ESI SBA

Supervised By: Dr Belkacem Khaldi December 10, 2023

Graph Neural Networks for Real Time Traffic Prediction

by

MADANI YOUSFI Assia LASHEB Mohamed Amine
ABDALLAOUI Walid

ESI SBA – 2023

1 Introduction

The recent success of Neural Networks has boosted research on pattern recognition and data mining. Machine Learning tasks such us objects detection, machine translation and speech recognition have been given new life with an end-to-end Deep Learning paradigms like CNN, RNN or autoencoders. Deep Learning is good at capturing hidden patterns of Euclidean data (Euclidean data refers to data which can be represented in an Euclidean Space like Vectors Arrays..etc. For instance, images and videos are a good example of Euclidean data).

But what about fields where data is generated from a non-Euclidean domains, represented as graphs, with complex relationships and inter-dependencies between objects ?

That's where Graph Neural Networks become handy which we will explore in depth in the next sections. We will start with a basic Graph Theory, passing to Neural networks in general ,Graph Neural Networks in particular and and we will finish by how we can use a GNN for Real Time Traffic Prediction.

2 Definitions

2.1 Graph Theory

In computer science, a Graph is a data structure which has two components: nodes and edges.Nodes represent entities and edges reprsent the relationship between entities. The nodes and edges form the topology structure of the graph. Besides the graph structure, nodes, edges and/or the whole graph can be associated with rich information reprsented as node/edge/graph features (also known as attributes). So given a graph **G** we can define it as follow:

$$\mathbf{G = (V, E)}$$

where **V** is the set of Nodes and **E** is the set of Edges.
If there are directional dependencies between nodes then the graph is called a **Directed Graph**. (See Figure 1).
If there are not any directional dependencies between nodes, then it's called an **Undirected Graph** (See Figure 2).

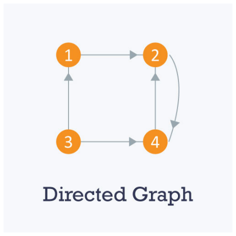


Figure 1: Directed Graph

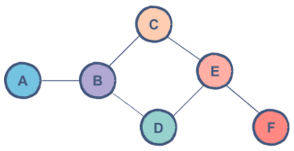


Figure 2: Undirected Graph

2.2 Neural Networks

A neural network is a computational model inspired by the way biological neural networks in the human brain process information. It is a key technology in the field of machine learning and artificial intelligence, used for recognizing patterns and making decisions based on complex and often non-linear data.

2.3 Graph Neural Networks

A Graph Neural Network (GNN) is a type of neural network designed to process data represented in graph form. They provide an easy way to do node-level, edge-level, and graph-level prediction task. GNNs can do what other neural networks architecture (CNN, RNN) fail to do when it comes to working on graph based data. Here is an example of a graph based data (See figure 3) and a GNN (See figure 4).

