a graph-based convolution in non-Euclidean space. Yu et al. modeled the sensor network as an undirected graph and proposed a deep learning framework, called a spatiotemporal graph convolutional network, for speed forecasting (*34*). They applied graph convolution and gated temporal convolution through spatiotemporal convolutional blocks. The experiments were done on two datasets: BJER4, collected by the Beijing Municipal Traffic Commission, and PeMS (Performance Measurement System) D7, collected by the California Department of Transportation. The maximum size of their dataset was 1,026 sensors of California District 7. However, these spectral-based convolution methods require the graph to be undirected. Therefore, moving from a spectral-based to a vertex-based method, Atwood and Towsley first proposed convolution as a diffusion process across the node of the graph (*35*). Later, Hechtlinger et al. developed convolution to graphs by convolving every node and its closest neighbors selected by a random walk (*36*). However, none of these methods could capture the temporal dependencies.

Li et al. used the DCRNN method to forecast performances for 15, 30, and 60 min on two datasets: a Los Angeles dataset with 207 locations collected over 4 months, and a Bay Area dataset with 325 locations collected over 6 months (*14*). Their results showed improvement on the state-of-the-art baseline methods such as historical average, an autoregressive integrated moving average model with a Kalman filter, a vector autoregressive model, a linear support vector regression, a feed-forward neural network, and an encoder-decoder framework using LSTM (*9*,*37–40*).

The approach of this study differs from these works in several respects. None have addressed a large set of sensor locations, whereas this study used 11,160 sensor locations that cover the major part of the California highway system. Moreover, this graph-partitioning-based approach for large-scale traffic forecasting, using overlapping nodes, and multi-output multitask

forecasting, using DCRNN, significantly extends the initial DCRNN idea to real-world solutions.

Methodology

In this section, the DCRNN approach for traffic modeling is described, followed by graph partitioning for DCRNN, the overlapping node method, and multi-output learning.

Diffusion Convolutional Recurrent Neural Network (DCRNN)

Formally, the problem of short-term traffic forecasting can be modeled as a spatial-temporal time series forecast defined on a weighted directed graph $G = (V, \varepsilon, A)$, where V is a set of N nodes that represent sensor locations, ϵ is the set of edges connecting the sensor locations, and $A \in R^{N \times N}$ is the weighted adjacency matrix that represents the connectivity between the nodes in relation to highway network distance. Given the graph G and the time series data $X_{t-T'+1}$ to X_t, the goal of the traffic forecasting problem is to learn a function h(.) that maps historical data of T' time steps at given time t to future T time steps.

$$X_{t-T'+1}, \ldots, X_t; G \overset{\text{h(.)}}{\rightarrow} X_{t+1}, \ldots, X_{t+T}$$

In DCRNN, the temporal dependency of the historical data has been captured by the encoder-decoder architecture of a recurrent neural network (*40*,*41*). The encoder steps through the input historical time series data and encodes the entire sequence into a fixed-length vector. The decoder predicts the output of the next T time steps while reading from the vector. GRU is used to design the encoder-decoder architecture (*41*). Inside the encoder-decoder architecture of RNN, a diffusion convolution operation is used to capture the spatial dependencies. The diffusion process can be described by a random walk on G (*42*). The traffic flow from one node to the neighbor nodes can be represented as a weighted combination of infinite random walks on the graph. The matrix multiplications of RNN is replaced with the diffusion convolution operation to make the DCRNN cell, which is defined as

$$\begin{aligned} r^t &= \sigma(W_{r*G}[X_t, h_{t-1}] + b_r) \\ u^t &= \sigma(W_{u*G}[X_t, h_{t-1}] + b_u) \\ c^t &= \tanh(W_{c*G}[X_t(r_t \odot h_{t-1}] + b_c) \\ h_t &= u_t \odot h_{t-1} + (1 - u_t) \odot c_t, \end{aligned}$$

where X_t and h_t denote the input and final state at time t, respectively; r_t, u_t, and c_t are the reset gate, update gate, and cell state at time t, respectively; $*G$ denotes the diffusion convolution operation; and W_r, W_u, and W_c are parameters for the corresponding filters. The diffusion convolution operation ($*G$) over the input graph signal X is defined as

$$W_{*G}X = \sum_{d=0}^{K-1} (W_O(D_O^{-1}A)^d + W_I(D_I^{-1}A)^d)X$$

where K is a maximum diffusion steps; $D_O^{-1}A$ and $D_I^{-1}A$ are transition matrices of the diffusion process and the reverse one, respectively; D_O and D_I are the in-degree and out-degree diagonal matrices, respectively; and W_O and W_I are the learnable filters for the bidirectional diffusion process. The in-degree and out-degree diagonal matrices provide the capability to capture the effect of the upstream as well as the downstream traffic.

During the training phase, the historical time series data and the graph are fed into the encoder, and the final stage of the encoder is used to initialize the decoder. The decoder predicts the output of the next T time steps, and the layers of DCRNN are trained by using back-propagation through time. During the test, the ground truth observations are replaced by previously predicted output. The discrepancy between the input distributions of training and testing can cause performance degradation. To resolve this issue, scheduled sampling has been used, where the model is fed a ground truth observation with a probability of ϵ_i or the prediction by the model with a probability of $1 - \epsilon_i$ at the ith iteration (*43*). The model is trained with a mean absolute error (MAE) loss function, defined as

$$\text{MAE} = \frac{1}{s}\sum_{i=1}^{s} |y_i - \hat{y}_i|,$$

where y_i is the observed value, $\hat{y}_i$ corresponds to the forecasted values for the ith training data, and s denotes the number of samples. See Li et al. for a detailed exposition on DCRNN (*14*).

Graph-Partitioning-Based DCRNN

For the graph-partitioning-based DCRNN approach, the large graph G is partitioned into k subgraphs such as $G = \{G_0, \ldots, G_{k-1}\} = [\{V_0, \varepsilon_0\}, \{V_2, \varepsilon_2\}, \ldots, \{V_{k-1}, \varepsilon_{k-1}\}]$, where each ε_i consists only of the edges between nodes in V_i. Each subgraph has a set of historical time series data $X = \{X_0, \ldots, X_{k-1}\}$, and the adjacency matrix $A = [A_0, \ldots, A_k]$ is calculated. Now, each subgraph can be trained independently.

For selecting the graph partitions, various graph clustering and community detection methods have been

developed, such as multilevel k-way graph partitioning, multilevel recursive bisection, spectral clustering, and Louvain (*44–48*). The different clustering methods were compared and it was found that multilevel k-way partitioning has several advantages over all other methods (see the Supporting Experiments, Impact of graph partitioning methods section, for the comparison). This method can partition a million-node graph in a few seconds with tightly connected clusters. It takes the adjacency matrix as input and divides it into multiple partitions in three phases: (*1*) coarsening phase—the graph is coarsened down to a smaller graph with fewer vertices and a set of vertices is collapsed iteratively to form a single vertex; (*2*) initial partitioning phase—the smaller coarsened graph is partitioned by using multilevel k-way partitioning, where k is the number of partitions (*45*); and (*3*) uncoarsening phase—the partitions are projected back to the original graph by backtracking through the coarsened graph. To get tightly connected partitions, nodes are swapped between partitions by using the Kernighan-Lin algorithm (49).

An example of applying the k-way partitioning algorithm on the Los Angeles region (2,036 sensors) is shown in Figure 1. Here, the graph with 2,036 nodes was partitioned into eight parts by using the adjacency matrix as input to the partitioning algorithm and the partitions were rendered in different colors. Each partition has approximately 254 nodes.

Overlapping Nodes

In Figure 1, it is observed that nodes at the boundary of any partition lose their neighboring correlated nodes to nearby partitions despite the shorter driving distance. This issue will become critical and affect the prediction accuracy of DCRNN when the number of partitions increases. To address this issue, an overlapping-nodes approach is developed wherein, for each partition, spatially correlated nodes from other partitions are found and included. Consequently, the nodes that are near the boundary of the partition will appear in more than one partition.

