Independent Study Report On Advanced Graph-Based Deep Learning for Predictive Analytics in Urban Bike-Sharing Systems

Srushti Manjunath

University of Illinois Urbana - Champaign

Guided by: Ujjal K Mukherjee

June 3, 2024

Table of Contents

1 Introduction

1.1 Background

Urban mobility and sustainable transportation are significantly enhanced by bike-sharing systems like Citibike, which offer flexible, environmentally friendly travel options. However, the effectiveness of such systems hinges on the accurate prediction of bike demand, which varies widely due to factors like weather, urban events, and commuter habits. Traditional forecasting methods often fall short in accurately predicting such demand due to their inability to effectively incorporate the complex spatial and temporal dynamics of urban environments.

Graph convolutional networks (GCNs) present a cutting-edge approach, well-suited to address these complexities due to their proficiency in handling graph-structured data. In the context of bike-sharing systems, the network of bike stations forms a natural graph, where nodes represent stations and edges denote the historical or potential connectivity through rider flows. This setting provides a unique opportunity to leverage GCNs to predict bike demand more accurately.

1.2 Problem Statement

The challenge in predicting bike demand lies in the dynamic and non-linear nature of usage patterns, influenced by an interplay of various urban factors. Existing predictive models often do not fully exploit the spatial interconnections between stations nor adapt to the temporal variations in bike usage, leading to inefficient resource allocation and user dissatisfaction. Additionally, integrating these spatial and temporal dynamics into a coherent predictive model without excessive computational demands poses a substantial challenge.

1.3 Objectives

This study aims to implement and evaluate three different models based on advanced graph convolutional networks to predict traffic flows accurately. The objectives of this study are as follows:

1. To adapt and evaluate the efficacy of Spatio-Temporal Graph Convolutional Networks (STGCN) for modeling the complex spatio-temporal relationships inherent in bike-sharing demand.
2. To implement Attention Based Spatial-Temporal Graph Convolutional Networks (ASTGCN) with an emphasis on their ability to prioritize key spatial and temporal features critical for predicting bike demand fluctuations.
3. To explore the capabilities of a third model, focusing on its specialized handling of dynamic spatial-temporal correlations, for its suitability in predicting bike station demand under varying urban conditions.
4. To compare the performance of these models using Mean Absolute Error (MAE) and other relevant metrics, aiming to identify which model best captures the dynamics of bike demand.
5. To determine the practical implications of these findings for real-time bike-sharing management and suggest directions for future enhancements in predictive modeling techniques.

2 Literature Review

The integration of Graph Neural Networks (GNNs) into smart transportation, specifically within bike-sharing systems, has made significant strides in enhancing demand prediction and operational efficiency. This literature review delves into several key studies that leverage GNN architectures and contextual data integration to advance demand forecasting methodologies in bike-sharing networks.

In 2018, the introduction of the Graph Convolutional Neural Network with Data-driven Graph Filter (GCNN-DDGF) marked a significant advancement. This model utilizes two distinct architectures—one regular and one incorporating LSTM blocks—to predict hourly demands in large-scale networks like New York City's Citibike. The GCNN-DDGF models, particularly the recurrent version, demonstrated superior accuracy by capturing intricate correlations between stations not evident in traditional models.

The Hierarchical Prediction Model based on Two-Level Gaussian Mixture Model Clustering, studied in 2019, introduces a sophisticated clustering mechanism to predict rents and returns in bike-sharing systems. This model clusters bike stations based on geographical and usage trends to optimize the redistribution and forecasting accuracy across the network, highlighting the utility of advanced statistical models alongside GNNs in managing operational challenges.

Parallel to this, the Context Integrated Graph Neural Network (CIGNN), introduced in 2020, incorporates dynamic contextual factors such as weather into its forecasting model. Unlike traditional methods that handle temporal data in isolation, CIGNN employs a holistic approach that captures spatial, relational, and temporal dependencies simultaneously, thus offering a robust solution for multi-step ahead forecasting in varying geographical contexts.