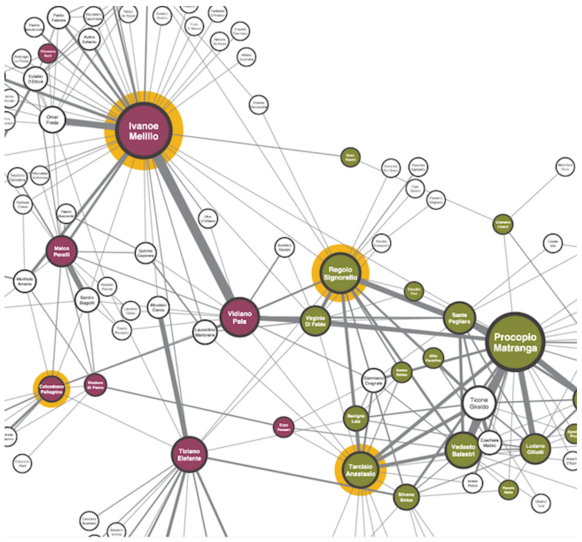


Figure 3: Social Media Data

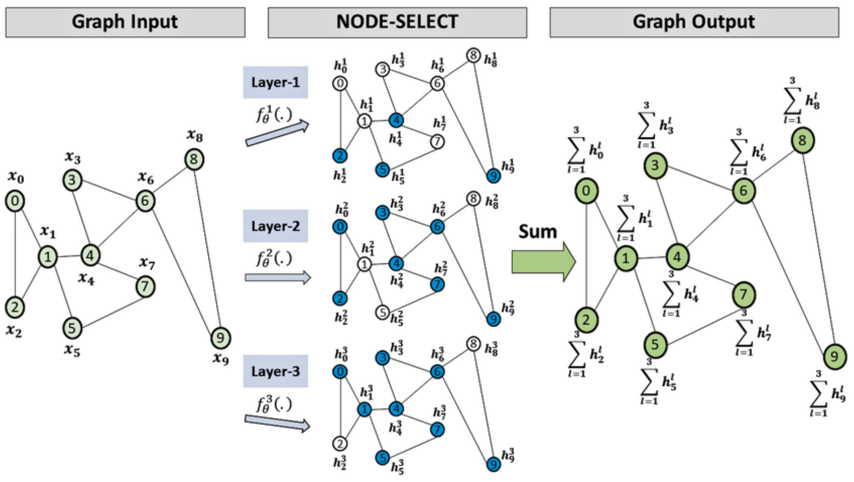


Figure 4: Example of a GNN

3 Real Time Traffic Detection

3.1 Introduction

In urban environments there are daily issues of traffic congestion which city authorities need to address. Real time analysis of traffic flow information is crucial for efficiently managing urban traffic. The 2018 UN Urbanization report predicts that smart cities worldwide are developing rapidly and more then 2.5 billion people are going to live in cities by 2050. Thus, many transport problems are emerging simultaneously especially traffic congestion in urban areas. Due to the nonlinearity and the complexity of traffic flow traditional methods cannot satisfy the requirements of mid-and-long term prediction tasks and often neglect spatial and temporal dependencies.

3.2 Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting

In this paper, researchers suggested a novel deep learning framework: Spatio-Temporal Graph Convulotional Networks (STGCN) to tackle the time series prediction problem in traffic domain. Instead of applying regular convolutional and recurrent units, they formulate the problem on graphs and build the model with complete convolutional structures, which enable much faster training speed with fewer parameters. Experiments show that their model STGCN effectively captures comprehensive spatio-temporal correlations through modeling multi-scale traffic networks and consistently outperforms state-of-the-art baselines on various real-world traffic datasets.

3.2.1 Traffic Prediction On Road Graphs

Traffic forecast is a typical time-series prediction problem, i.e. predicting the most likely traffic measurements (e.g. speed or traffic flow) in the next H time steps given the previous M traffic observations as:

$$\hat{v}_{t+1}, \ldots, \hat{v}_{t+H} = \underset{v_{t+1},\ldots,v_{t+H}}{\arg\max} \log P(v_{t+1}, \ldots, v_{t+H} | v_{t-M+1}, \ldots, v_t)$$

where $v_t \in R^n$ is an observation vector of *n* road segments at time step *t*, each

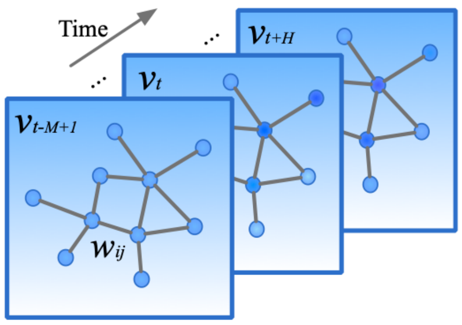


Figure 5: Graph-structured traffic data. Each v_t indicates a frame of current traffic status at time step *t*, which is recorded in a graph- structured data matrix [1]

element of which records historical observation for a single road segment. In this work, they define the traffic network on a graph and focus on structured traffic time series. The observation v_t is not independent but linked by pairwise connection in a graph. Therefore, the data point v_t can be regarded as a graph signal that is defined on an undirected graph (or directed one) $\mathcal{G}$ with weights w_{ij}; as shown in Figure 5. At the t-th time step, in graph $\mathcal{G}_t = (V_t, E, W)$, V_t is a finite set of vertices, corresponding to the observations from n monitor stations in a traffic network; E is a set of edges, indicating the connectedness between stations; while $W \in R^{n \times n}$ denotes the weighted adjacency matrix of $\mathcal{G}_t$.

3.2.2 Proposed Model and Network Architecture

In this section, we elaborate on the proposed architecture of spatio-temporal graph convolutional networks (STGCN) discussed in the related paper. As shown in Figure 6, STGCN is composed of several spatio- temporal convolutional blocks, each of which is formed as a “sandwich” structure with two gated sequential convolution layers and one spatial graph convolution layer in between.The framework STGCN consists of two spatio-temporal convolutional blocks (ST-Conv blocks) and a fully-connected output layer in the end. Each ST-Conv block contains two temporal gated convolution layers and one spatial graph convolution layer in the middle. The residual connection and bottleneck strategy are applied inside each block. The input $v_{t-M+1}, \ldots, v_t$ is uniformly processed by ST-Conv blocks to ex-

