TrajNet: A Trajectory-Based Deep Learning Model for Traffic Prediction

Bo Hui
Auburn University
bohui@auburn.edu

Da Yan
University of Alabama at Birmingham
yanda@uab.edu

Haiquan Chen
California State University, Sacramento
haiquan.chen@csus.edu

Wei-Shinn Ku
Auburn University
weishinn@auburn.edu

ABSTRACT

Ridesharing companies such as Ube and DiDi provide ride-hailing services where passengers and drivers are matched via mobile apps. As a result, large amounts of vehicle trajectories and vehicle speed data are collected that can be used for traffic prediction. The recent popularity of graph convolutional networks (GCNs) has opened up new possibilities for real-time traffic prediction and many GCN-based models have been proposed to capture the spatial correlation on the urban road network. However, the graph-based approaches fail to capture the intricate dependencies of consecutive road segments that are well captured by trajectories.

Instead of proposing yet another GCN-based model for traffic prediction, we propose a novel deep learning model that treats vehicle trajectories as first-class citizens. Our model, called *TrajNet*, captures the spatial dependency of traffic flow by propagating information along real trajectories. To improve training efficiency, we organize the multiple trajectories in a batch used for training with a trie structure, to reuse shared computation. TrajNet uses a spatial attention mechanism to adaptively capture the dynamic correlations between different road segments, and dilated causal convolution to capture long-range temporal dependency. We also resolve the inconsistency between the fine-grained road segment coverage by trajectories, and the ground-truth traffic data that are coarse-grained, following a trajectory-based refinement framework. Extensive experiments on real traffic datasets validate the performance superiority of TrajNet over the state-of-the-art models.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**.

KEYWORDS

Trajectory, Traffic prediction

ACM Reference Format:
Bo Hui, Da Yan, Haiquan Chen, and Wei-Shinn Ku. 2021. TrajNet: A Trajectory-Based Deep Learning Model for Traffic Prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467236

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467236

1 INTRODUCTION

With the rapid process of urbanization, the number of vehicles is growing fast. According to the U.S. Department of Transportation, the average number of motor vehicles owned by households has grown to 1.88 by 2017 [4]. As a result, it is important to develop the Intelligent Transportation System (ITS) [27] for efficient traffic management. Traffic forecasting is an indispensable part of ITS in a smart city with the goal to predict future traffic conditions (e.g., speeds, volumes) in a road network given historic traffic data. Traffic forecasting can help a city better manage traffic to alleviate congestion, help carsharing companies pre-allocate cars to high demand regions, etc. However, due to complex spatiotemporal dependencies, accurate traffic forecasting is challenging.

Recent advances in graph convolution networks (GCN) have enabled more effective modeling of the complex spatiotemporal dependencies of traffic in a road network. A lot of GCN-based traffic prediction models such as T-GCN [29], DCRNN [14], STGCN [26], ASTGCN [9], Graph WaveNet [21] and HetETA [11] have been developed and achieved state-of-the-art performance. These works construct a graph to represent the transportation network, where each node in the graph is a road segment or a sensor station. At each point in time, a node is associated with a vector of features

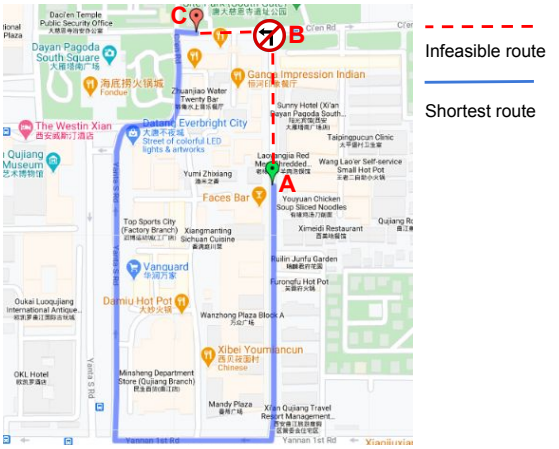


Figure 1: A Traffic Scenario with No-Left-Turn Sign

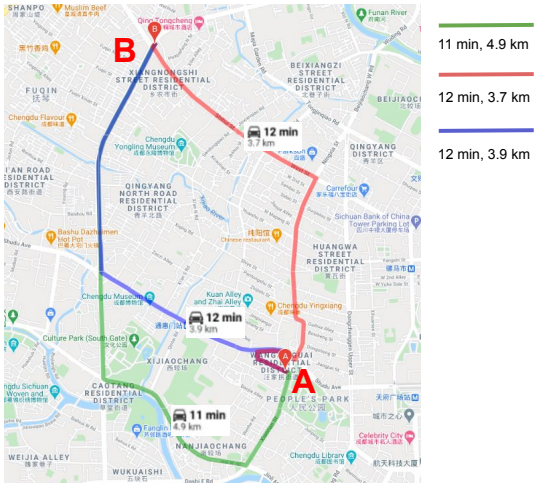


Figure 2: Three Routes from A to B

such as speed, volume, and segment length, so that GCN is directly applicable to capture the spatial dependencies between nodes. Different schemes have been used to capture temporal dependencies such as recurrent neural networks (RNN) and 1D convolutions.

An assumption of the above models is that the graph structure reflects the genuine dependency relationships among nodes. However, in real traffic flow, circumstances can be much more complicated and the assumption does not always hold true.

For example, a connection does not guarantee dependency between two nodes in the road graph. Figure 1 provides an illustration where there is a no-left-turn sign at B, making the blue path the shortest path from the origin location A to the destination location C. In this case, even though segments AB and BC are adjacent, we should not propagate information from one to the other directly which is, however, the case in GCN-based models.

As another example, the distance between two nodes in a road graph may not reflect the true degree of dependency in real traffic flow, since people may consider various road conditions such as the existence of traffic lights, speed limits, chuckholes on the road surface, and even personal preferences (e.g., beautiful scenery along the route). Figure 2 illustrates a scenario where the longest path in green is preferred over the other two shorter paths by actual drivers because the speed limits of road segments are higher, there are only right turns (right-hand traffic) without the need of waiting for traffic lights, and the route passes through two parks with beautiful scenery rather than monotonous neighborhood blocks.

While it is impractical to enumerate and collect all those feature variables that may affect drivers' actual travel patterns, these patterns are faithfully reflected by their trajectories. More importantly, such trajectories are being tracked and collected by carsharing companies, since the mobile apps need to provide timely routes to navigate the drivers to the destinations specified by the passengers. For example, DiDi's Gaia initiative provides drivers' trajectories collected every 2–4 seconds where trajectory points have been map-matched. Such trajectory data provides a valuable source to improve the performance of traffic prediction, but are not effectively utilized by existing GCN-based models. In fact, only HetETA [11] has considered using trajectory data to improve prediction. However, HetETA still falls in the GCN framework where in addition to the road graph, it further creates a vehicle-trajectory-based graph to jointly consider the traffic behavior pattern. Such aggregate graph created out of vehicle trajectories loses important segment correlation information that each individual trajectory provides, and cannot effectively capture the travel patterns of individual rides. A graph is essentially a binary relation between nodes and cannot reserve high-order segment correlations beyond 2 hops without loss of travel information. A solution completely different from a graph-based one is essential to capture multi-hop travel patterns.