A naive approach for finding the correlated nodes consists of computing nearest neighbors for each node in the partition based on the driving distance, and excluding the nodes already included in the partition. The disadvantage of this approach is that it can include, for a given node, several spatially correlated nodes that are close to each other. This can lead to an increase in the number of nodes per partition and, consequently, higher training time and memory requirements. This issue is illustrated on a partition of the San Joaquin area,

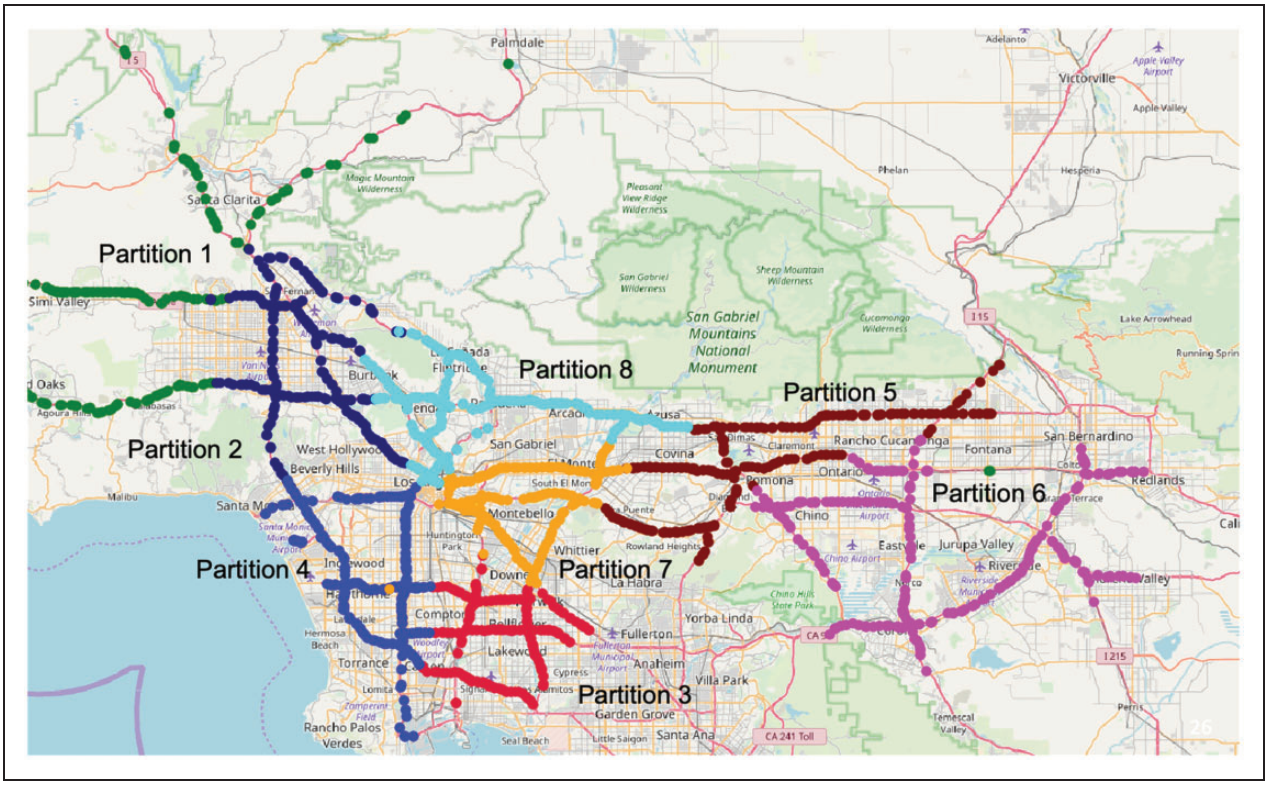

Figure 1. Result of k-way partitioning on Los Angeles region: 2,036 sensor locations are partitioned into eight parts, with each part rendered in different colors.

District 10 in Figure 2, where Figure 2*a* shows a small partition with 11 nodes rendered in red cross symbols, and Figure 2*b* shows the same partition after adding the 162 overlapping nodes rendered in black dot symbols. After adding the overlapping nodes, the total number of nodes become 15 times more than the original partition. Therefore, the spatially correlated nodes are downsampled from other partitions as follows: given two spatially correlated overlapping nodes from a different partition, only one is selected and the other is removed if they are within D' driving distance miles, where D' is a parameter.

Figure 2*c* shows the results after downsampling with a distance threshold of 1 mi. The total number of nodes becomes 40 after after downsampling.

Multi-Output Forecasting with a Single Model

Previously, DCRNN was used to forecast speed based on historical speed data. The input and output layers of the DCRNN are customized for multi-output forecasting such that a single DCRNN model can be trained and used for forecasting speed and flow simultaneously. The three key modifications were needed for multi-output forecasting: (*1*) Normalization of speed and flow: to bring speed and flow to the same scale, normalization was done separately on the two features using the standard scalar transformation. The normalized values of speed are given by $z_{sp} = \frac{x_{sp} - \mu_{sp}}{\sigma_{sp}}$, where μ_{sp} is the mean and σ_{sp} is the standard deviation of the speed values x_{sp}. The same method was applied for normalizing the flow values ($z_{fl} = \frac{x_{fl} - \mu_{fl}}{\sigma_{fl}}$, where μ_{fl} and σ_{fl} are the standard deviation of the flow values x_{fl}). An inverse transformation was applied to the normalized speed and flow forecasting values to transform them to the original scale (for computing error on the test data); (*2*) Multiple output layers in the DCRNN: in the previous study of DCRNN, the diffusion convolution layer learns P-dimensional input, such as speed and flow, and predicts Q-dimensional output, such as speed and flow; (*3*) Loss function: for multi-output training, a loss function was used of the form

$$\begin{aligned} \text{MAE}_{\text{multi}} &= \text{MAE}_{sp} + \text{MAE}_{fl} \\ &= \frac{1}{s}\sum_{i=1}^{s}|y_{sp_i} - \hat{y}_{sp_i}| + \frac{1}{s}\sum_{i=1}^{s}|y_{fl_i} - \hat{y}_{fl_i}|, \end{aligned}$$

where y_{sp_i} and y_{fl_i} are observed speed and flow values, respectively; $\hat{y}_{sp_i}$ and $\hat{y}_{fl_i}$ are corresponding forecast values, respectively, for the ith training data; and s is the total number of training points.

Experimental Results

First, the dataset of the California highway network used for these experiments is described. Next, the efficacy of the DCRNN model is evaluated with respect to the number of partitions, and the model errors are analyzed using a sensitivity analysis approach. Then, the impact of overlapping nodes and of hyperparameter search on the DCRNN accuracy are studied. The section is concluded with a demonstration of multi-output forecasting using a graph-partitioning-based DCRNN.

Dataset: California Highway Network

This approach is evaluated on the California highway network. Data from PeMS is used, which provides access to real-time and historical performance data