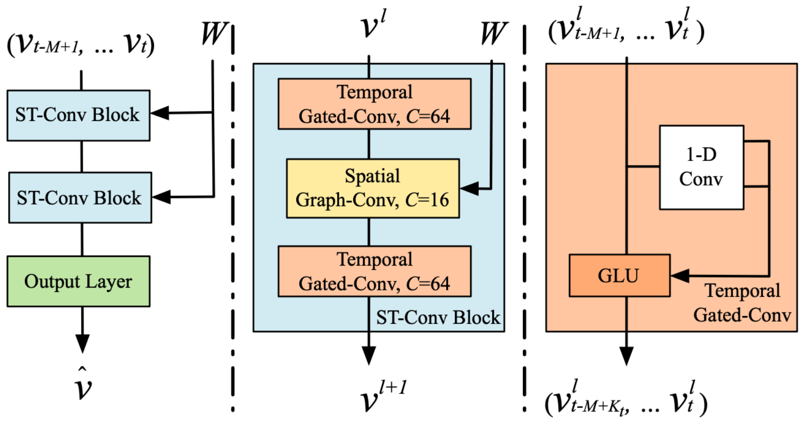

$(v_{t-M+1}, \dots v_t)$
ST-Conv Block
ST-Conv Block
Output Layer
$\hat{v}$
W
v^l
Temporal Gated-Conv, C=64
Spatial Graph-Conv, C=16
Temporal Gated-Conv, C=64
ST-Conv Block
W
v^{l+1}
$(v^l_{t-M+1}, \dots v^l_t)$
1-D Conv
GLU
Temporal Gated-Conv
$(v^l_{t-M+K_t}, \dots v^l_t)$

Figure 6: Architecture of spatio-temporal graph convolutional networks

plore spatial and temporal dependencies coherently. Comprehensive features are integrated by an output layer to generate the final prediction $\hat{v}$.

3.2.3 Graph CNNs for Extracting Spatial Features

The traffic network generally organizes as a graph structure. It is natural and reasonable to formulate road networks as graphs mathematically. However, previous studies neglect spatial attributes of traffic networks: the connectivity and globality of the networks are overlooked, since they are split into multiple segments or grids. Even with 2-D convolutions on grids, it can only capture the spatial locality roughly due to compromises of data modeling. Accordingly, in their model, the graph convolution is employed directly on graph structured data to extract highly meaningful patterns and features in the space domain. Though the computation of kernel θ in graph convolution can be expensive due to $O(n^2)$ multiplications with graph Fourier basis, two approximation strategies are applied to overcome this issue.

Chebyshev Polynomials Approximation to localize the filter and reduce the number of parameters, the kernel θ can be restricted to a polynomial of Λ as $\Theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$ where $\theta \in R^k$ is a vector of polynomial coefficient. *K* is the kernel size of graph convolution which determines the maximum radius of the convolution from central nodes. Traditionally, Chebyshev polynomial $T_k(x)$ is used to approximate kernels as a truncated expansion of order $K-1$ as

$$\Theta(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda})$$

with rescaled $\tilde{\Lambda} = 2\Lambda/\lambda_{\text{max}} - I_n$ (λ_{max} denotes the largest eigenvalue of L) [Hammond et al., 2011]. The graph convolution can then be rewritten as:

$$\Theta *_g x = \Theta(L)x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x \tag{1}$$

where $T_k(\tilde{L}) \in R^{n\times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} = 2L/\lambda_{\text{max}} - I_n$

1st-order Approximation A layer-wise linear formulation can be defined by stacking multiple localized graph convolutional layers with the first-order approximation of graph Laplacian [Kipf and Welling, 2016]. Consequently, a deeper architecture can be constructed to recover spatial information in depth without being limited to the explicit parameterization given by the polynomials. Due to the scaling and normalization in neural networks, we can further assume that $\lambda_{\text{max}} \approx 2$. Thus, the Eq. (1) can be simplified to,

$$\Theta *_g x \approx \theta_0 x + \theta_1 \left(\frac{2}{\lambda_{\text{max}}} L - I_n\right) x \tag{2}$$

$$\approx \theta_0 x - \theta_1 \left(D^{-\frac{1}{2}} W D^{-\frac{1}{2}}\right) x, \tag{2}$$

where θ_0, θ_1 are two shared parameters of the kernel. In order to constrain parameters and stabilize numerical performances, θ_0 and θ_1 are replaced by a single parameter θ by letting $\theta = \theta_0 = -\theta_1$; W and D are renormalized by $\widetilde{W} = W + I_n$ and $\widetilde{D}_{ii} = \sum_j \widetilde{W}_{ij}$ separately. Then, the graph convolution can be alternatively expressed as,

$$\Theta *_g x = \theta(I_n + D^{-\frac{1}{2}} \widetilde{W} D^{-\frac{1}{2}})x \tag{3}$$

$$= \theta\left(\widetilde{D}^{-\frac{1}{2}} \widetilde{W} \widetilde{D}^{-\frac{1}{2}}\right) x. \tag{3}$$