Floating car data (FCD) is a term in traffic engineering that refers to timestamped geolocation and speed data directly collected by moving vehicles [3]. Large amounts of FCD data are currently being collected by carsharing companies from the smartphones of their users who use their mobile apps. While most of those data are proprietary assets of carsharing companies used to optimize their services, DiDi's Gaia initiative [1] provides the academic community with real FCD after data anonymization. This allows us to peek into how the carsharing industry monitors and manages traffic data. Specifically, Gaia's *Travel Time Index* (TTI) dataset tracks the average speed data on pre-defined roads in cities. Here, a *road* refers to consecutive *road segments* that are treated as a whole by DiDi when computing the average speed. While the speed on a road may vary from one segment to another, this paper aims to estimate future smoothed speed on the granularity of road segment. Then, the estimated speed on a road can be aggregated by averaging speeds of all segments on the road. We remark that existing works predict traffic conditions (e.g., speeds and volumes) also in aggregated forms (e.g., every 5 minutes as in [26]) to counteract short-term variance. Moreover, according to [2], the speed aggregation method by DiDi's TTI dataset is the *Smart Transportation Industry Standard*, and thus we follow this standard. We also remark that although this paper focuses on segment-level speed prediction, our model is general and can be used to predict other metrics such as volume when such ground-truth supervision data is available.

In this paper, we propose a novel deep learning model that directly treats vehicle trajectories as first-class citizens. Our model, called *TrajNet*, captures the multi-hop spatial dependency of road segments by **propagating information along real trajectories**. First, we conduct dilated causal convolution [20] on recent, daily and weekly periodic data to capture long-range temporal dependency. Then a propagation module is proposed to capture the spatial dependency of traffic flow by propagating temporal features along real trajectories. We also resolve the inconsistency between the coarse granularity (i.e., road-level rather than segment-level) of ground-truth speed data, and the fine-grained road segment coverage by trajectories, following a **trajectory-based refinement** framework. Considering that spatial dependency is dynamic, we thus use a spatial attention mechanism to adaptively capture the correlations between different road segments. Finally, the smoothed spatiotemporal features are used to generate the prediction.

Besides prediction accuracy, this paper also proposes a novel method to sample a batch of trajectories that pass each segment to speed up training. To further improve training efficiency, we organize the multiple trajectories in a training batch with a trie structure, to **reuse shared computation**. Our experiments show that this technique is able to speed up training by a few times.

The contribution of our paper can be summarized as:

- To the best of our knowledge, it is the first time that trajectory based propagation path is used for traffic prediction. We extract real trajectory in the traffic and utilize it to model the spatial dependency.
- We develop an algorithm to reduce the redundancy of the propagation in the model and speed up the training process.
- We refine the prediction on the level of fine-grained segments, while existing methods directly predict the traffic on the level of coarse-grained road.
- Extensive experiments on various real-world traffic datasets validate the superiority of our model over the state of the art. Our model can capture the complex spatial correlation that can not be leveraged by GCN-based methods.

2 FRAMEWORK OVERVIEW

Notations and Problem Definition. We denote the set of road segments by $S = \{s_1, s_2, \ldots, s_{n_s}\}$, where s_i is a road segment and n_s is the total number of segments. Recall that a road is the unit on which the carsharing companies compute the average speed. We denote the set of all roads by $\mathcal{R} = \{R_1, R_2, \ldots, R_{n_r}\}$, where R_i is a road and n_r is the total number of roads. Each road consists of a sequence of consecutive road segments, denoted by $R_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_{n_i}}\}$ where $s_{i_j} \in S$, and R_i has n_i segments. We use x_i^t to denote the traffic speed of road R_i at time t, and define $\mathbf{x}_t = (x_1^t, x_2^t, ..., x_{n_r}^t) \in \mathbb{R}^{n_t}$ as the vector of traffic speeds of all n_r roads.

We denote that set of trajectories by $\mathcal{T} = \{T_1, T_2, \ldots, T_{n_t}\}$, where T_i is a trajectory and n_t is the total number of trajectories. Each trajectory has been map-matched into a sequence of segments: $T_i = [s_{i_1}, s_{i_2}, \ldots, s_{i_{\ell_i}}]$ where $s_{i_j} \in S$, and ℓ_i is the number of segments passed by T_i. Note that we have removed those segments that are not on any road in $\mathcal{R}$.

Given a set of trajectories $\mathcal{T}$ and road-level historical speed data, represented by matrix $\mathbf{X} \triangleq \mathbf{X}_{0:t} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t)^T \in \mathbb{R}^{t \times n_r}$ (we regard $\mathbf{x}_t$ as a column vector here), our goal is to predict the traffic speeds in future τ time slices, represented by matrix $\mathbf{X} \triangleq \mathbf{X}_{(t+1):(t+\tau)} = (\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_{t+\tau})^T \in \mathbb{R}^{\tau \times n_r}$. Note that our problem definition is general: x_i^t can also be other traffic values beyond speed, such as volume and density.

Since our speed data from DiDi is at the coarse-grained road level while our trajectory is at the fine-grained segment level, for each road R_i, we first predict speeds of those segments $s_{i_j} \in R_i$, i.e., $\{\hat{y}_{i_j}^t \mid s_{i_j} \in R_i\}$, then follow [2] to obtain the predicted speed estimate $\hat{x}_i^t$ for road R_i by averaging speeds of all segments on R_i. Let us define $\hat{\mathbf{X}}_{(t+1):(t+\tau)} = (\hat{\mathbf{x}}_{t+1}, \hat{\mathbf{x}}_{t+2}, \ldots, \hat{\mathbf{x}}_{t+\tau})^T \in \mathbb{R}^{\tau \times n_r}$ as the prediction of $\mathbf{X}_{(t+1):(t+\tau)}$. The loss function is computed as the mean absolute error between $\hat{\mathbf{X}}_{(t+1):(t+\tau)}$ and $\mathbf{X}_{(t+1):(t+\tau)}$.

Model Architecture. Figure 3 overviews the architecture of our proposed TrajNet model. We first extract three sequences of traffic data from the matrix $\mathbf{X}_{0:t}$ (Figure 3(a)): recent, daily-periodic and weekly-periodic. Then each type of data is input into dilated convolution layer. Three outputs are concatenated as the temporal feature features $\mathbf{h}_{R_i}$ for each road. Note that we extract C channels, which are marked in Figures 3(b)–(e). In the trajectory propagation module, we refine the coarse-grained features on road level with the fine-grained trajectories to model the spatial dependency. Since there is no segment-level speed data, we initialize the feature vector of a segment s_i, denoted by $\mathbf{h}_{s_i}$, using the feature vector $\mathbf{h}_{R_j}$ of the road R_j that s_i belongs to (i.e., $s_i \in R_j$). Figure 3(d) illustrates the propagation process, where a sequence of temporal features are propagated along the trajectory. A spatial attention mechanism is applied to adaptively capture the dynamic correlations between different road segments. For each segment s_i, its extracted spatiotemporal feature vector $\mathbf{z}_i$ then goes through a fully-connected neural network to obtain the prediction of s_i. We further aggregate the segment-level predictions into road-level predictions following the averaging scheme of [2], which gives rise to $\hat{\mathbf{X}}_{(t+1):(t+\tau)} = [\hat{\mathbf{x}}_{t+1}, \ldots, \hat{\mathbf{x}}_{t+\tau}]^T$.