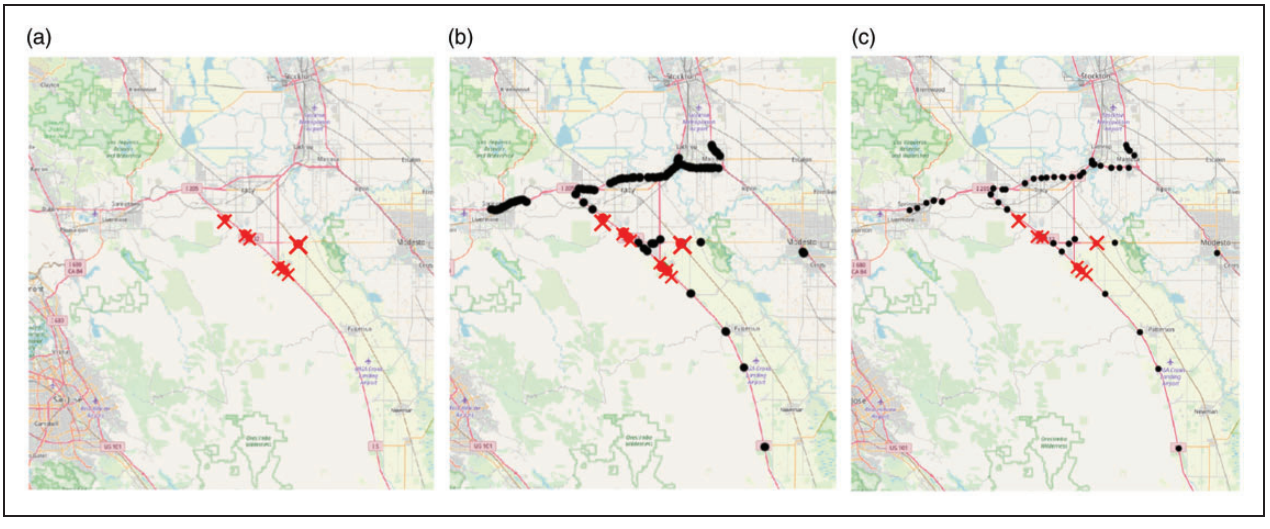

Figure 2. Example after adding overlapping nodes: (*a*) partition without overlapping nodes, (*b*) partition after adding overlapping nodes, (*c*) partition after downsampling the overlapping nodes with distance threshold of 1 mi.

from over 39,000 individual sensors (*50*). The individual sensors placed on the different highways are aggregated across several lanes and are fed into vehicle detector stations. Vehicle detector stations include a variety of sensors such as inductive loops, radar, and magnetometers. The sensors may be located on high-occupancy vehicle (HOV) lanes, mainlines, on-ramps, and off-ramps. The dataset covers nine districts of California—D3 (North Central) with 1,212 stations, D4 (Bay Area) with 3,880 stations, D5 (Central Coast) with 382 stations, D6 (South Central) with 624 stations, D7 (Los Angeles) with 4,864 stations, D8 (San Bernardino) with 2,115 stations, D10 (Central) with 1,195 stations, D11 (San Diego) with 1,502 stations, and D12 (Orange County) with 2,539 stations—giving a total of 18,313 stations listed by location. Detectors capture samples every 30 s. PeMS then aggregates that data to the granularity of 5 min, an hour, and a day. The data include timestamp, station ID, district, freeway, direction of travel, total flow, and average speed (mph).

PeMS does not list the latitude and longitude for the stations IDs, which are essential for defining the connectivity matrix used by DCRNN. Instead, the latitude and longitude are associated with postmile markers of every freeway given the direction. The latitude and longitude for sensor IDs were found by matching the absolute postmile markers of every freeway. Linear interpolation was used to find the exact latitude and longitude, if the absolute postmile markers did not match exactly.

The official PeMS website shows that 69.59% of the ≈18K stations are in good working condition. The remaining 30.41% do not capture time series data throughout the year and therefore were excluded from the dataset. The final dataset has speed and flow of 11,160 stations from January 1, 2018, to December 31, 2018, with a granularity of every 5 min.

It was observed that the flow and speed values are missing for multiple time periods in the time series data. The percentage of missing data is small—for speed 0.06% (698,162 out of 1,173,139,200 data points) and for flow 0.04% (504,688 out of 1,173,139,200 data points). Three different imputation techniques were explored—temporal mean, temporal median, and linear interpolation—and no significant difference was found because the percentage of missing data is less than 1%. The details of the experiments and results are reported in the Supporting Experiments, Missing data imputation, section. Therefore, the missing data was replaced by the temporal mean.

Experimental Setup

The highway network of 11,160 sensors was represented as a weighted directed graph. From the 1-year data, the first 70% of the data (36 weeks approx.) was used for training and the next 10% (5 weeks approx.) and 20% (10 weeks approx.) of the data was used for validation and testing, respectively. Different training data sizes were experimented with by selecting 2, 4, 12, 20, and 36 weeks of data and it was found that using 36 weeks results in better accuracy. Given 60 min of time series data on the nodes in the graph, a forecast was made for the next 60 min. The dataset was prepared in a way to look back (T') for 60 min or 12 time steps (granularity of the data is 5 min) to predict (T) the next 60 min or 12 time steps. The T' window slides by 5 min or one time step and repeats until all the available data is consumed. The forecasting performance of the models was evaluated on the test data by using MAE.

The Open Source Routing Machine (OSRM) was used to compute the highway network distance between the nodes, which is required by the adjacency matrix for DCRNN (*51*). Given the latitude and longitude of two nodes, a locally running OSRM gives the shortest driving distance between them using OpenStreetMap (*52*). To speed up the computation, first, 30 nearest neighbors for each node were found using the Euclidean distance, and then the OSRM queries were limited only to the nearest neighbors. As in the original DCRNN work, the adjacency matrix was built using a thresholded Gaussian kernel: $A_{ij} = \exp(-\frac{dist(v_i,v_j)^2}{\sigma^2})$ *if* $dist(v_i, v_j)^2 \leq$ *thresh*, otherwise 0, where A_{ij} represents the edge weight between node v_i and node v_j; $dist(v_i, v_j)$ denotes the highway network distance from node v_i to node v_j; σ is the standard deviation of distances; and *thresh* is the threshold, which introduces the sparsity in the adjacency matrix (*53*).

For the experimental evaluation, a GPU-based cluster at the Argonne Leadership Computing Facility was used. It has 126 compute nodes, where each node consists of two 2.4 GHz Intel Haswell E5-2620 v3 processors (six cores per CPU, 12 cores total), one NVIDIA Tesla K80 (two GPUs per node), 384 GB RAM per node, and 24 GB GPU RAM per node (12 GB per GPU). Python 3.6.0, TensorFlow 1.3.1 was used, and Metis 5.1.0. DCRNN code of Li et al., available on GitHub, was customized for this implementation (*14*,*54*). Given k partitions of the highway network, partition-specific DCRNNs were trained simultaneously on GPU nodes. Two MPI ranks were used per node, where each rank ran a partition-specific DCRNN using one GPU. The input data for different partitions (time series and adjacency matrix of the graph) was prepared offline and loaded into the partition-specific DCRNN before the training started. It was noted that such simultaneous training is not required, in particular when a multi-GPU cluster is not available. The training on a single GPU would consist of running the training for each partition sequentially.

The default hyperparameter configuration values for the DCRNN were used: batch size—64; filter type—random walk (used to model the stochastic nature of highway traffic); number of diffusion steps—2; RNN layers—2; RNN units per layer—16; threshold for gradient clipping—5; initial learning rate—0.01; and learning rate decay—0.1. The model was trained by minimizing MAE using the Adam optimizer (*55*).

Number of Partitions

Here, the efficacy of the graph-partitioning-based DCRNN is demonstrated.

The multilevel k-way graph-partitioning method from the Metis software package was used to create 2, 4, 8, 16, 32, 64, and 128 partitions of the California highway network graph (*56*). The average number of nodes in each cases was 5,580, 2,790, 1395, 697, 348, 174, and 87, respectively. Partitions of size 1, the full network, and 2 are not presented because the training data was too large to fit in the memory of a single K80 GPU node. Given k partitions, $k/2$ nodes (or k GPUs) are used to run the partition-specific DCRNNs simultaneously. The training time is defined as the maximum time taken by any partition-specific DCRNN training, excluding the data loading time.

Figure 3 shows the distribution of MAE of all 11,160 nodes obtained by using box-and-whisker plots. From the results it can be observed that medians, 75% quantiles, and the maximum MAE values show a trend in which an increase in the number of partitions decreases the MAE. From 4 to 64 partitions, the median of MAE decreases from 2.11 to 2.02. The increase in accuracy can be attributed to two factors: (*1*) the effectiveness of the k-way graph partitioning of Metis that separates sensor locations that were far apart with respect to driving distance, and (*2*) the model training for each partition becomes relatively easier as the number of nodes in a given partition decreases. For 128 partitions (with only 87 nodes per partition), the observed MAE values are higher than for 64 partitions. The reason is that the graph partition results in a significant number of spatially correlated nodes ending up in different partitions. This can be assumed as a tipping point for graph partitioning, which relates to the size and spread of the actual network.