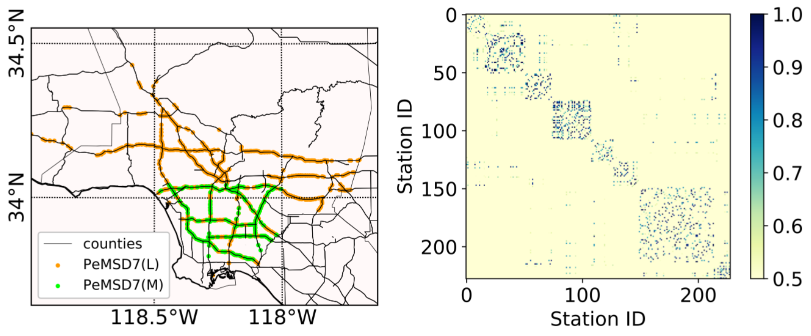


Figure 7: PeMS sensor network in District 7 of California (left), each dot denotes a sensor station; Heat map of weighted adjacency matrix in PeMSD7(M) (right).

3.2.4 Experiments

a- Dataset Description: The mentioned model is verified on two real-world traffic datasets: BJER4 and PeMSD7, collected by Beijing Municipal Traffic Commission and California Department of Transportation, respectively. Each dataset contains key attributes of traffic observations and geographic information with corresponding timestamps, as detailed below:

BJER4 was gathered from the major areas of east ring No.4 routes in Beijing City by double-loop detectors. There are 12 roads selected for our experiment. The traffic data are aggregated every 5 minutes. The time period used is from 1st July to 31st August, 2014 except the weekends. We select the first month of historical speed records as training set, and the rest serves as validation and test set respectively.

PeMSD7 was collected from Caltrans Performance Measurement System (PeMS) in real-time by over 39000 sensor stations, deployed across the major metropolitan areas of California state highway system [Chen et al., 2001]. The dataset is also aggregated into 5 minute interval from 30 second data samples. We randomly select a medium and a large scale among the District 7 of California containing 228 and 1026 stations, labeled as **PeMSD7(M)** and **PeMSD7(L)**, respectively, as data sources (shown in the left of Figure 7). The time range of PeMSD7 dataset is in the weekdays of May and June of 2012. We split the training and test sets based on the same principles as above.

b- Experiments Results: Table 1 and 2 demonstrate the results of STGCN and base-lines on the datasets BJER4 and PeMSD7(M/L). The proposed model achieves the best performance with statistical significance (two-tailed T-test, α = 0.01, $P < 0.01$) in all three evaluation metrics. We can easily observe that traditional statistical and machine learning methods may perform well for short-term forecasting, but their long-term predictions are not accurate because of error accumulation, memorization issues, and absence of spatial information. ARIMA model performs the worst due to its incapability of handling complex spatio-temporal data. Deep learning approaches generally achieved better prediction results than traditional machine learning models.

Model	BJER4 (15/ 30/ 45 min)		
	MAE	MAPE (%)	RMSE
HA	5.21	14.64	7.56
LSVR	4.24/ 5.23/ 6.12	10.11/ 12.70/ 14.95	5.91/ 7.27/ 8.81
ARIMA	5.99/ 6.27/ 6.70	15.42/ 16.36/ 17.67	8.19/ 8.38/ 8.72
FNN	4.30/ 5.33/ 6.14	10.68/ 13.48/ 15.82	5.86/ 7.31/ 8.58
FC-LSTM	4.24/ 4.74/ 5.22	10.78/ 12.17/ 13.60	5.71/ 6.62/ 7.44
GCGRU	3.84/ 4.62/ 5.32	9.31/ 11.41/ 13.30	5.22/ 6.35/ 7.58
STGCN(Cheb)	**3.78/ 4.45/ 5.03**	**9.11/ 10.80/ 12.27**	**5.20/ 6.20/ 7.21**
STGCN(1^{st})	3.83/ 4.51/ 5.10	9.28/ 11.19/ 12.79	5.29/ 6.39/ 7.39

Table 1: Performance comparison of different approaches on the dataset BJER4.