3 MODEL DESIGN

3.1 Temporal Feature Extraction

To extract temporal features from historical traffic data, we employ dilated causal convolution layers to capture long-range temporal dependencies as inspired by [21]. A 1D causal convolution layer shifts convolutional outputs so that when sliding a filter, we will not use future data in calculation. By stacking 1D causal convolution layers, we make sure that inputs do not influence output steps that precede them in time. Additionally, the convolution layers are dilated with a doubling dilation rate so that the receptive field grows exponentially with the number of hidden layers, enabling our model to capture longer sequences with fewer layers.

To incorporate the impacts of temporal periodicity in traffic, we consider three components for temporal feature extraction from recent, daily-periodic and weekly-periodic traffic data, $\mathbf{X}_P \in \mathbb{R}^{\tau_P \times n_r}$, $\mathbf{X}_D \in \mathbb{R}^{(\tau_D \cdot \tau) \times n_r}$ and $\mathbf{X}_W \in \mathbb{R}^{(\tau_W \cdot \tau) \times n_r}$, respectively, which are temporal subsequences of traffic speeds (or their concatenations) obtained from the historical sequence $\mathbf{X}_{0:t}$. These subsequences are specified by three hyperparameters τ_P, τ_D and τ_W, and we illustrate their meanings using $\tau_P = \tau = 6$, $\tau_D = \tau_W = 2$ as follows.

Assume that we are predicting the speeds from 8:00 am to 9:00 am and today is Thursday, and we use time slices with 10-minute intervals. Then, we have 3 input subsequences: (1) the average speeds of the previous six 10-minute slices (i.e., 7:00 am – 8:00 am); (2) the average speeds between 8:00 am – 9:00 am on Tuesday and Wednesday (6 + 6 = 12 steps concatenated), i.e., the previous $\tau_D = 2$ days; (3) the average speeds between 8:00 am – 9:00 am on Thursday of the last two weeks (6 + 6 = 12 steps concatenated), i.e., the previous $\tau_W = 2$ weeks. Note that we can directly concatenate the historical data on different days for each input sequence, since 1D dilated causal convolutions are good at capturing periodicity.

We also need to add non-linearity to the output of the dilated convolutional network, and we achieve this by using gated linear units (GLU) [7]. The temporal convolution module over $\mathbf{X}_P$ is depicted in Figure 4, and represented by the following equation:

$$\mathcal{H}_P = (\Theta_1 \star \mathbf{X}_P) \odot \sigma(\Theta_2 \star \mathbf{X}_P) \in \mathbb{R}^{n_r \times n_p \times C}, \quad (1)$$

where we use $\star$ to represent the stacked dilated convolution, σ to denote the sigmoid function, $\odot$ to denote the Hadamard product (i.e., element-wise multiplication) and n_p to denote the output vector after convolution. Specifically, given the recent data $\mathbf{X}_P$, we pass it through two parallel dilated convolutional networks with kernels Θ_1 and Θ_2, respectively, to obtain two output tensors. The second

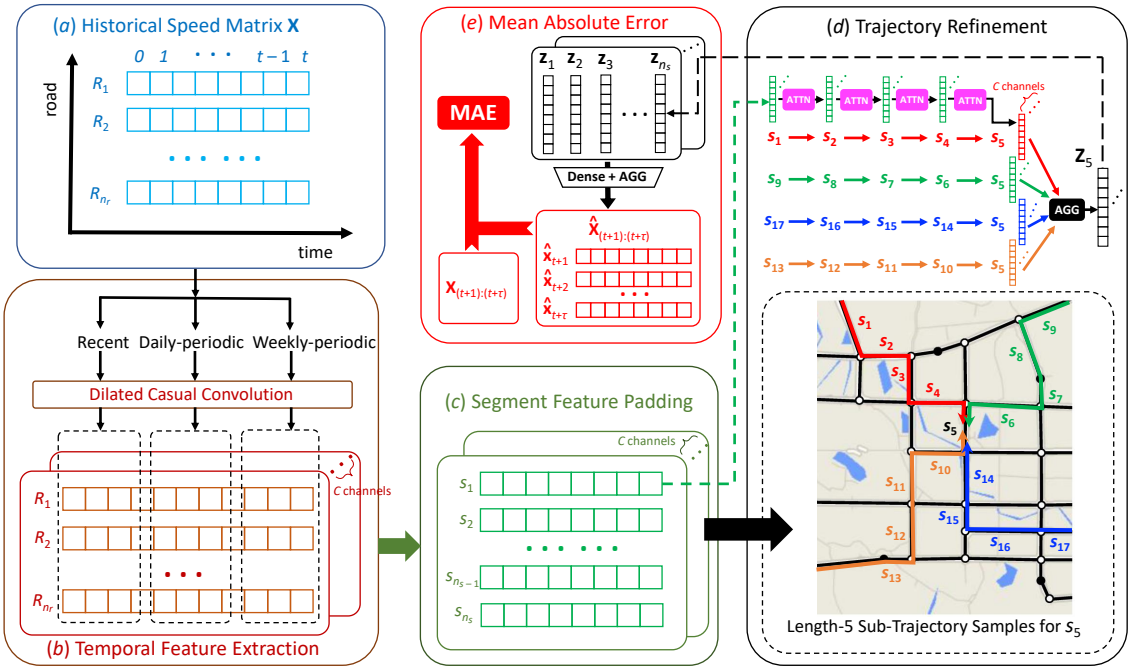


Figure 3: TrajNet Architecture Overview

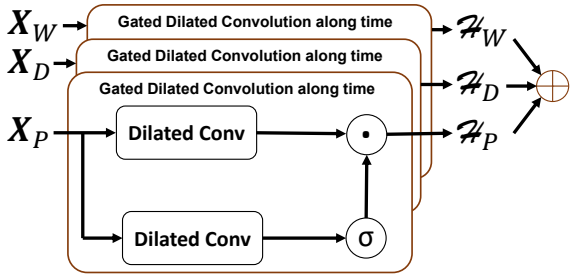


Figure 4: The Temporal Convolution Module with GLU

convolution output is then passed through a sigmoid gate σ which determines the information passed to the downstream modules. This GLU mechanism has been shown to be superior to the LSTM-style gating mechanism, as it allows for faster convergence (c.f. the analysis in Section 3 of [7]).

Likewise, we feed $\mathbf{X}_D$ and $\mathbf{X}_W$ into the same network and then concatenate three output tensors as the final temporal feature:

$$\mathcal{H} = \mathcal{H}_P \oplus \mathcal{H}_D \oplus \mathcal{H}_W \in \mathbb{R}^{n_r \times n_h \times C}, \tag{2}$$

where n_h is the length of concatenated feature.