Demand Forecasting in Bike-sharing Systems Using Multiple Spatiotemporal Fusion Network (MSTF-Net), explored in 2021, merges various spatiotemporal dynamics with external factors such as weather and time variables. MSTF-Net's use of advanced 3D convolutional and LSTM networks has proven it superior in forecasting accuracy compared to several state-of-the-art models.

The Attention-based Spatial-Temporal Graph Convolutional Network (AST-GCN), developed in 2022, stands out for its innovative use of an attention mechanism within its GNN architecture. This model excels in extracting and modeling traffic features, significantly improving the prediction of bike availability. Validated using data from the Dublinbike system, the AST-GCN showcases the importance of adaptive spatial-temporal adjacency matrices in enhancing forecasting accuracy. Spatio-temporal Neural Structural Causal Model (STNSCM), introduced in 2023, provides a causal perspective to bike flow prediction, addressing spurious correlations caused by external disturbances. This model's application of counterfactual reasoning improves its adaptability and reliability under varying environmental conditions.

Together, these studies underscore the transformative potential of GNNs and related computational models in enhancing the predictability and efficiency of bike-sharing systems. By adopting these sophisticated technologies, systems like Citibike can achieve improved operational insights and user satisfaction.

3 Methodology

3.1 Data Collection

Data about NYC Citi Bike Rentals
The data can be accessed at `https://ride.citibikenyc.com/system-data`. The dataset includes the following features:

- Start Station name (one hot vector)
- Rideable Type (classic bike, electric bike, docked bike)
- Date
  - Month
  - Day of the week
- Start time
- End time
- Trip Duration (seconds)
- Start latitude/longitude
- End latitude/longitude
- Weather (daily summaries) available at NCDC
- User Type (customer or member)
- Year of Birth
- Gender

3.2 Data Processing

I have cleaned and preprocessed the data. I have written the steps and attached the code here.

3.3 Time-series Analysis

I have uploaded the complete time-series analysis here.

- Seasonal Decomposition: Revealed an annual cyclic trend in the data with a lack of discernible patterns in the trend component.
- Autocorrelation and Partial Autocorrelation Plots: No significant lags in autocorrelation, but strong partial autocorrelation at lag zero suggested auto-regressive component.
- Dickey-Fuller Test: The dataset showed stationary behavior, confirming consistent patterns over time.
- Variance Inflation Factor (VIF): Multicollinearity concerns were addressed through feature selection, improving model robustness.

Model Development ARIMA

- Initial ARIMA Model: Achieved an Accuracy Rate of 73.3%, MAPE of 26.69%, and RMSE of 905.44.
- Optimal Forecast Horizon: Six months forecast horizon improved Accuracy Rate to 81.66
- ARIMA Forecast with 6-Month Horizon: Actual versus predicted values aligned well, capturing data patterns with confidence intervals.

Multiple Linear Regression

- Ridge Regression: Employed Ridge regression to address multicollinearity and overfitting.
- Initial Ridge Model: Achieved an Accuracy Rate of 73.23%, R-squared of 82.60%, MAPE of 26.77%, and RMSE of 893.52.
- Feature Importance: 'Bikecountslog' and 'pandemicperiod' were influential predictors in the model.
- Second Ridge Model: Accuracy Rate of 73.28% and RMSE of 893.52, relying on twelve predictors.
- Optimal Forecast Horizon: 6-month horizon improved R-squared to 93.97%, MAPE to 17.66%, and RMSE to 494.17.
- Residual Analysis: Residuals exhibited deviation in extreme observations, indicating room for improvement.

These findings provide insights into ridership patterns, influential variables, predictive models' performance, and the impact of forecast horizons. The combination of data exploration, time-series analysis, and model development offers a comprehensive understanding of Citi Bike ridership trends and predictive accuracy.

4 Overview of Each Paper

4.1 Spatio-temporal Graph Convolutional Networks for Traffic Forecasting

4.1.1 Summary

This paper introduces a novel deep learning framework called Spatio-Temporal Graph Convolutional Networks (STGCN) designed for traffic forecasting. Traditional methods often fail to capture the complex, nonlinear dynamics of traffic flow, particularly over mid to long-term periods, and typically overlook the spatial and temporal dependencies inherent in traffic data. STGCN addresses these challenges by utilizing a graph-based approach rather than traditional convolutional or recurrent units, allowing for the efficient modeling of traffic networks as graphs and enabling faster training speeds with fewer parameters.
STGCN consists of spatio-temporal convolutional blocks that integrate graph convolutional layers to handle spatial features and gated convolutional layers to manage temporal dynamics, effectively capturing the interdependencies in traffic data.