Model	PeMSD7(M) (15/ 30/ 45 min)			PeMSD7(L) (15/ 30/ 45 min)		
	MAE	MAPE (%)	RMSE	MAE	MAPE (%)	RMSE
HA	4.01	10.61	7.20	4.60	12.50	8.05
LSVR	2.50/ 3.63/ 4.54	5.81/ 8.88/ 11.50	4.55/ 6.67/ 8.28	2.69/ 3.85/ 4.79	6.27/ 9.48/ 12.42	4.88/ 7.10/ 8.72
ARIMA	5.55/ 5.86/ 6.27	12.92/ 13.94/ 15.20	9.00/ 9.13/ 9.38	5.50/ 5.87/ 6.30	12.30/ 13.54/ 14.85	8.63/ 8.96/ 9.39
FNN	2.74/ 4.02/ 5.04	6.38/ 9.72/ 12.38	4.75/ 6.98/ 8.58	2.74/ 3.92/ 4.78	7.11/ 10.89/ 13.56	4.87/ 7.02/ 8.46
FC-LSTM	3.57/ 3.94/ 4.16	8.60/ 9.55/ 10.10	6.20/ 7.03/ 7.51	4.38/ 4.51/ 4.66	11.10/ 11.41/ 11.69	7.68/ 7.94/ 8.20
GCGRU	2.37/ 3.31/ 4.01	5.54/ 8.06/ 9.99	4.21/ 5.96/ 7.13	2.48/ 3.43/ 4.12 *	5.76/ 8.45/ 10.51 *	4.40/ 6.25/ 7.49 *
STGCN(Cheb)	**2.25/ 3.03/ 3.57**	5.26/ **7.33/ 8.69**	**4.04/ 5.70/ 6.77**	**2.37/ 3.27/ 3.97**	**5.56/ 7.98/ 9.73**	**4.32/ 6.21/ 7.45**
STGCN(1^{st})	2.26/ 3.09/ 3.79	**5.24**/ 7.39/ 9.12	4.07/ 5.77/ 7.03	2.40/ 3.31/ 4.01	5.63/ 8.21/ 10.12	4.38/ 6.43/ 7.81

Table 2: Performance comparison of different approaches on the dataset PeMSD7.

Previous methods did not incorporate spatial topology and modeled the time series in a coarse-grained way. Differently, through modeling spatial topology of the sensors,STGCN model has achieved a significant improvement on short and mid-and-long term forecasting. The advantage of STGCN is more obvious on dataset PeMSD7 than BJER4, since the sensor network of PeMS is more complicated and structured (as illustrated in Table 3), and STGCN model can effectively utilize spatial structure to make more accurate predictions.

Dataset	Time Consumption (s)		
	STGCN(Cheb)	STGCN(1^{st})	GCGRU
PeMSD7(M)	**272.34**	271.18	3824.54
PeMSD7(L)	1926.81	**1554.37**	19511.92

Table 3: Time consumptions of training on the dataset PeMSD7

To compare the three methods based on graph convolution: GCGRU, STGCN(Cheb) and STGCN(1st), their predictions during morning peak and evening rush hours are shown in Figure 8. It is easy to observe that STGCN model captures the trend of rush hours more accurately than other methods; and it detects the ending of the rush hours earlier than others. Stemming from the efficient graph convolution and stacked temporal convolution structures, this model is capable of fast responding to the dynamic changes among the traffic network without over-reliance on historical average as most of recurrent networks do.

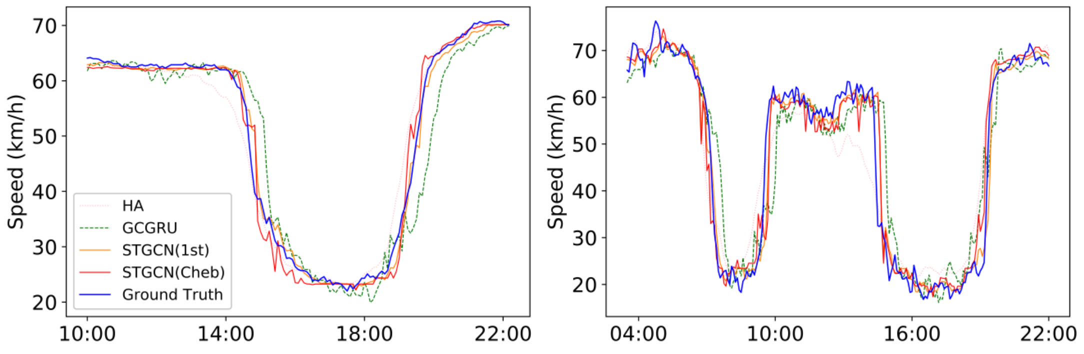


Figure 8: Speed prediction in the morning peak and evening rush hours of the dataset PeMSD7.