Since trajectories are sequences of road segments, to allow our downstream trajectory-based feature smoothing module to use the temporal features, we pad segment-level temporal features using the road-level features. The process is illustrated in Figure 3(c), where we compute a segment-level temporal feature tensor $\mathcal{H}' \in \mathbb{R}^{n_s \times n_h \times C}$ as follows: for each segment s_i, assume that it belongs to road R_j (i.e., $s_i \in R_j$), then we pad the content of tensor slice $\mathcal{H}'[s_i] \in \mathbb{R}^{n_h \times C}$ with that of tensor slice $\mathcal{H}[R_j]$. The downstream module will then refine each $\mathbf{h}_{s_i}$ using the trajectories that pass s_i.

3.2 Feature Smoothing by Trajectories

To capture the spatial dependency, we utilize trajectories to refine the coarse-grained temporal features as sketched in Figure 3(d). We remark that feature refinement is necessary and important since roads are predefined segment groups for convenient speed aggregation, and they may not reflect the actual traffic flow where vehicles can join and leave a road at any of its segments. Traffic variability exists even between consecutive road segments, and high-order traffic correlations between segments exist beyond one segment hop. All these fine-grained characteristics are well reflected by trajectories, but mostly missing from the sparse (sampled every 10 minutes) and coarse-grained historical traffic data.

The key idea of our spatial module is to propagate and aggregate temporal features along each individual trajectory. Consider a length-ℓ trajectory $\Gamma = [s_{i_1}, s_{i_2}, \ldots, s_{i_\ell}]$ that ends at segment s_{i_ℓ}. For simplicity, let us omit the channel dimension first so that each segment $s_{i_j} \in \Gamma$ is associated with a temporal feature vector $\mathbf{h}_{s_{i_j}} \in \mathbb{R}^{n_h}$ obtained in $\mathcal{H}'$. Since the propagation context Γ is clear, we directly use $\mathbf{h}_j \triangleq \mathbf{h}_{s_{i_j}}$. Hereafter, we use $\mathbf{z}_j$ and $\mathbf{z}_{s_{i_j}}$ interchangeably when the context Γ is clear. Then, each propagation step is given by:

$$\mathbf{z}_j = \alpha_{s_{i_{j-1}}, s_{i_j}} \cdot \mathbf{W} \cdot \mathbf{z}_{j-1} + \mathbf{h}_j, \quad j = 2, 3, \cdots, \ell, \tag{3}$$

$$\mathbf{z}_1 = \mathbf{h}_1, \tag{4}$$

where $\mathbf{W} \in \mathbb{R}^{n_h \times n_h}$ is a learnable weight matrix, $\alpha_{s_{i_{j-1}}, s_{i_j}}$ is the attention coefficient for propagation from segment $s_{i_{j-1}}$ to s_{i_j} (we will explain how it is computed later), and $\mathbf{z}_j \in \mathbb{R}^{n_h}$ is the adjusted feature vector of segment s_{i_j} after it has been propagated along Γ.

Figure 5 illustrates this recursive propagation process: it takes a sequence of feature vectors $[\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}_4, \mathbf{h}_5]$ as the input, and each step takes a feature vector $\mathbf{z}_{j-1}$ from the previous step and the

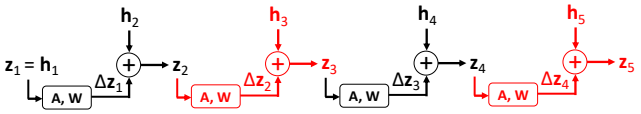


Figure 5: Feature Propagation along a Trajectory

current input $\mathbf{h}_j$ to compute an output $\mathbf{z}_j$ to be used for the next step. Intuitively, $\mathbf{z}_j$ has already taken inputs $[\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_j]$ into account. Also, $\mathbf{W}$ is shared by all steps following the RNN design paradigm. After $(\ell - 1)$ propagation steps, the output of the last propagation, $\mathbf{z}_\ell$, represents the spatially smoothed feature vector of segment s_{i_ℓ} that has been adjusted based on its correlation with the other segments in Γ. Note that for each transition between two consecutive segments, there is a dedicated attention coefficient, which accounts for the dynamic spatial dependencies and variability.

Intuitively, attention coefficients should be dependent on the extracted temporal features, since the spatial dependency of traffic conditions is highly dynamic along the time dimension: for example, two consecutive segments on the downtown route can be highly correlated in a weekday morning because crowds are on their way to work, but not that correlated at noon. Motivated by the spatial attention of ASTGCN [9], we propose the following formula to compute the attention matrix $\mathbf{A}$, using the extracted temporal segment features $\mathcal{H}' \in \mathbb{R}^{n_s \times n_h \times C}$ as the input:

$$\mathbf{E} = \mathbf{V} \cdot \sigma((\mathcal{H}'\mathbf{w}_1)\mathbf{W}_2(\mathbf{w}_3\mathcal{H}')^T + \mathbf{B}), \tag{5}$$

$$\mathbf{A}_{i,j} = \frac{\exp(\mathbf{E}_{i,j})}{\sum_{s_k \in \mathcal{N}(s_i)} \exp(\mathbf{E}_{i,k})}, \text{ if } s_j \in \mathcal{N}(s_i) \tag{6}$$

where $\mathbf{V}, \mathbf{B} \in \mathbb{R}^{n_s \times n_s}$, $\mathbf{w}_1 \in \mathbb{R}^C$, $\mathbf{W}_2 \in \mathbb{R}^{n_h \times C}$, $\mathbf{w}_3 \in \mathbb{R}^{n_h}$ are learnable parameters and σ is the sigmoid activation function. As an overview, a spatial attention matrix $\mathbf{E} \in \mathbb{R}^{n_s \times n_s}$ is first computed from $\mathcal{H}'$, then a normalized attention matrix $\mathbf{A}$ is computed to ensure that the attention weights from a segment s_i to all its adjacent segments sum to 1. To implement Eq (4), we mask out $A_{i,j}$ for non-adjacent pairs (s_i, s_j) when calculating softmax. Given a segment $s \in S$, let us denote its adjacent segments as $\mathcal{N}(s) \subseteq S$ and a segment $s' \in \mathcal{N}(s)$ must immediately follow s in any trajectory.

3.3 Training Efficiency Optimization

The previous section describes how to extract spatialtemporal regarding a specific trajectory. This section describes two optimization techniques that we use in our TrajNet model training in order to improve the training efficiency.