Figure 4 shows the training time required for different numbers of partitions. The time decreases significantly with an increase in the number of partitions. In addition, This approach reduces the training time from 2,820 min on 4 partitions (= 4 GPUs) to 178.67 min on 64 partitions (= 64 GPUs), resulting in a 15.78x speedup. There is an almost linear speedup until 64 partitions, where doubling the number of partitions (and GPUs) results in ≈2X speedup. However, the speedup gains drop significantly with 128 nodes. At this point, with only 87 nodes per partition there is not enough workload for the GPU.

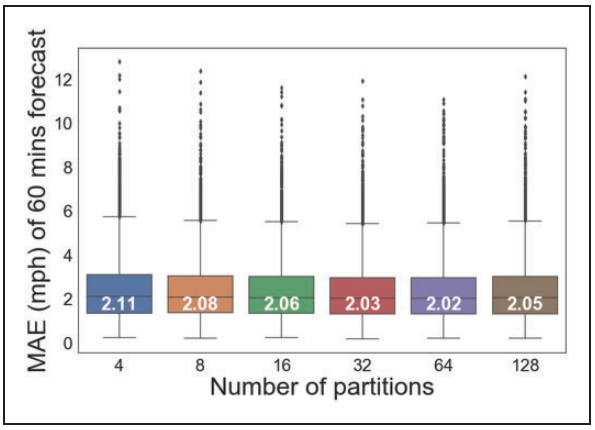

Figure 3. Distribution of mean absolute error (MAE) for different number of partitions.

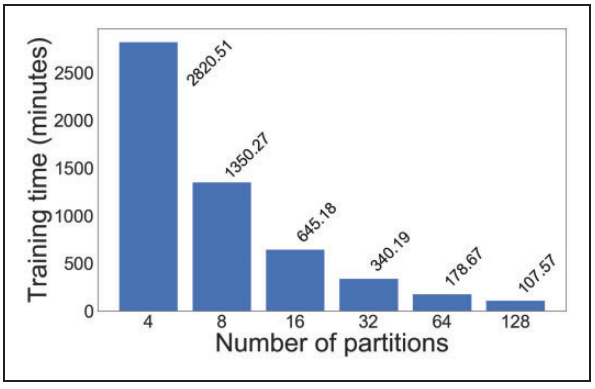

Figure 4. Training time for diffusion convolutional recurrent neural networks (DCRNNs) with different numbers of partitions.

Since the best forecasting accuracy and speedup were obtained by using 64 partitions, it was used as a default number of partitions in the rest of the experiments.

Error Analysis

Figure 3 shows several outliers (large errors) in the MAE distribution. Here, using a decision tree method for sensitivity analysis, the factors that led to these large errors were investigated.

The factors studied are (*1*) sensor type (loop detector, magnetometers, etc.), (*2*) district where the sensor is located (Los Angeles, Bay Area, etc.), (*3*) lane type (mainline, HOV, etc.), and (*4*) traffic dynamics at a given sensor location measured by the coefficient of variation (standard deviation (σ)/mean (μ) of the time series data). The coefficient of variation becomes large (small) when the variations in the speed values are large (small).

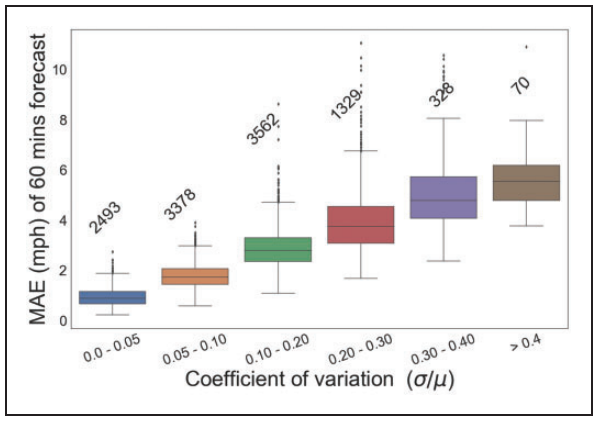

Figure 5. Impact of the coefficient of variation of the data on the forecasting accuracy. The coefficient of variation of 11,160 sensors is binned into six categories. The number of nodes in each distribution is shown above each box.
Note: MAE = mean absolute error.

For each node, the values of the four factors are used as input (independent variables), and the corresponding forecasting error values (MAE) from 64-partition DCRNN are used as output (dependent variables). The MAE values are separated into four classes: class 0 if MAE is less than 1, class 1 if MAE is between 1 and 3, class 2 if MAE is between 3 and 5, and class 3 if MAE is greater than 5. Consequently, for each node there is an input-output pair; the dataset comprises the pair from all the nodes.

Next, a decision tree was trained from the scikit-learn package, a supervised machine learning method, to model the error classes as a function of the four factors (*57*). The decision tree was selected because of the model interpretability, which can be used for analyzing the factors under study. 80% of the data was used for training and 20% for testing. To avoid overfitting, the depth of the tree was set to 8; all other values were set to the default as in scikit-learn decision tree interface. The decision tree model obtained 82.11% and 78.18% accuracy on the training and testing sets, respectively. The high accuracy indicates that the error classes can be modeled and explained through the four factors.

The trained decision tree model provides a normalized importance score (between 0 and 1) for each factor. The values for traffic dynamics, district, sensor type, and lane type were 0.87, 0.07, 0.04, and 0.02, respectively. These results show that the traffic dynamics has the most significant impact on the forecasting error and that the other factors do not have impact. Figure 5 shows the distribution of MAE values with respect to the coefficient of variation. A clear trend is observed in which the median (and other quantiles) of the MAE distribution increases linearly with respect to the coefficient of variation.

To take a closer look at the sensors with high errors, class 3 (MAE $>$ 5) was analyzed. This class had 449 sensors, approximately 4% of the total data (449 out of 11,160 sensors). Of these 449 sensors, 244 were from D7 (Los Angeles), 85 from D4 (Bay Area), 42 from D8 (San Bernardino), and 36 district D12 (Orange County). Given the forecasting horizon of 60 min, forecasting becomes difficult for the nodes with a high coefficient of variation. For these nodes, the error decreases with a decrease in the forecasting horizon. For the 30- and 15-min forecast horizons, only 125 and 36 sensors have MAE values that are greater 5 mph. From these results, it is inferred that for the nodes with high traffic dynamics, good forecasting accuracy can be achieved by reducing the forecasting horizon to 15 min.

Impact of Overlapping Nodes

Here, the impact of overlapping nodes on the forecasting accuracy of the graph-partitioning-based DCRNN is discussed.

Table 1 describes four different experiments. The graph-partitioning-based DCRNN was trained on 64 partitions of the California highway network. This variant is referred to as DCRNN_64_naive. Training time took 178 min. After training, the speed was forecast for 60 min on the test data and the MAE was calculated for each node. The results are summarized in the first row of Table 1 and the 64 partition results in Figure 6. It is observed that the MAE values of 1,716, 6,729, 2,266, and 449 nodes are less than 1, between 1 and 3, between 3 and 5, and greater than 5, respectively.

The graph-partitioning-based DCRNN was also trained on 64 partitions with overlapping nodes. Nodes were downsampled with different distance threshold (D') values: 0.5 mi, 1 mi, 1.5 mi, 2 mi, and 3 mi. The results showed no significant improvement beyond the 1 mi threshold; therefore, 1 mi was used as the distance threshold for the experiments. In a given partition, while calculating the MAE value for each node, the overlapping nodes were not considered because they are included in a different partition, where their MAE values will be computed. This variant is referred to as DCRNN_64_overlap. The results are shown in row 2 of Table 1. It is observed that DCRNN_64_overlap outperforms DCRNN_64_naive. With reference to the latter, the number of nodes with MAE values less than 1 has increased from 1,716 to 1,837; similarly, the number of nodes with MAE values between 1 and 3, 3 and 5, and greater than 5 decreased from 6,729 to 6,687, from 2,266 to 2,204, and from 449 to 432, respectively. It is observed that the training time increased from 178.67 min to 221.04 min, which can be attributed to the increase in the number of nodes per partition.