4.1.2 Theoretical Background

The theoretical foundation of STGCN is rooted in graph theory and convolutional neural networks, specifically adapted to handle the spatio-temporal data characteristic of traffic networks.

1. **Graph Theory and Traffic Networks:**
    - **Graph Representation of Traffic:** Traffic networks are modeled as graphs where intersections and stretches of road are represented as nodes, and the roads connecting these nodes as edges. This graph representation allows the application of graph-based algorithms to analyze and predict traffic flow dynamics.
    - **Weighted Graphs:** The traffic network is considered a weighted graph where weights on edges could represent distance, traffic capacity, or speed limits, which influence the flow of traffic and consequently, the predictions.
2. **Graph Convolutional Networks (GCNs):**
    - **Extension to Non-Euclidean Data:** Traditional convolutional networks are effective for Euclidean data (like images and videos), but traffic data represented as graphs are non-Euclidean. GCNs extend the capabilities of CNNs to graph-structured data.
    - **Spectral Graph Convolutions:** This approach involves the Fourier transform of graph signals, allowing convolutions to be defined in the spectral domain using the eigenvectors of the graph Laplacian. It translates the convolution operation into a multiplication in the spectral domain, which can be computationally intensive but is powerful for feature extraction from structured data.
    - **Spatial Graph Convolutions:** Instead of working in the spectral domain, this method applies convolutions directly on the graph nodes and their neighbors, preserving locality and improving computational efficiency. It treats convolutions as a neighborhood aggregation or message-passing process, where node features are updated by aggregating features of their neighbors.
3. **Temporal Feature Extraction:**

- **Gated Convolutional Layers:** To capture temporal dependencies, STGCN uses gated convolutional layers which include gating mechanisms (similar to those in GRUs and LSTMs) to control the flow of information. These layers help the network focus on relevant temporal features without the vanishing gradient problem common in traditional recurrent networks.
- **1-D Convolutional Layers for Temporal Data:** These layers process temporal sequences across the traffic network graph, capturing patterns over time. The use of 1-D convolutions allows for faster training and parallel processing compared to recurrent models.

4. **Spatio-temporal Blocks:**

   - **Integrated Spatial and Temporal Processing:** The core of STGCN consists of spatio-temporal blocks where each block contains layers designed for spatial processing (using graph convolutions) sandwiched between layers for temporal processing (using gated convolutions). This design allows STGCN to effectively capture and integrate both spatial and temporal dependencies.
   - **Efficiency and Scalability:** By leveraging convolutional structures both in spatial and temporal dimensions, STGCN reduces the number of parameters compared to fully connected models and improves computational efficiency, making it scalable to large traffic networks.

4.1.3 Mathematical Formulations

The STGCN framework integrates graph-based spatial analysis with temporal convolution operations to effectively model and predict traffic dynamics. Below are the detailed mathematical formulations used in STGCN to capture both spatial and temporal dependencies:

1. **Graph Convolutional Networks**

   - **Spectral Graph Convolutions:** The spectral formulation of graph convolutions leverages the eigen-decomposition of the graph Laplacian, defined as L=D-A, where A is the adjacency matrix of the graph and D is the diagonal degree matrix with $D_{ii} = \sum_j A_{ij}$.
   - **Fourier Transform of Graph Signals:** The graph Fourier transform of a signal x on a graph is given by:
   $$\hat{x} = U^T x$$
   where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I - D^{-1/2} A D^{-1/2}$, and x is a signal defined at the nodes of the graph.
   - **Convolution Operation:** In the spectral domain, the convolution of a signal x with a filter g_θ (parameterized by θ) is defined as:
   $$y = g_\theta * x = U g_{\theta_0}(U^T x) = U g_{\theta_0}(\Lambda) U^T x$$
   where Λ is the diagonal matrix of eigenvalues of L, and $g_\theta(\Lambda)$ is a function of these eigenvalues, typically parameterized as a polynomial for computational efficiency.
   - **Spectral Graph Convolutions:** Spatial graph convolutions directly operate in the node space and typically use a localized first-order approximation:
   $$y_i = \theta \sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} x_j$$

where $\mathcal{N}(i)$ denotes the neighbors of node i including i itself, x_j is the feature vector of the j-th node, θ is a learnable parameter, and c_{ij} is a normalization constant, often taken as $c_{ij} = \sqrt{d_i d_j}$, with d_i being the degree of node i.