Trajectory Fragment Sampling. Similar to RNN models, our sequence model also admits fixed-length segment sequences from trajectories. Given a sequence length hyperparameter ℓ and a target segment $s \in S$, we truncate a trajectory that passes s as a fragment ending up with s with ℓ segments, and use this trajectory fragment for temporal feature propagation when computing the smoothed feature vector $\mathbf{z}_s$. Besides trajectory truncation, another optimization that we use is trajectory sampling. Following the paradigm of stochastic gradient descent (SGD), deep learning models are trained in mini-batches. Our training of TrajNet is not an exception. Let us assume we have preprocessed the set $\mathcal{T}$ of all trajectories, so that for each segment $s \in S$ we have computed a set of trajectories $\mathcal{T}_s$

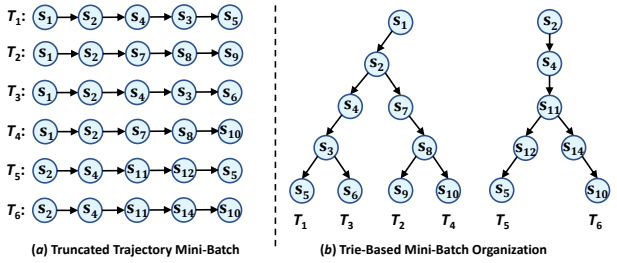


Figure 6: Trie-Based Trajectory Mini-Batch Organization

that pass s. To compute the smoothed feature vector $\mathbf{z}_s$ of segment s, we sample a subset of trajectories from $\mathcal{T}_s$ uniformly at random, and let us denote this set by $\mathcal{B}_s$. Then, for each trajectory T_i in the mini-batch $\mathcal{B}_s$, we truncate it to length-ℓ and then perform temporal feature propagation over the truncated T_i to obtain the smoothed feature vector for s, denoted by $\mathbf{z}_s|_{T_i}$. The final smoothed feature vector $\mathbf{z}_s$ is then estimated by averaging vectors $\mathbf{z}_s|_{T_i}$ of all $T_i \in \mathcal{B}_s$, as is illustrated by Figure 3(d) where $\ell = 5$ and $|\mathcal{B}_s| = 4$.

Compared with averaging $\mathbf{z}_s|_{T_i}$ over all $T_i \in \mathcal{T}_s$, using a trajectory mini-batch to estimate $\mathbf{z}_s$ significantly speeds up computation since otherwise, we need to perform temporal feature propagation along all $T_i \in \mathcal{T}_s$. As long as $|\mathcal{B}_s|$ is not too small, the vector $\mathbf{z}_s$ obtained from $\mathcal{B}_s$ is not too different from one obtained from $\mathcal{T}_s$.

Reusing Shared Propagation Computations. One way to compute $\mathbf{z}_s$ for a mini-batch $\mathcal{B}$ is to perform the propagation computation along each truncated $T_i \in \mathcal{B}$ individually. However, a lot of computations would be redundant. Consider, for example, the mini-batch of 6 truncated trajectories shown in Figure 6(a). Apparently, the first 4 truncated trajectories all start with $s_1 \rightarrow s_2$ and the propagation computation is the same: $\mathbf{z}_2 = \alpha_{s_1,s_2} \cdot \mathbf{W} \cdot \mathbf{h}_1 + \mathbf{h}_2$.

To reuse such computations, we propose to organize the truncated trajectories in a mini-batch $\mathcal{B}$ as a prefix forest (i.e., a forest of tries, aka. prefix trees) as illustrated in Figure 6(b). In this way, for example, we can compute $\mathbf{z}_{11}$ along the propagation path $s_2 \rightarrow s_4 \rightarrow s_{11}$ once in the second trie, and $\mathbf{z}_{11}$ can then be shared by the two branches under node s_{11} to obtain the target vectors $\mathbf{z}_5|_{T_5}$ and $\mathbf{z}_{10}|_{T_6}$. Here we use $\mathbf{z}_5|_{T_5}$ to donate the smoothed vector regarding a specific trajectory T_5. Likewise, $\mathbf{z}_{10}|_{T_6}$ is regarding T_6.

Since the prefix forest is acyclic, we can sort the edges $\{(s_i, s_j)\}$ in the prefix forest topologically, and then compute propagation over the sorted edges one by one. This is because with any topological order, when we process edge (s_i, s_j), we are guaranteed to have already obtained $\mathbf{z}_i$ so the invariant required by Eq (3) is maintained. In our implementation, we simply traverse the trie edges in depth-first order to ensure the topological ordering of edge visits.

Loss Computation: Segment-Road Speed Matching. Now that we have obtained the spatially refined feature vectors $\mathbf{z}_{s_i}$ (aka. $\mathbf{z}_i$) for all segments $s_i \in S$, we need to extract the speed predictions in the future τ time slices out of these features.

Recall that each feature vector $\mathbf{z}_{s_i}$ has length n_h, so a natural choice is to apply a dense layer that takes the length-n_h vector $\mathbf{z}_{s_i}$ as the input, and outputs a vector of length τ:

$$\hat{\mathbf{y}}_{s_i} = [\hat{y}_{s_i}^{t+1}, \hat{y}_{s_i}^{t+2}, \ldots, \hat{y}_{s_i}^{t+\tau}] = FC(\mathbf{z}_{s_i}). \tag{7}$$

Note that we have ignored the channel dimension in previous description, and in actual implementation, $\mathbf{z}_i$ should be replaced by a matrix $\mathbf{Z}_i \in \mathbb{R}^{n_h \times C}$. We flatten $\mathbf{Z}_i$ as input of the dense layer. The final output from the dense layer is the segment-level speed prediction matrix $\hat{\mathbf{Y}}_{(t+1):(t+\tau)} = [\hat{\mathbf{y}}_{s_1}, \hat{\mathbf{y}}_{s_2}, \ldots, \hat{\mathbf{y}}_{s_{n_s}}] = [\hat{\mathbf{y}}_{t+1}, \ldots, \hat{\mathbf{y}}_{t+\tau}]^T$.

Since our ground-truth are associated with roads rather than segments, we need to further aggregate the segment-level predictions into road-level predictions following the averaging scheme of [2], which gives rise to $\hat{\mathbf{X}}_{(t+1):(t+\tau)} = [\hat{\mathbf{x}}_{t+1}, \ldots, \hat{\mathbf{x}}_{t+\tau}]^T$, so that it can be used to compare with road-level ground truth $\mathbf{X}_{(t+1):(t+\tau)}$ for loss computation. The loss function we use is the mean absolute error (MAE) between $\hat{\mathbf{X}}_{(t+1):(t+\tau)}$ and $\mathbf{X}_{(t+1):(t+\tau)}$ as specified by:

$$\mathcal{L} = \mathbf{MAE}(\hat{\mathbf{X}}_{(t+1):(t+\tau)}, \mathbf{X}_{(t+1):(t+\tau)}) = \sum_{i=1}^{n_r} \sum_{j=t+1}^{t+\tau} |\hat{x}_i^j - x_i^j|. \quad (8)$$

4 EXPERIMENTS

4.1 Datasets

We used two data sources from DiDi's Gaia initiative [1] in our experiments to demonstrate the superiority of our model:

- **Travel Time Index (TTI).** This data source provides average driving speed data for cities in Chengdu, Xi'an.
- **Anonymized Trajectories.** This data source provides the anonymized trajectory data of DiDi drivers in two cities.