Table 1. Results of Graph-Partitioning-Based Diffusion Convolutional Recurrent Neural Network (DCRNN) with Overlapping Nodes and Hyperparameter Search

	MAE <1	1 <= MAE <3	3 <= MAE <5	MAE = >5	Trainable parameters	Training time (minutes)	Forecast time (minutes)
1. DCRNN_64_naive	1,716	6,729	2,266	449	14,608	178.67	4.38
2. DCRNN_64_overlap	1,837	6,687	2,204	432	14,608	221.04	4.88
3. DCRNN_64_naive_hps	1,920	6,897	1,980	363	19,808	287.05	4.92
4. DCRNN_64_overlap_hps	1,897	6,940	1,972	351	38,048	461.57	5.83

Note: MAE = mean absolute error.

Earlier, it was noted that the forecasting accuracy decreased after increasing the number of partitions from 64 to 128, as a significant number of spatially correlated nodes ended up in different partitions. It is hypothesized that including overlapping nodes can improve forecasting accuracy significantly for those partitions. Therefore, to analyze the impact of the overlapping nodes in different partition scales, 512 and 1,024 partitions were considered where each graph partition contains approximately 21 and 11 nodes, respectively. The overlapping nodes were added to each partition by downsampling them with a 1-mi distance threshold. The graph-partitioning-based DCRNN model was then trained and tested for 64, 128, 512, and 1,024 partitions with and without overlapping nodes. Figure 6 shows the distribution of MAE values at each scale with and without overlapping nodes. The median of MAE decreases from 2.02 to 2.01 for 64 partitions, from 2.05 to 2.03 for 128 partitions, from 2.15 to 2.00 for 512 partitions, and from 2.22 to 2.04 for 1,024 partitions. These results show that the impact of overlapping nodes becomes more significant when the subgraph becomes very small and spatially correlated neighboring nodes belong to different partitions.

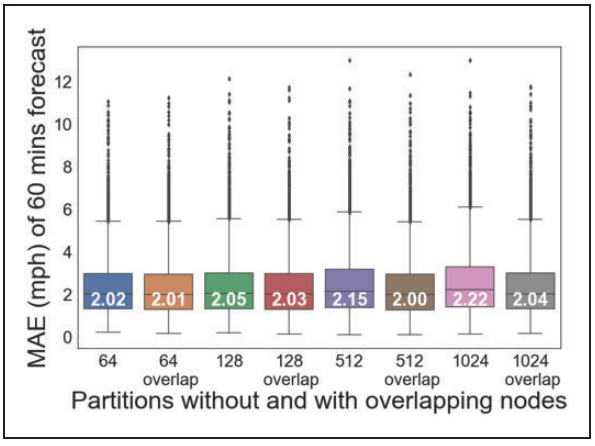

Figure 6. Impact of overlapping nodes in different partition scales.
Note: MAE = mean absolute error.

Impact of Hyperparameter Tuning

Here, the impact of hyperparameter search on the forecasting accuracy of the graph-partitioning-based DCRNN is investigated.

The hyperparameters that can impact the forecasting accuracy of the DCRNN include batch size, filter type (i.e., random walk, Laplacian), maximum diffusion steps, number of RNN layers, number of RNN units per layers, a threshold max_grad_norm to clip the gradient norm to avoid exploring gradient problem of RNN, initial learning rate, and learning rate decay (*58*).

DeepHyper, a scalable hyperparameter search (HPS) package for neural networks, was used to search for high-performing hyperparameter values for DCRNN_64_naive and DCRNN_64_overlap (*59*). Five months of data (from May 2018 to October 2018) from partition 1 were used. As the stopping criteria, 32 nodes with a 12-h wall-clock time were used. DeepHyper sampled 518 and 478 hyperparameter configurations for naive and overlapping approaches, respectively. The best hyperparameter configurations were selected from each and used to train the models and infer the forecasting accuracy. These two variants are referred to as DCRNN_64_naive_hps and DCRNN_64_overlap_hps. The results are shown in rows 3 and 4 of Table 1. It is observed that DCRNN_64_naive_hps outperforms DCRNN_64_naive, where hyperparameter tuning improved the accuracy of several nodes. The number of nodes with MAE values less than 1 and between 1 and 3 increased from 1,716 to 1,920 and from 6,729 to 6,897, respectively. The number of nodes with MAE values between 3 and 5, and greater than 5 decreased from 2,266 to 1,980 and from 449 to 363, respectively. A similar trend is also observed with the comparison of DCRNN_64_overlap and DCRNN_64_overlap_hps. In particular, the number of nodes with MAE values between 3 and 5, and greater than 5 decrease from 2,204 to 1,972 and from 432 to 351, respectively. Moreover, hyperparameter tuning resulted in an increase in the number of trainable parameters, which led to an increase in training time from 221.04 min to 461.57 min.

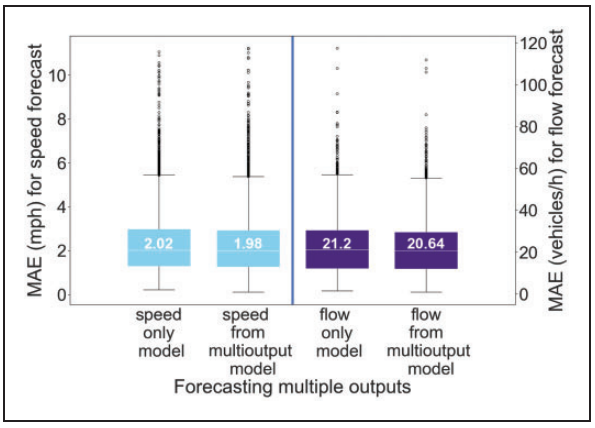

Figure 7. Box plot distribution of mean absolute error (MAE) for speed and flow forecasting. From left to right the box plots show the results of: speed forecasting from speed only model, speed forecasting from multi-output model, flow forecasting from flow only model, and flow forecasting from multi-output model.

This is because with overlapping nodes, DCRNN models already reach high accuracy; further improvements with hyperparameter search require a significant increase in the number of trainable parameters, which increases the training time. No significant difference was observed in the time required for forecasting by the different trained models on the test data. An exception is the DCRNN_64_overlap_hps case, where the large number of additional trainable parameters increases the forecasting time by 1 min (5.83 min).

Next, it was tested whether hyperparameter search on individual partitions can improve the accuracy further. Two partitions were selected (partition 38 from the Bay Area with 179 sensors and partition 62 from South Central with 184 sensors) from 64 partitions, and a hyperparameter search was run using the same setup. The best hyperparameter configurations were selected for each partition and used to train and infer the forecasting accuracy on the same partition. The results of