4 Implementation

4.1 Dataset

The dataset used to train the model is **METR-LA** dataset. This traffic dataset is widely used for traffic speed prediction. It contains traffic information collected from loop detectors in the highway of Los Angeles County. 207 sensors were selected, and the dataset contains 4 months of data collected ranging from Mar 1st 2012 to Jun 30th 2012. Combining the IDs of the sensors and the distance between them, we create an adjacency matrix which in turn is used to create a graph. (See sensor distribution below).

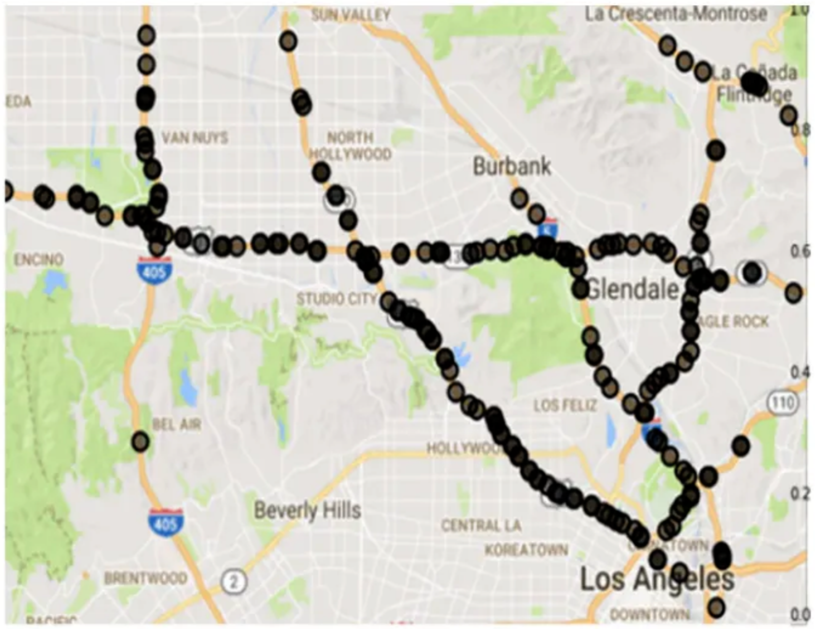
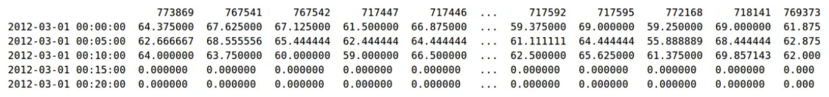


Figure 9: Sensor distribution of the METR-LA dataset

The file **metr-la.h5** contains an array of shape [34272, 207], where 34272 is total number of time steps, and 207 is number of sensors. The array contains only speed data, meaning that the GNN model uses the historical speed to predict future speed. No other features (road type, weather, holidays) are involved. The speed was recorded every 5 mins with sensors. The 207 sensors are distributed on roads within the area. See the picture above for the distribution. Speed was collected every 5 mins. So one day should have 24*(60/5)=288 records. So the data of one day is simply an array of shape [288, 207], where 288 is total time steps, and 207 is number of sensors. Since the data was collected across 4 months, there are a total number of 34272 time steps after optional data cleaning. Here below is the first 5 rows. The headers are ids of sensors and the values of content are speed.



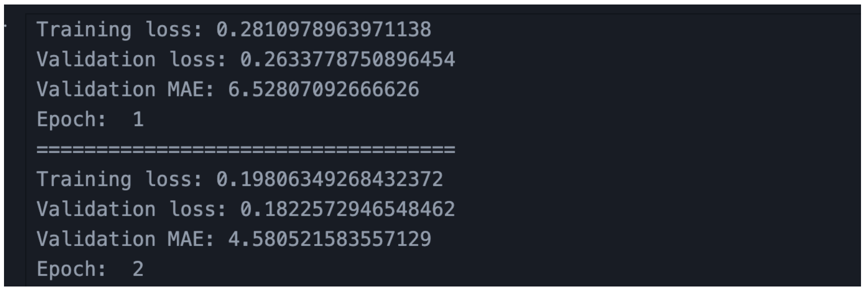
```
                        773869     767541     767542     717447     717446  ...     717592     717595     772168     718141  769373
2012-03-01 00:00:00  64.375000  67.625000  67.125000  61.500000  66.875000  ...  59.375000  69.000000  59.250000  69.000000  61.875
2012-03-01 00:05:00  62.666667  68.555556  65.444444  62.444444  64.444444  ...  61.111111  64.444444  55.888889  68.444444  62.875
2012-03-01 00:10:00  64.000000  63.750000  60.000000  59.000000  66.500000  ...  62.500000  65.625000  61.375000  69.857143  62.000
2012-03-01 00:15:00  0.000000   0.000000   0.000000   0.000000   0.000000   ...  0.000000   0.000000   0.000000   0.000000   0.000
2012-03-01 00:20:00  0.000000   0.000000   0.000000   0.000000   0.000000   ...  0.000000   0.000000   0.000000   0.000000   0.000
```