Since Chengdu (CD) and Xi'an (XA) have both speed data and trajectory data, we construct our datasets for these two cities. Specifically, each road is represented by a road ID and a polyline (a list of GPS coordinates). Similar to the representation of a road, a trajectory is also a list of points specified by their coordinates collected approximately every 2-4 seconds.

We preprocessed the raw data to project both roads and trajectories as a sequence of road segments. The road segments are obtained from OpenStreetMap[1] (OSM) which is built by a community of mappers who contribute data about roads all over the world. Python's Fast Map Matching (FMM) package[2] [23] was used to map roads and trajectories (which are lists of points in the raw data) into segments. The package runs a map-matching algorithm that integrates hidden Markov model with precomputation, to project GPS coordinates to segments (in ESRI shapefile format). We truncated trajectories as trajectory fragments with fixed length for sampling and feature propagation. For a trajectory truncating length ℓ, we slide a window of ℓ segments with stride 1 over the trajectories to obtain the truncated trajectories to sample from during training. We used the first 70% of our datasets along the timeline for training, and the remaining 10% and 20% for validation and test, respectively.

4.2 GCN-Based Models for Comparison

To validate that our trajectory-based approach is truly more effective than the plethora of recent GCN-based predictive models, we selected the following 5 recent state-of-the-art GCN-based models:

- **DCRNN [14].** Diffusion convolutional recurrent neural network captures the spatial dependency using bidirectional random walks on the graph, and the temporal dependency using the encoder-decoder architecture with sampling.
- **STGCN [26].** Spatio-temporal graph convolutional network stacks two spatiotemporal convolutional blocks that adopts a sandwich structure with two temporal gated convolution layers and one spatial graph convolution layer in the middle.
- **ASTGCN [9].** Attention based spatial-temporal graph convolutional network introduces the attention mechanism to capture the dynamic spatialtemporal correlations by graph convolutions and common standard convolutions.
- **Graph Wavenet [21].** Graph WaveNet captures long-range temporal sequences with with a stacked dilated 1D convolution component and learns a self-adaptive adjacency matrix through node embedding to capture the spatial dependency.
- **HetETA [11].** This model comprises temporal convolutions and graph convolutions to learn representations of the spatiotemporal heterogeneous information. Specifically, HetETA translates the road map into a multi-relational graph. and utilizes trajectories to construct a co-occurrence graph. HetETA was proposed to predict the estimated time of arrival; we revised the loss function to predict the traffic.

Model Configuration and Environment. In the experiment, we used the following default hyperparameters in our TrajNet model. The historical speed data of the past hour, periodic data in past 2 days and 1 week were used to predict the traffic speeds in the future 60 minutes (i.e., $\tau = 6$ time slices). The kernel size and dilated step in the 1D dilated causal convolution are configured as 2, and we use $C = 16$ channels for the temporal convolution. The default length for trajectory truncation is $\ell = 7$. We sample trajectories randomly in each epoch and the number of trajectory samples for each segment is $|\mathcal{B}_s| = 5$. We train our models using an Adam optimizer with a learning rate of 0.001 and a weight decay of 0.1. The prefix forest is implemented using the pygtrie library. Our experiments were run on a machine equipped with a 2.20 GHz CPU, 26 GB RAM, and an NVIDIA Tesla P100 GPU with 16GB memory.

4.3 Comparison with GCN-Based Models

Table 1 compares the performance of our model TrajNet with baselines in terms of RMSE, MAPE and MAE. To demonstrate how the prediction accuracy varies with the time, we report RMSE, MAPE and MAE for 10, 20, 30, 60-minutes ahead prediction, as specified by the four columns titled "10 min", "20 min", "30 min", and "60 min" in Table 1. From the table, we obtain the following observations:

- Our TrajNet model consistently has the smallest error in all four durations for all three error metrics. Specifically, in terms of RMSE which is dominated by large prediction errors, TrajNet outperforms the second best model HetETA* by 10% at 10 minutes and by 16% at 60 minutes. This indicates that our model tends to have fewer large errors in individual speed predictions than existing GCN-based models. This is thanks to our treatment of trajectories as first-class citizens, which better captures the speed correlations between segments compared with GCN approaches that mainly utilize a distance-based adjacency matrix that cannot capture the more complex dependencies in real traffic scenarios.
- Compared with DCRNN, both STGCN and ASTGCN tend to have smaller errors thanks to the use of causal convolution followed by GLU in STGCN, and the use of the spatial-temporal attention mechanism in ASTGCN.

[1] https://www.openstreetmap.org/
[2] https://fmm-wiki.github.io/

Table 1: Model Comparison on Speed Prediction Performance

Method	Dataset	10 min			20 min			30 min			60 min		
		RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE
DCRNN	XA	5.38	0.1073	2.99	5.62	0.1140	3.17	5.79	0.1195	3.31	6.38	0.1376	3.78
STGCN		5.02	0.1029	2.86	5.36	0.1137	3.13	5.54	0.1213	3.30	5.94	0.1348	3.63
ASTGCN		4.94	0.0923	2.53	5.37	0.1098	3.01	5.61	0.1209	3.21	6.20	0.1364	3.77
Graph WaveNet		4.56	0.0869	2.51	5.19	0.1049	2.96	5.44	0.1141	3.17	6.05	0.1372	3.71
HetETA*		4.87	0.0926	2.62	5.27	0.1073	2.98	5.47	0.1142	3.22	6.04	0.1244	3.55
TrajNet		**4.32**	**0.0849**	**2.39**	**4.78**	**0.1017**	**2.81**	**4.93**	**0.1071**	**2.93**	**5.23**	**0.1167**	**3.19**
DCRNN	CD	5.62	0.1332	3.40	5.83	0.1395	3.54	6.01	0.144	3.65	6.64	0.1635	4.15
STGCN		5.33	0.1236	3.21	5.65	0.1348	3.44	5.78	0.1399	3.54	6.09	0.1518	3.97
ASTGCN		5.15	0.1119	2.92	5.72	0.1341	3.39	5.95	0.1432	3.58	6.45	0.1623	4.03
Graph WaveNet		4.71	0.1121	2.84	5.35	0.1337	3.36	5.62	0.1412	3.56	6.04	0.1574	3.83
HetETA*		5.12	0.1137	2.95	5.66	0.1324	3.35	5.91	0.1403	3.54	6.16	0.1544	3.85
TrajNet		**4.36**	**0.1064**	**2.71**	**4.79**	**0.1204**	**2.98**	**4.87**	**0.1236**	**3.03**	**5.11**	**0.1291**	**3.18**

- As the time slice moves from "10 min" to "60 min", the errors generally increase due to the difficulty in predicting long-range into the future. However, the previous three observations on the comparative performance remain true. In particular, we observe larger improvements over baselines on long-range prediction (about 0.4 RMSE improvement on 10 minutes and 0.9 RMSE improvement on 60 minutes).