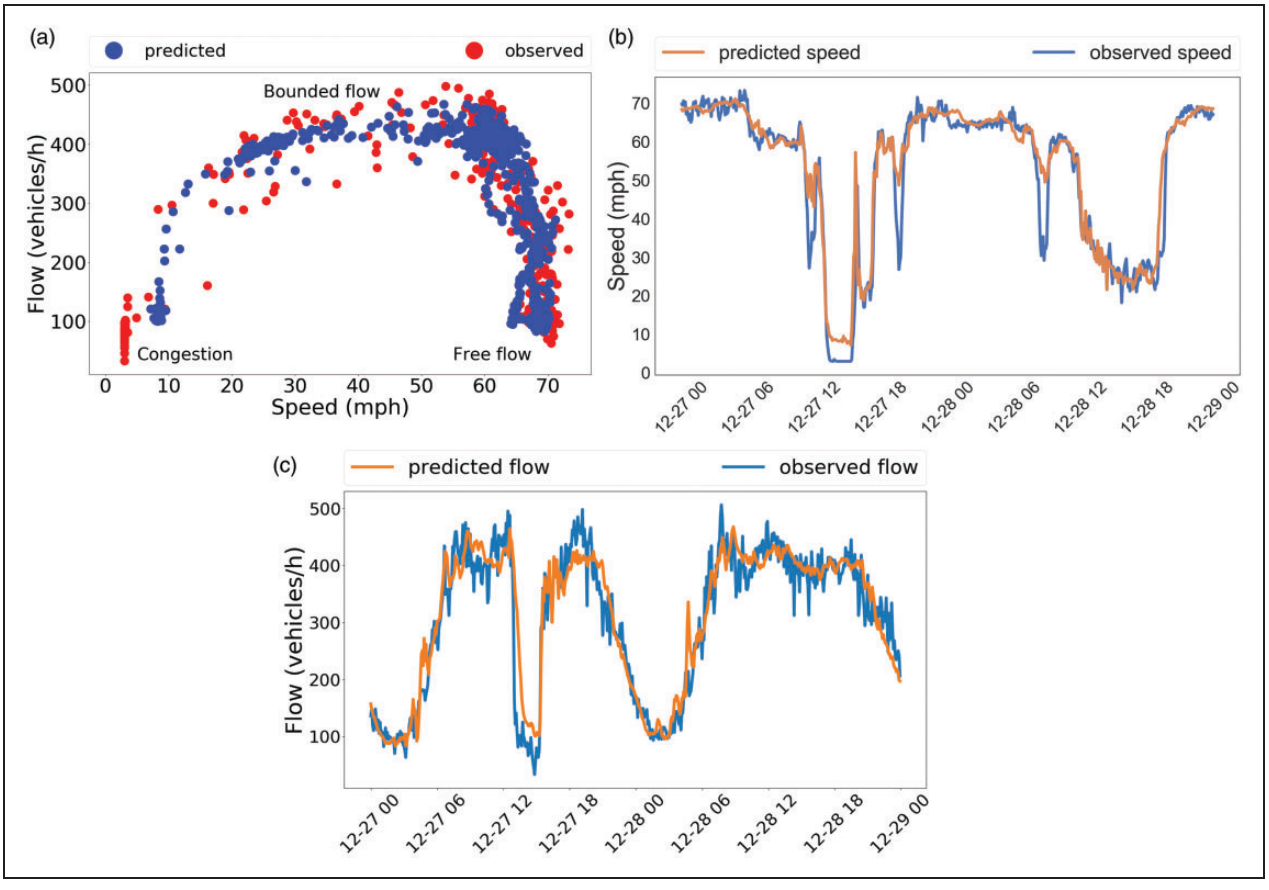

Figure 8. Fundamental traffic flow diagram and corresponding speed and flow forecasting results: (*a*) Closeness of the predicted flow and speed on sensor 717322 with fundamental traffic flow diagram. (*b*) Speed forecasting on sensor 717322 used to estimate the traffic flow diagram of Figure 8*a*. (*c*) Flow forecasting on sensor 717322 used to estimate the traffic flow diagram of Figure 8*a*.

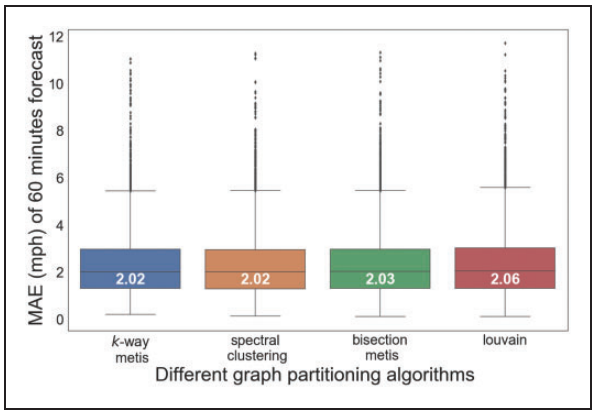

Figure 9. Impact of different graph-partitioning methods on forecasting error of the graph-partition-based diffusion convolutional recurrent neural network (DCRNN) on 64 partitions.
Note: MAE = mean absolute error.

hyperparameter tuning on individual partitions were compared with results of DCRNN_64_naive_hps on the same partitions. For partition 38, the number of nodes with MAE values less than 1, between 1 and 3, and between 3 to 5 increased from 5 to 6, from 118 to 119, and from 38 to 41, respectively. The number of nodes with MAE values greater than 5 decreased from 18 to 13. Similarly, for partition 62, the number of nodes with MAE values less than 1 and between 1 and 3 increased from 46 to 48 and from 121 to 123, respectively. The number of nodes with MAE values between 3 to 5 decreased from 13 to 9; nodes with MAE values greater than 5 had no effect and remained at 4. The number of trainable parameters for partition 38 and 62 were 19,520 and 37,856, respectively. The results show that the hyperparameter search on individual partitions can improve the accuracy slightly but not significantly.

Multi-Output Forecasting

The graph-partitioning-based DCRNN model was trained to forecast both speed and flow simultaneously and the results were compared with those of models that predict either speed or flow. Figure 7 shows the distribution of MAE values. The median of MAE from the speed-only model (first box plot) is 2.02, which was reduced to 1.98 with the multi-output model (second box plot). Similarly, the median of MAE from the flow-only model (third box plot) is 21.20, which was reduced to 20.64 with the multi-output model (fourth box plot). It was found that the multi-output model obtains MAE values that are significantly lower than those of the speed-only or flow-only models (a paired t-test $p-$values of 9.20×10^{-4} for speed and 5.77×10^{-5} for flow). The superior performance of multi-output forecasting can be attributed to multitask learning (*60*). The key advantage is that it leverages the commonalities across speed and flow learning tasks, which are related. This results in improved learning efficiency and consequently improved forecasting accuracy when compared with training the models separately.

Figure 8*a* shows speed and flow forecasting results of a congested node (ID: 717322 located on Highway 60-E in the Los Angeles area) in a scatter plot. It can be seen that the speed and flow forecast values closely follow the fundamental flow diagram with three distinct phases: congestion, bounded, and free flow. This forecasting pattern of DCRNN shows that the model has learned and preserved the properties of the traffic flow. Figure 8, *b* and *c*, shows the time-varying speed and flow corresponding to Figure 8*a* for the same sensor 717322.

Conclusion and Future Work

A graph-partitioning approach was developed to divide a large highway network into several partitions and a DCRNN model was trained for each of the partitions independently. An overlapping-nodes improvement strategy was implemented that includes data from partitions that are geographically close to a given partition. It was shown that DCRNN can be extended for multi-output learning to forecast both flow and speed simultaneously as opposed to a previous DCRNN implementation that predicted either speed or flow. The effectiveness of the proposed approach was demonstrated using Caltrans PeMS data to model the traffic on a large California highway network with 11,160 sensor locations. The model error was analyzed and it was shown that higher traffic dynamics caused by rapid changes in traffic behavior lead to high forecasting error. It was also shown that including overlapping region can be more impactful when the partition size become small and spatially correlated nodes belong to different partitions.

The DCRNN model, once trained, can be run on traditional hardware such as CPUs for forecasting without the need for multiple GPUs and could be readily integrated into a traffic management center. Once integrated into such a center, the scale and accuracy of the forecasting techniques discussed in this work have the potential to enable more proactive decision-making as well as better decisions themselves given the capability to make large-scale and accurate forecasts in relation to future traffic states.

The authors plan to extend the approach to large-scale traffic forecasting with mobile device data. The goal will be to determine whether mobile device data can act as a proxy for inductive loop data, which could be used either to substitute for poorly working loops, or to extend the scope of the monitoring to areas where

loops would be prohibitively expensive. It is also planned to combine DCRNN with large-scale simulations to integrate realistic speed and flow forecasts into active traffic management decision algorithms. Furthermore, it is planned to develop models for route and policy scenario evaluation in adaptive traffic routing and management studies.

Supporting Experiments

Several supporting experiments are detailed in this section.