Figure 10: The header and first 5 rows of the file metr-la.h5

4.2 Model

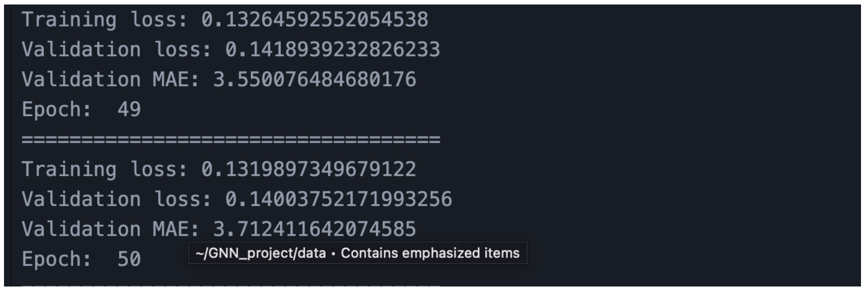
4.2.1 Training Phase

We used the **STGCN** architecture as described in the previous section to predict the future speed. The model was trained with 50 epochs improving the training and validation loss after each epoch. For example here are the first and last two training epochs (Figure 11 and Figure 12 respectively):



```
Training loss: 0.2810978963971138
Validation loss: 0.2633778750896454
Validation MAE: 6.52807092666626
Epoch:  1
===================================
Training loss: 0.19806349268432372
Validation loss: 0.1822572946548462
Validation MAE: 4.580521583557129
Epoch:  2
```

Figure 11: First two epochs

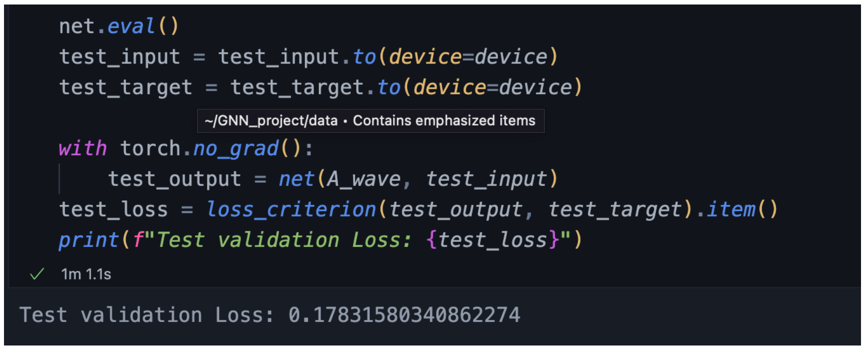


```
Training loss: 0.13264592552054538
Validation loss: 0.1418939232826233
Validation MAE: 3.550076484680176
Epoch:  49
===================================
Training loss: 0.1319897349679122
Validation loss: 0.14003752171993256
Validation MAE: 3.712411642074585
Epoch:  50
```

Figure 12: Last two epochs

4.2.2 Test Phase

After the training, we tested the model and here are results for the validation loss and mean absolute error (Figure 13 and Figure 14 respectively):

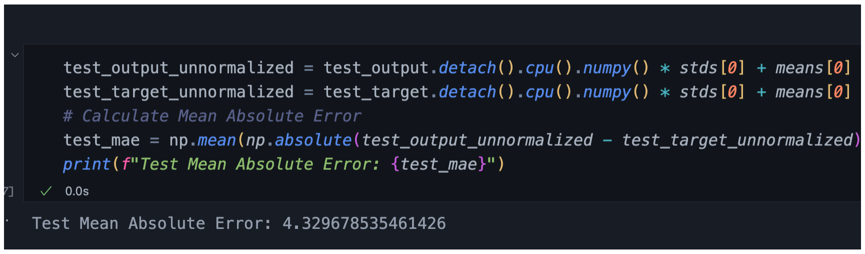


```
net.eval()
test_input = test_input.to(device=device)
test_target = test_target.to(device=device)

with torch.no_grad():
    test_output = net(A_wave, test_input)
test_loss = loss_criterion(test_output, test_target).item()
print(f"Test validation Loss: {test_loss}")
```

```
Test validation Loss: 0.17831580340862274
```