2. **Temporal Convolution Layers**

   - **Temporal Convolution Operation:**

$$Y(t) = \sum_{k=0}^{K-1} \theta_k X(t-k)$$

   where $X(t)$ is the input feature at time t, $Y(t)$ is the output feature at time t, θ_k are the weights of the convolution filter, and K is the size of the temporal kernel.

3. **Spatio-Temporal Convolutional Blocks**
   Each spatio-temporal block in STGCN combines both spatial and temporal convolutions:
   Block Structure:

   - **Temporal Gated Convolution:** Temporal dependencies are modeled using gated linear units (GLU) in combination with 1-D convolutions:

$$Y = (W * X) \otimes \sigma(V * X)$$

   where W and V are convolutional filters, $*$ denotes the convolution operation, σ is the sigmoid activation function, and $\otimes$ denotes element-wise multiplication.

   - **Spatial Graph Convolution:** Applied between the temporal gated convolutions to integrate spatial information:

$$Z = \text{ReLU}(U g_\theta (U^T Y))$$

   where Y is the output from the previous temporal convolution, and g_θ is the graph convolution filter applied in the spectral domain.

   Combining Operations:
   The outputs from sequential layers within a block are typically combined using residual connections to enhance training stability and convergence:

$$H^{(l+1)} = H^{(l)} + \text{ST-Conv}(H^{(l)})$$

   where $H^{(l)}$ is the input to the l-th block, and ST-Conv represents the combined operations of spatial and temporal convolutions within the block.

These mathematical formulations enable STGCN to efficiently capture and leverage the complex interdependencies between spatial and temporal aspects of traffic data, leading to improved forecasting accuracy.

4.1.4 Adaptation of STGCN for Bike Demand Prediction

This application leverages the core capabilities of STGCN in handling spatio-temporal data, which is crucial in traffic flow and bike-sharing demand prediction.

Graph Representation

- **Nodes:** In the context of bike-sharing demand prediction, each node in the graph represents a bike station.
- **Edges:** The edges between these nodes can represent the physical distance between stations, or potentially other relationships such as biking paths, connectivity, or common traffic routes.
- **Edge Features:** The distance between stations, used as edge features, can influence the bike demand forecasting by indicating the likelihood of people choosing to bike between two locations based on distance.

Spatio-Temporal Dynamics

- **Spatial Component:** The spatial configuration of the stations influences demand patterns. Stations closer together may have higher interaction and shared demand characteristics, especially during peak commuting hours or around special events. By using graph convolutions, STGCN can capture these spatial dependencies, where the demand at one station may be affected by the conditions at nearby stations.
- **Temporal Component:** Bike demand is also heavily influenced by temporal factors such as time of day, day of the week, season, and weather conditions. The temporal convolution layers in STGCN allow it to learn from past demand patterns over time and predict future demands accurately.

To apply STGCN to bike demand forecasting, the model would be trained on historical data from the bike-sharing system, including:

- **Demand data:** Historical checkout and return counts at each station.
- **Temporal data:** Time stamps of bike checkouts/returns.
- **Spatial data:** Locations of the stations and distances between them.