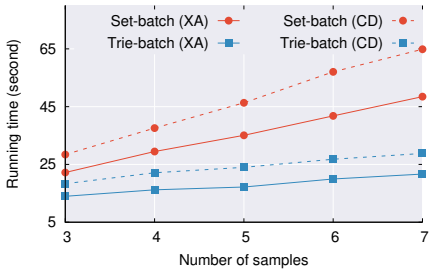
4.4 TrajNet Performance Study

Efficiency of Propagation Computation Reuse. We now study how much efficiency improvement can be achieved by organizing a trajectory mini-batch $\mathcal{B}$ using our proposed prefix forest (denoted by **Trie-batch**), compared with treating the mini-batch simply as a set of individually-propagated truncated trajectory samples (denoted by **Set-batch**). In addition, we also study how the running time for each trajectory mini-batch during training changes with (1) trajectory truncation length ℓ, and (2) the number of random trajectory samples for each segment (i.e., target of propagation).

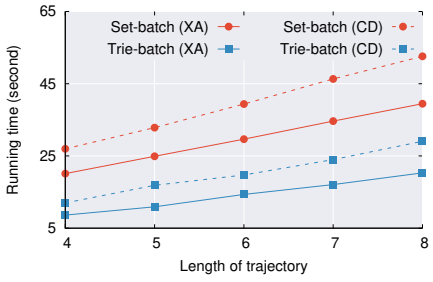
Figure 7(a) shows the results when we fix $\ell = 7$ as default, and vary $|\mathcal{B}_s| = 3, 4, 5, 6, 7$. Figure 7(b) shows the results when we fix $|\mathcal{B}_s| = 5$ as default, and vary $\ell = 4, 5, 6, 7, 8$. From both figures, we can observe a significant speed up of Trie-batch over Set-batch. Moreover, as ℓ and $|\mathcal{B}_s|$ increase, Trie-batch reduces more running time thanks to the increase in the degree of prefix redundancy that can be reused for propagation computation.

Effectiveness of Using Daily- and Weekly-Periodic Data. Figure 8 depicts the performance ("10 minutes" ahead prediction) of TrajNet variant (the red line), which only utilizes the recent data. Compared with TrajNet (the blue line), the RMSE of this variant increases significantly. We can also observe larger RMSE increase as the time varies from "10 minutes" to "60 minutes". This verifies the advantage of using daily and weekly periodic data, especially for long-range prediction.

Effectiveness of Attention Mechanism. We also investigate the effectiveness of attention mechanism by removing it from TrajNet. As shown in Figure 8, compared with TrajNet, this variant (the green line) has larger prediction errors on both data sets. Since the spatial dependencies are dynamic and the real-time correlation can be revealed by the patterns of historical traffic data, the attention



(a) Varying Per-Segment Trajectory Mini-Batch Size



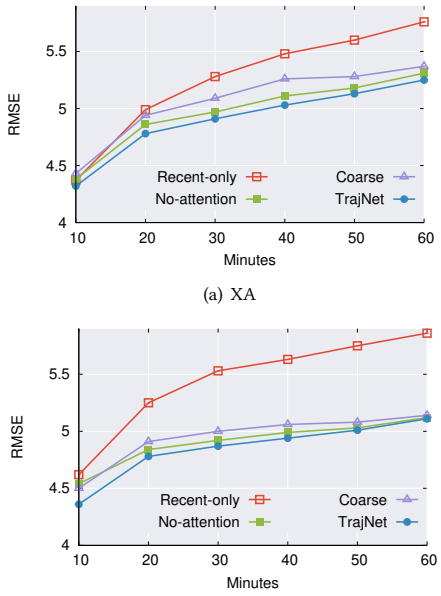
(b) Varying Trajectory Truncation Length

Figure 7: Running Time per Trajectory Mini-Batch

coefficients learned from the temporal feature in TrajNet can reveal the dynamic dependencies.

Effectiveness of Fine-Grained Segment Modeling. TrajNet uses trajectories to refine temporal features at the segment level to account for the speed difference in the same road unit. Instead of fine-tuning the prediction at the segment level, we can also directly predict the traffic at the road-unit level as in existing works. We modify TrajNet and treat trajectories as a list of road-unit sequences. This variant is shown in Figure 8 as "Coarse", and we can see that its RMSE (the purple line) is larger than TrajNet on both data sets. This indicates that our segment-level refinement is effective in modeling the segment differences inside the road units.

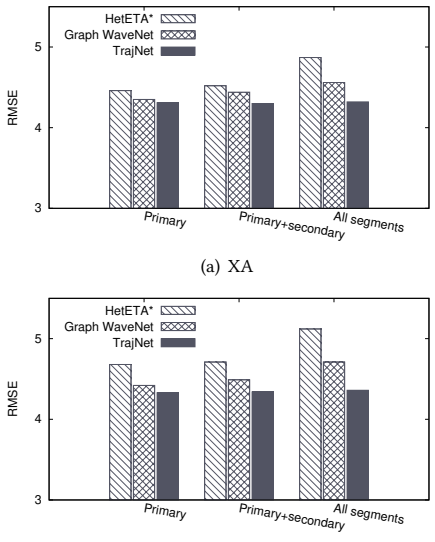
Sparsity of Road Segments. OpenStreetMap classifies road segments into several levels such as primary, secondary, residential,



(a) XA

(b) CD

Figure 8: Ablation study



(a) XA

(b) CD

Figure 9: Sparsity of Roads

etc. This classification provides us a way to investigate how the sparsity of road segments that we consider influences the prediction quality. We consider 3 sets of road segments: (1) Primary, which only includes road segments that correspond to primary roads; (2) Primary+secondary, which includes both primary and secondary roads; (3) All segments: which considers roads at all levels.

As shown in Figure 9 (RMSE at "10 minutes"), our model outperforms the best two GCN-based model, HetETA* and Graph WaveNet, in all three categories. Note that when more lower-level segments are involved, the performance of HetETA* and Graph WaveNet decreases on both datasets since speed prediction on non-primary roads is more difficult (due to more diverse driving patterns). In contrast, the performance of TrajNet is more stable with varying sparsity thanks to the trajectory-based spatial smoothing.

Effect of Model Hyperparameters on Accuracy. Lastly, to investigate the effectiveness of trajectory sampling, we vary the number of trajectory samples for each segment, and the results at "10 minutes" are reported in Figure 10(a)–(b), where we see performance improvements as $\mathcal{B}_s$ increases. Likewise, we vary the trajectory truncation length ℓ. Figure 10(c)–(d) shows that the error at "10 minutes" decreases with ℓ as longer-range spatial dependencies are captured. In both experiments, we observe limited performance improvement beyond our default values, hence our choice is justified since larger values will increase computation costs.