Figure 13: Test Validation Loss



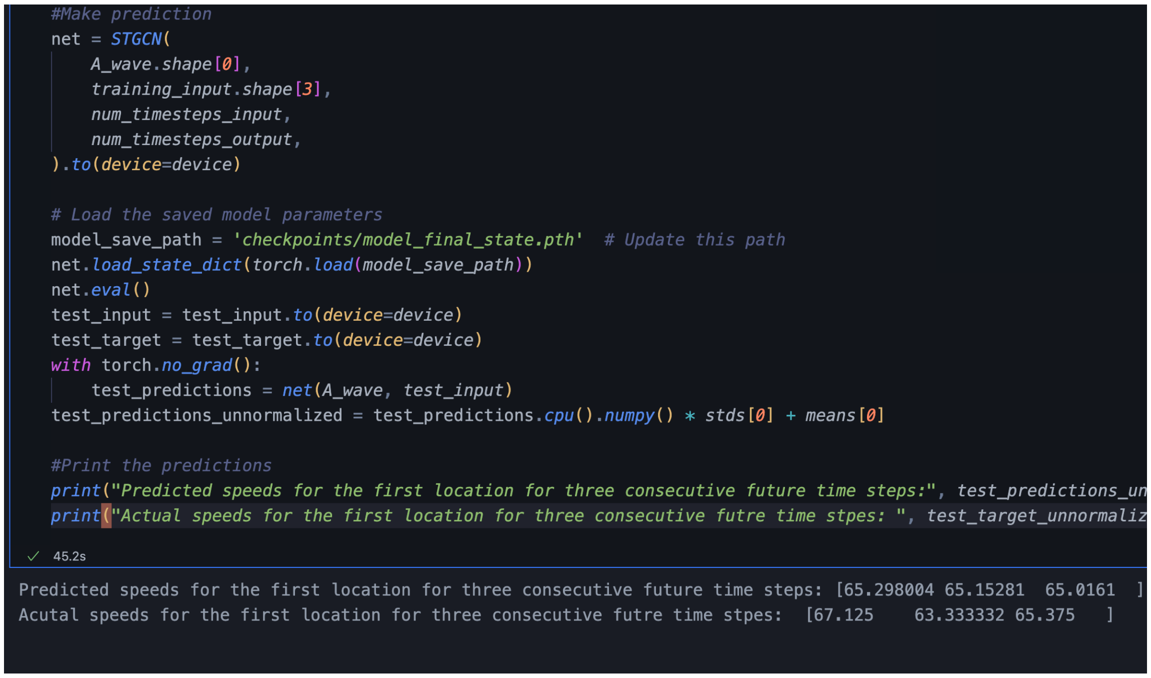
```
test_output_unnormalized = test_output.detach().cpu().numpy() * stds[0] + means[0]
test_target_unnormalized = test_target.detach().cpu().numpy() * stds[0] + means[0]
# Calculate Mean Absolute Error
test_mae = np.mean(np.absolute(test_output_unnormalized - test_target_unnormalized)
print(f"Test Mean Absolute Error: {test_mae}")
```

```
Test Mean Absolute Error: 4.329678535461426
```

Figure 14: Test Mean Absolute Error

4.2.3 Predictions

The model predicts the future speeds for a given location for a defined interval. In the dataset the speed was recorded by sensors every 5 minutes for a given location. For instance, when we pass test data to our model so it makes predictions, the output will be an array of the shape: $[i, j, k]$ where i is the the number of the sample in test data, j is the location and k represents the index of the future time step for which the prediction is made. In the figure below we tried to predict the future speed for the next 15 minutes (5 minutes interval for each) for the first location in the first data sample (Figure 15). Also showed the expected speed. As we can notice, there isn´t a big difference between the model's predicted speed and the target speed.



```
#Make prediction
net = STGCN(
    A_wave.shape[0],
    training_input.shape[3],
    num_timesteps_input,
    num_timesteps_output,
).to(device=device)

# Load the saved model parameters
model_save_path = 'checkpoints/model_final_state.pth'  # Update this path
net.load_state_dict(torch.load(model_save_path))
net.eval()
test_input = test_input.to(device=device)
test_target = test_target.to(device=device)
with torch.no_grad():
    test_predictions = net(A_wave, test_input)
test_predictions_unnormalized = test_predictions.cpu().numpy() * stds[0] + means[0]

#Print the predictions
print("Predicted speeds for the first location for three consecutive future time steps:", test_predictions_un
print("Actual speeds for the first location for three consecutive futre time stpes: ", test_target_unnormaliz
```

```
✓ 45.2s
```

```
Predicted speeds for the first location for three consecutive future time steps: [65.298004 65.15281  65.0161  ]
Acutal speeds for the first location for three consecutive futre time stpes:  [67.125    63.333332 65.375   ]
```

Figure 15: Model's predictions and target predictions

A complete implementation of the model can be found in this github repository

5 Bibliography

[1] Bing Yu, Haoteng Yin, Zhanxing Zhu, Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecastinghttps, 2018, IJCAI.