4.1.5 Implementation Details

The implementation can be divided into following key parts:

1. **Model Definition**
   Fully Connected Layer: This layer is a basic fully connected layer using a 1x1 convolution, often used to reduce dimensionality.

```
    class FullyConnLayer(nn.Module):
        def __init__(self, c):
            super(FullyConnLayer, self).__init__()
            self.conv = nn.Conv2d(c, 1, 1)

        def forward(self, x):
            return self.conv(x)
```

   Output Layer: This module defines the output layer of the STGCN model, incorporating temporal convolutions and layer normalization.

```
class OutputLayer(nn.Module):
def __init__(self, c, T, n):
    super(OutputLayer, self).__init__()
    self.tconv1 = nn.Conv2d(c, c, (T, 1), 1, dilation=1, padding
        ↪ =(0,0))
    self.ln = nn.LayerNorm([n, c])
    self.tconv2 = nn.Conv2d(c, c, (1, 1), 1, dilation=1, padding
        ↪ =(0,0))
    self.fc = FullyConnLayer(c)

def forward(self, x):
    x_t1 = self.tconv1(x)
    x_ln = self.ln(x_t1.permute(0, 2, 3, 1)).permute(0, 3, 1, 2)
    x_t2 = self.tconv2(x_ln)
    return self.fc(x_t2)
```

Main Model: The TrafficModel class builds the entire model architecture by stacking multiple STConv blocks followed by an OutputLayer, customizing for graph-based spatial-temporal data handling.

```
class TrafficModel(torch.nn.Module):
def __init__(self, device, num_nodes, channel_size_list,
    ↪ num_layers,
                kernel_size, K, window_size, \
                normalization = 'sym', bias = True):
# num_nodes = number of nodes in the input graph
# channel_size_list =  2d array representing feature dimensions
    ↪ throughout the model
# num_layers = number of STConv blocks
# kernel_size = length of the temporal kernel
# K = size of the chebyshev filter for the spatial convolution
# window_size = number of historical time steps to consider
    super(TrafficModel, self).__init__()
    self.layers = nn.ModuleList([])
    for l in range(num_layers):
        input_size, hidden_size, output_size = \
        channel_size_list[l][0], channel_size_list[l][1], \
        channel_size_list[l][2]
        self.layers.append(STConv(num_nodes, input_size,
            ↪ hidden_size, \
                                output_size, kernel_size, K, \
                                normalization, bias))
    # add output layer
    self.layers.append(OutputLayer(channel_size_list[-1][-1], \
                                window_size - 2 * num_layers *
                                    ↪ (kernel_size - 1), \
                                num_nodes))
    for layer in self.layers:
        layer = layer.to(device)

def forward(self, x, edge_index, edge_weight):
    for layer in self.layers[:-1]:
      x = layer(x, edge_index, edge_weight)
```

```
        out_layer = self.layers[-1]
        x = x.permute(0, 3, 1, 2)
        x = out_layer(x)
        return x
```

2. **Data Preparation**

   The datatransform function is used to prepare the data into a suitable format for training the STGCN model. This function organizes historical data into input features and labels for the model, catering to the network's need for historical sequences.

```
    def data_transform(data, n_his, n_pred, device):
    # data = slice of features(number of bikes at the station) matrix
    # n_his = number of historical speed observations to consider
    # n_pred = number of time steps in the future to predict
        num_nodes = data.shape[1]
        num_obs = len(data) - n_his - n_pred
        x = np.zeros([num_obs, n_his, num_nodes, 1])
        y = np.zeros([num_obs, num_nodes])
        obs_idx = 0
        for i in range(num_obs):
            head = I
            tail = i + n_his
            x[obs_idx, :, :, :] = data[head: tail].reshape(n_his,
               ↪ num_nodes, 1)
            y[obs_idx] = data[tail + n_pred - 1]
            obs_idx += 1
        return torch.Tensor(x).to(device), torch.Tensor(y).to(device)
```

3. **Model Training**

   Training is executed over a specified number of epochs, using an Adam optimizer and MSE loss function. The model learns to minimize the difference between predicted and actual bike demand. Each epoch iterates over the entire dataset, updating the model parameters to reduce the training loss.

```
    for epoch in tqdm(range(1, num_epochs + 1), desc = 'Epoch',
       ↪ position = 0):
        l_sum, n = 0.0, 0
        model.train()
        for x, y in tqdm(train_iter, desc = 'Batch', position = 0):
            y_pred = model(x.to(device), edge_index, edge_weight).view
               ↪ (len(x), -1)
            l = loss(y_pred, y)
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            l_sum += l.item() * y.shape[0]
            n += y.shape[0]
        #validation loss
        val_loss = evaluate_model(model, loss, val_iter, edge_index,
           ↪ edge_weight, device)
        if val_loss < min_val_loss:
            min_val_loss = val_loss
            torch.save(model.state_dict(), model_save_path)
        print("epoch", epoch, ", train loss:", l_sum / n, ",
            ↪ validation loss:", val_loss)
```