Missing Data Imputation

Three data imputation methods were studied: (*1*) temporal median, which takes the median of similar time and day of the week over a period, temporal mean, which takes the mean of similar time and day of the week over a period of time to fill the missing value, and (*3*) linear interpolation. These methods were evaluated on a partition from the D10 Central area with 180 nodes, where 0.054% of data was missing. Three datasets were created using these imputation methods and the model was trained. For training and testing data, ≈36 weeks of time series data from January 1, 2018, to September 13, 2018, and ≈10 weeks of time series data from October 20, 2018, to December 31, 2018, were used. It was found that the median of MAE distribution is 1.77 for the temporal mean and interpolation and 1.78 for the temporal median. The results show that missing data imputation does not have a significant impact on the forecasting accuracy. This can be attributed to the small percentage of the missing data (less than 1%). Therefore, the missing data were replaced by temporal mean. Weekends are handled separately from normal working days.

Impact of Graph-Partitioning Methods

The impact of different graph-partitioning approaches was analyzed by comparing the default multilevel k-way graph partitioning approach with spectral clustering, multilevel recursive bisection, and Louvain methods (*45–48*). For the multilevel k-way graph-partitioning and multilevel recursive bisection approaches, the Metis software package was used (*56*). For spectral clustering, the sklearn implementation was used (*61*). For Louvain, python-louvain GitHub implementation was used (*62*). Multilevel k-way graph partitioning, spectral clustering, and multilevel recursive bisection methods take the adjacency matrix and number of partitions (k) as input and produce k partitions as output. The number of partitions k was set to 64 for each of the three methods. In contrast to three methods, Louvain takes the adjacency matrix as input and produces an optimized number of partitions as output, in this case 110 partitions.

Figure 9 shows the distribution of the MAE values obtained on the test data from four graph-partitioning methods. The MAE values obtained with k-way graph partitioning (2.02), spectral clustering (2.02), and multilevel recursive bisection (2.03) are similar. The median of MAE with the Louvain method is 2.06, which is larger than that of other methods.

The k-way graph partitioning was adopted because of the speed advantage. In particular, the complexity of the k-way graph partitioning approach is $O(\varepsilon \times k)$ (where ε is the number of edges in the graph and k is the number of partitions), and the complexity of spectral clustering is N^3 (where N is the number of nodes in the graph). In a sparse graph such as ours, the number of edges is much smaller than the number of nodes. Therefore, the complexity of the k-way graph partitioning approach is much lower than that of spectral clustering. To perform 64 partition on a graph of 11,160 nodes, the multilevel k-way graph-partitioning method of Metis took only 0.030 s, whereas spectral clustering took 255.56 s.

Comparison with Other Methods

DCRNN is compared with four methods: (*1*) lasso regression (LR): $\alpha = 0.1$, the multiplier of the L1 term (implemented with sklearn Python package) (*63*); (*2*) autoregressive integrated moving average (ARIMA): order (5,1,0) is used for AR parameters, differences, and MA parameters (implemented with statsmodel python package) (*37*); (*3*) standard feed-forward neural networks (FNN): two hidden layers and 256 neuron per layer (implemented with Keras) (*39*); and (*4*) random forests (RF): number of trees in the forest is 100 (implemented with scikit-learn Python package) (*64*).

The METER-LA dataset in Li et al. was used for model comparison (*14*). The dataset contains 207 sensors and the time series data of 4 months collected from March 1, 2012, to June 27, 2012. As described earlier, 70% (from 1 March to 2 May) of the data was used for training, 10% (from 23 May to 4 June) of the data for validation, and 20% (from 4 June to 27 June) of the data for testing. Given 60 min of time series data, all of the models forecast for the next 60 min. The results showed that DCRNN outperformed all other methods. It achieved MAE of 3.60, which is lower than that of LR (7.89), ARIMA (7.73), RF (8.40), and FNN (4.49).

Acknowledgments

David Anderson and Prasad Gupte, the DOE Office of Energy Efficiency and Renewable Energy (EERE) program managers played important roles in establishing the project concept, advancing implementation, and providing ongoing guidance.

Author Contributions

The authors confirm contribution to the paper as follows: study conception and design: T. Mallick, P. Balaprakash, E. Rask, J. Mcfarlane; data collection: T. Mallick; analysis and interpretation of results: T. Mallick, P. Balaprakash, E. Rask, J. Mcfarlane; draft manuscript preparation: T. Mallick, P. Balaprakash. All authors reviewed the results and approved the final version of the manuscript.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This material is based in part on work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility under contract DE-AC02-06CH11357. This report and the work described were sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Big Data Solutions for Mobility Program, an initiative of the Energy Efficient Mobility Systems (EEMS) Program (DE-AC02-06CH11357).

Data Accessibility Statement

The source code and the dataset for graph-partitioning-based DCRNN are available at the following URL along with the script to run it: github.com/tanwimallick/graph_partition_based_DCRNN.

References

1. Bishop, R. *Intelligent Vehicle Technology and Trends*. Artech House, 2005.
2. Teklu, F., A. Sumalee, and D. Watling. A Genetic Algorithm Approach for Optimizing Traffic Control Signals Considering Routing. *Computer-Aided Civil and Infrastructure Engineering*, Vol. 22, No. 1, 2007, pp. 31–43.
3. Tang, S., and H. Gao. Traffic-Incident Detection-Algorithm Based on Nonparametric Regression. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 6, No. 1, 2005, pp. 38–42.
4. Fadlullah, Z. M., F. Tang, B. Mao, N. Kato, O. Akashi, and T. Inoue. State-of-the-Art Deep Learning: Evolving Machine Intelligence toward Tomorrow's Intelligent Network Traffic Control Systems. *IEEE Communications Surveys & Tutorials*, Vol. 19, No. 4, 2017, pp. 2432–2455.
5. Abdulhai, B., R. Pringle, and G. J. Karakoulas. Reinforcement Learning for True Adaptive Traffic Signal Control. *Journal of Transportation Engineering*, Vol. 129, No. 3, 2003, pp. 278–285.
6. Lv, Y., Y. Duan, W. Kang, Z. Li, and F. Y. Wang. Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 16, No. 2, 2014, pp. 865–873.
7. Pang, G. K., K. Takabashi, T. Yokota, and H. Takenaga. Adaptive Route Selection for Dynamic Route Guidance System Based on Fuzzy-Neural Approaches. *IEEE Transactions on Vehicular Technology*, Vol. 48, No. 6, 1999, pp. 2028–2041.
8. DeCorla-Souza, P., J. Everett, B. Gardner, and M. Culp. Total Cost Analysis: An Alternative to Benefit-Cost Analysis in Evaluating Transportation Alternatives. *Transportation*, Vol. 24, No. 2, 1997, pp. 107–123.
9. Williams, B. M., and L. A. Hoel. Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results. *Journal of Transportation Engineering*, Vol. 129, No. 6, 2003, pp. 664–672.
10. Chan, K. Y., T. S. Dillon, J. Singh, and E. Chang. Neural-Network-Based Models for Short-Term Traffic Flow Forecasting using a Hybrid Exponential Smoothing and Levenberg–Marquardt Algorithm. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 13, No. 2, 2012, pp. 644–654.
11. Karlaftis, M. G., and E. I. Vlahogiann. Statistical Methods Versus Neural Networks in Transportation Research: Differences, Similarities and Some Insights. *Transportation Research Part C: Emerging Technologies*, Vol. 19, No. 3, 2011, pp. 387–399.
12. Castro-Neto, M., Y. S. Jeong, M. K. Jeong, and L. D. Han. Online-SVR for Short-Term Traffic Flow Prediction under Typical and Atypical Traffic Conditions. *Expert Systems with Applications*, Vol. 36, No. 3, 2009, pp. 6164–6173.
13. Do, L. N., N. Taherifar, and H. L. Vu. Survey of Neural Network-Based Models for Short-Term Traffic State Prediction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 9, No. 1, 2019, p. 1285.
14. Li, Y., R. Yu, C. Shahabi, and Y. Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. *International Conference on Learning Representations (ICLR '18)*, 2018.
15. Dean, J., G. Corrado, R. Monga, K. Chen, M. Devin, and M. Mao. Large Scale Distributed Deep Networks.