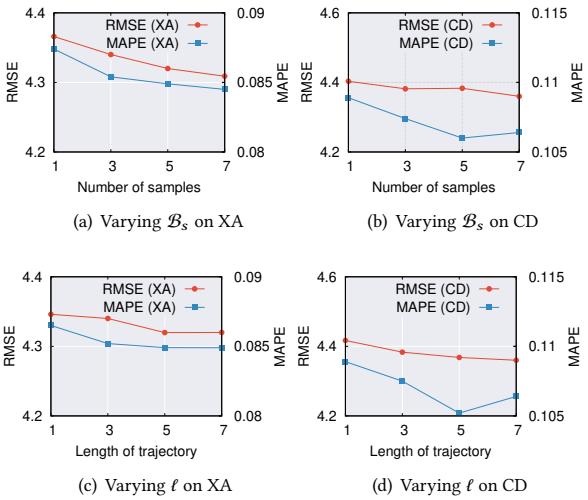
5 RELATED WORK

We have reviewed the most important GCN-based models for traffic prediction in Section 4.2. This section covers more related works in the literature. The idea of capturing spatiotemporal dependencies using deep learning models was originally driven by the efforts to combine CNN that operates on images with RNN that operate on a sequence. CNN-LSTM is a commonly used design paradigm for modeling a sequence of images (e.g., frames in a video), where a feature vector is extracted from each image using CNN, so that the sequence of feature vectors can be input to RNN for learning. ConvLSTM [16] combines CNN and LSTM by replacing the matrix-vector multiplications in LSTM with convolutions, so that ConvLSTM can directly admit a sequence of images (rather than vectors). Later improvements include ConvGRU and TrajGRU [17] that learn the natural location-variant structure.

As the most popular graph neural network (GNN) paradigm, GCN has gained much momentum recently. Spectral graph convolutions [5, 13] operate on the entire graph Laplacian while localized graph convolutions [10, 12, 25] generate node features by aggregating features from a node's local neighborhood for efficiency.

Motivated by the success of combining CNN and RNN, recent works explore the combination of GCN and RNN (or 1D CNN) to capture spatiotemporal correlations. Besides the models reviewed in Section 4.2, other works include T-GCN [29] which proposes a CNN-LSTM counterpart, i.e., GCN-GRU. GCRN [15] replaces the matrix-vector multiplications in LSTM with graph convolutions. GaAN [15] adds a multi-head attention mechanism to GCN, with a convolutional sub-network to control each attention head's importance. ST-GCN [22] adds temporal edges to connect each node with itself at the previous and the next time steps, so that GCN can be applied in the unfolded graph with both spatial and temporal edges. STSGCN [19] adopts a similar idea but it considers localized spatial-temporal graphs not unfolding along all time steps. Graph WaveNet [21] pioneers the idea of using dilated causal convolution to capture temporal correlations. GSTNet [8] proposes tensor causal convolution to capture correlations of feature along the temporal dimension, and multi-resolution convolutions are used to capture multi-resolution temporal dependencies. A number of models also consider temporal periodicity by using historical data of different temporal granularity, such as ASTGCN, HetETA and APTN [18]. STDN [24] further handles the temporal shifting of periodicity.

Besides trajectories, other data sources have been utilized for traffic forecast, such as navigation data by H-STGCN [6] and pickup



(a) Varying $\mathcal{B}_s$ on XA (b) Varying $\mathcal{B}_s$ on CD

(c) Varying ℓ on XA (d) Varying ℓ on CD

Figure 10: Effect of Model Hyperparameters (10 minutes)

data by HetETA. Also, besides traffic prediction, spatiotemporal deep learning models have also been used for other geospatial applications such as citywide crowd flow [28].

In summary, a plethora of GCN-based models have been proposed in recent years combining GCN for spatial correlations with RNN (or temporal CNN) to capture temporal correlations.

6 CONCLUSION

Instead of proposing yet another GCN-based model for traffic prediction, we broke new ground by proposing a novel deep learning model that treats vehicle trajectories as first-class citizens. Our model, called TrajNet, captures the spatiotemporal dependency of traffic flow by propagating information along trajectories. Experiments validate the superiority of TrajNet over GCN-based models.

ACKNOWLEDGMENTS

This research has been funded in part by the U.S. National Science Foundation grants IIS-1618669 (III) and ACI-1642133 (CICI).

REFERENCES

[1] DiDi's GAIA Initiative. https://outreach.didichuxing.com/research/opendata/en/.
[2] DiDi's Travel Time Index. https://github.com/didi/TrafficIndex.
[3] Floating Car Data. https://en.wikipedia.org/wiki/Floating_car_data.
[4] Transportation Statistics Annual Report 2018. https://www.bts.dot.gov/sites/bts.dot.gov/files/docs/browse-statistical-products-and-data/transportation-statistics-annual-reports/Preliminary-TSAR-Full-2018-a.pdf.
[5] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.
[6] R. Dai, S. Xu, Q. Gu, C. Ji, and K. Liu. Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data. In *KDD*, pages 3074–3082. ACM, 2020.
[7] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *ICML*, volume 70, pages 933–941. PMLR, 2017.
[8] S. Fang, Q. Zhang, G. Meng, S. Xiang, and C. Pan. Gstnet: Global spatial-temporal network for traffic flow prediction. In *IJCAI*, pages 2286–2293. ijcai.org, 2019.
[9] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, pages 922–929. AAAI Press, 2019.
[10] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.
[11] H. Hong, Y. Lin, X. Yang, Z. Li, K. Fu, Z. Wang, X. Qie, and J. Ye. Heteta: Heterogeneous information network embedding for estimating time of arrival. In *KDD*, pages 2444–2454. ACM, 2020.
[12] B. Hui, D. Yan, W. Ku, and W. Wang. Predicting economic growth by region embedding: A multigraph convolutional network approach. In M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, editors, *CIKM*, pages 555–564. ACM, 2020.
[13] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net, 2017.
[14] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR (Poster)*. OpenReview.net, 2018.
[15] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *ICONIP (1)*, pages 362–373. Springer, 2018.
[16] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810, 2015.
[17] X. Shi, Z. Gao, L. Lausen, H. Wang, D. Yeung, W. Wong, and W. Woo. Deep learning for precipitation nowcasting: A benchmark and A new model. In *NIPS*, pages 5617–5627, 2017.
[18] X. Shi, H. Qi, Y. Shen, G. Wu, and B. Yin. A spatial-temporal attention approach for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10, 2020.
[19] C. Song, Y. Lin, S. Guo, and H. Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *AAAI*, pages 914–921. AAAI Press, 2020.
[20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, 2016.
[21] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *IJCAI*, pages 1907–1913. ijcai.org, 2019.
[22] S. Yan, Y. Xiong, and D. Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, pages 7444–7452. AAAI Press, 2018.
[23] C. Yang and G. Gidofalvi. Fast map matching, an algorithm integrating hidden markov model with precomputation. *International Journal of Geographical Information Science*, 32(3):547–570, 2018.
[24] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI*, pages 5668–5675. AAAI Press, 2019.
[25] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, pages 974–983. ACM, 2018.
[26] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640. ijcai.org, 2018.
[27] J. Zhang, F. Wang, K. Wang, W. Lin, X. Xu, and C. Chen. Data-driven intelligent transportation systems: A survey. *IEEE Trans. Intell. Transp. Syst.*, 12(4):1624–1639, 2011.
[28] J. Zhang, Y. Zheng, and D. Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661. AAAI Press, 2017.
[29] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Trans. Intell. Transp. Syst.*, 21(9):3848–3858, 2020.