Advances in Neural Information Processing Systems, 2012, pp. 1223–1231.
16. Cascetta, E. *Transportation Systems Engineering: Theory and Methods*, Vol. 49. Springer Science & Business Media, 2013.
17. Romero Calvo, M. *Queuing Theory, Time Series and an Application to Tollgate's Traffic Flow Prediction*. Undergraduate final thesis. University of Barcelona, Barcelona, Spain, 2018.
18. Lartey, J. D. Predicting Traffic Congestion: A Queuing Perspective. *Open Journal of Modelling and Simulation*, Vol. 2, No. 2, 2014, p. 57.
19. Yang, S., and X. Yang. The Application of the Queuing Theory in the Traffic Flow of Intersection. *International Journal of Mathematical, Computational Sciences*, Vol. 8, 2014, pp. 986–989.
20. Ricci, S., and A. Tieri. A Petri Nets Based Decision Support Tool for Railway Traffic Conflicts Forecasting and Resolution. *WIT Transactions on the Built Environment*, Vol. 103, 2008, pp. 483–492.
21. Kumar, S. V. Traffic Flow Prediction using Kalman Filtering Technique. *Procedia Engineering*, Vol. 187, 2017, pp. 582–587.
22. Ahn, J., E. Ko, and E. Y. Kim. Highway Traffic Flow Prediction using Support Vector Regression and Bayesian Classifier. *2016 International Conference on Big Data and Smart Computing (BigComp)*, IEEE, 2016, pp. 239–244.
23. Huang, W., G. Song, H. Hong, and K. Xie. Deep Architecture for Traffic Flow Prediction: Deep Belief Networks with Multitask Learning. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 15, No. 5, 2014, pp. 2191–2201.
24. Lv, Y., Y. Duan, W. Kang, Z. Li, and F. Y. Wang. Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, Vol. 16, No. 2, 2015, pp. 865–873.
25. Ma, X., Z. Tao, Y. Wang, H. Yu, and Y. Wang. Long Short-Term Memory Neural Network for Traffic Speed Prediction using Remote Microwave Sensor Data. *Transportation Research Part C: Emerging Technologies*. Vol. 54, 2015, pp. 187–197.
26. Fu, R., Z. Zhang, and L. Li. Using LSTM and GRU Neural Network Methods for Traffic Flow Prediction. *31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, IEEE, 2016, pp. 324–328.
27. Cui, Z., R. Ke, and Y. Wang. *Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-Wide Traffic Speed Prediction*. arXiv preprint arXiv:180102143, 2018.
28. Yu, R., Y. Li, C. Shahabi, U. Demiryurek, and Y. Liu. Deep Learning: A Generic Approach for Extreme Condition Traffic Forecasting. *Proceedings of the 2017 SIAM International Conference on Data Mining*, SIAM, 2017, pp. 777–785.
29. Ma, X., Z. Dai, Z. He, J. Ma, and Y. Wang. Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction. *Sensors*, Vol. 17, No. 4, 2017, p. 818.
30. Yu, H., Z. Wu, S. Wang, Y. Wang, and X. Ma. Spatiotemporal Recurrent Convolutional Networks for Traffic Prediction in Transportation Networks. *Sensors*, Vol. 17, No. 7, 2017, p. 1501.
31. Zhang, J., Y. Zheng, D. Qi, R. Li, and X. Yi. DNN-Based Prediction Model for Spatio-Temporal Data. *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 2016, p. 92.
32. Zhang, J., Y. Zheng, and D. Qi. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. *31st AAAI Conference on Artificial Intelligence*, 2017.
33. Du, S., T. Li, X. Gong, Z. Yu, Y. Huang, and S. J. Horng. *A Hybrid Method for Traffic Flow Forecasting using Multimodal Deep Learning*. arXiv preprint arXiv:180302099, 2018.
34. Yu, B., H. Yin, and Z. Zhu. *Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting*. arXiv preprint arXiv:170904875, 2017.
35. Atwood, J., and D. Towsley. Diffusion-Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.
36. Hechtlinger, Y., P. Chakravarti, and J. Qin. *A Generalization of Convolutional Neural Networks to Graph-Structured Data*. arXiv preprint arXiv:170408165, 2017.
37. Dw, X., W. Yd, J. Lm, Y. Qin, and D. Hh. Real-Time Road Traffic State Prediction Based on ARIMA and Kalman Filter. *Frontiers of Information Technology & Electronic Engineering*, Vol. 18, No. 2, 2017, pp. 287–302.
38. Hamilton, J. D. *Time Series Analysis*. Princeton University Press, 1994.
39. Raeesi, M., M. Mesgari, and P. Mahmoudi. Traffic Time Series Forecasting by Feedforward Neural Network: A Case Study Based on Traffic Data of Monroe. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, Vol. 40, No. 2, 2014, p. 219.
40. Sutskever, I., O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.
41. Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, and H. Schwenk. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv preprint arXiv:14061078, 2014.
42. Teng, S. H. Scalable Algorithms for Data and Network Analysis. *Foundations and Trends in Theoretical Computer Science*, Vol. 12, No 1–2, 2016, pp. 1–274.
43. Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.
44. Liu, Y., N. Shah, and D. Koutra. *An Empirical Comparison of the Summarization Power of Graph Clustering Methods*. arXiv preprint arXiv:151106820, 2015.

45. Karypis, G., and V. Kumar. Multilevel-Way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, Vol 48, No. 1, 1998, pp. 96–129.
46. Karypis, G., and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, 1998, pp. 359–392.
47. Ng, A.Y., M. I. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an Algorithm. *Advances in Neural Information Processing Systems*, 2002, pp. 849–856.
48. Blondel, V.D., J. L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 10, 2008.
49. Hendrickson, B., and R. Leland. A Multi-Level Algorithm For Partitioning Graphs. *Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, 1995, pp. 1–14.
50. *Caltrans Performance Measurement System (PeMS)*. http://pems.dot.ca.gov/. Accessed May 24, 2019.
51. Luxen, D., and C. Vetter. *Open Source Routing Machine – C++ backend, 2018*. https://github.com/Project-OSRM/osrm-backend. Accessed March 25, 2019.
52. *Open Street Map*, 2019. https://www.openstreetmap.org/#map=5/38.007/-95.844. Accessed May 22, 2019.
53. Shuman, D.I., S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. *The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains*. arXiv preprint arXiv:12110053, 2012.
54. Li, Y., R. Yu, C. Shahabi, and Y. Liu. *Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting*. GitHub, 2018. https://github.com/liyaguang/DCRNN. Accessed June 8, 2020.
55. Kingma, D.P., and J. Ba. *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:14126980. 2014.
56. *METIS – Serial Graph Partitioning and Fill-reducing Matrix Ordering*. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. Accessed March 19, 2019.
57. *Scikit-learn: Decision Tree Classifier*, 2020, pp. 2825–2830. http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Accessed June 8, 2020.
58. Pascanu, R., T. Mikolov, and Y. Bengio. On the Difficulty of Training Recurrent Neural Networks. *International Conference on Machine Learning*, 2013, pp. 1310–1318.
59. Balaprakash, P., M. Salim, T. Uram, V. Vishwanath, and S. Wild. DeepHyper: Asynchronous Hyperparameter Search for Deep Neural Networks. *25th International Conference on High Performance Computing (HiPC)*, IEEE, 2018, pp. 42–51.
60. Sener, O., and V. Koltun. Multi-Task Learning as Multi-Objective Optimization. *Advances in Neural Information Processing Systems*, 2018, pp. 527–538.
61. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, and O. Grisel. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*, Vol 12, 2011, pp. 2825–2830.
62. Aynaud, T. *Louvain Community Detection*. GitHub, 2018. https://github.com/taynaud/python-louvain. Accessed March 25, 2019.
63. Reid, S., R. Tibshirani, and J. Friedman. A Study of Error Variance Estimation in Lasso Regression. *Statistica Sinica*, 2016, pp. 35–67.
64. Breiman, L. Random Forests. *Machine Learning*, Vol. 45, No. 1, 2001, pp. 5–32.