4. **Model Evaluation**
   After training, the model is evaluated on a validation and test set to ensure it generalizes well to new, unseen data. This function calculates the loss over the dataset without making any parameter updates.

```
    def evaluate_model(model, loss, data_iter, edge_index, edge_weight
        ↪ , device):
        model.eval()
        l_sum, n = 0.0, 0
        with torch.no_grad():
            for x, y in data_iter:
                y_pred = model(x.to(device), edge_index, edge_weight).
                    ↪ view(len(x), -1)
                l = loss(y_pred, y)
                l_sum += l.item() * y.shape[0]
                n += y.shape[0]
            return l_sum / n

    def evaluate_metric(model, data_iter, scaler, edge_index,
        ↪ edge_weight, device):
        model.eval()
        epsilon = 1e-6
        with torch.no_grad():
            mae, mape, mse = [], [], []
            for x, y in data_iter:
                y = scaler.inverse_transform(y.cpu().numpy()).reshape
                    ↪ (-1)
                y_pred = scaler.inverse_transform(model(x.to(device),
                    ↪ edge_index, edge_weight).view(len(x), -1).cpu().
                    ↪ numpy()).reshape(-1)
                d = np.abs(y - y_pred)
                mae += d.tolist()
                mape += (d / (y+epsilon)).tolist()
                mse += (d ** 2).tolist()
            MAE = np.array(mae).mean()
            RMSE = np.sqrt(np.array(mse).mean())
            return MAE, RMSE
```

5. **Prediction Generation**
   Finally, the model is used to predict future bike demand based on historical data. This function generates predictions and potentially transforms them back to the original scale using an inverse of the initial data scaling.

```
    def get_predictions(model, pred_iter, scaler, edge_index,
        ↪ edge_weight, num_nodes, device):
        model.eval()
        with torch.no_grad():
            for x, y in pred_iter:
                y = scaler.inverse_transform(y.cpu().numpy()).reshape
                    ↪ (-1)
                y_pred = scaler.inverse_transform(model(x.to(device),
                    ↪ edge_index, edge_weight).view(len(x)-1).cpu().
                    ↪ numpy()).reshape(-1)
                y_pred = y_pred.reshape(-1, num_nodes)
            return y, y_pred
```

4.1.6 Results

In this study, the Spatio-Temporal Graph Convolutional Network (STGCN) model was utilized to predict bike demand across various locations, demonstrating a Mean Absolute Error (MAE) of 0.767 bikes and a Root Mean Square Error (RMSE) of 1.182 bikes. These metrics indicate a moderate level of prediction accuracy, with the MAE suggesting that the model's predictions are, on average, less than one bike away from actual values. The slightly higher RMSE points towards the presence of some larger errors, which could be due to outliers or specific time periods where the model's predictions deviate more significantly from the true values.

4.2 Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting

4.2.1 Summary

This paper introduces an innovative model known as ASTGCN (Attention-based Spatial-Temporal Graph Convolutional Network). This model aims to enhance traffic flow forecasting by addressing the challenges associated with the nonlinear and complex nature of traffic data. ASTGCN leverages a novel spatial-temporal attention mechanism to dynamically capture and model the spatial-temporal correlations within the data.

4.2.2 Theoretical Background

1. **Graph Convolutional Networks (GCNs)**

   - **Graph Theory Fundamentals:** A graph G = (V, E) consists of a set of vertices V and edges E, where edges represent relationships or interactions between vertices. Traffic networks can be represented as graphs where intersections and sensors are vertices, and roads are edges connecting these vertices.
   - **Spectral Decomposition:** The Laplacian can be decomposed into $L = U\Lambda U^T$, where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues. This decomposition is fundamental for defining graph Fourier transforms.
   - **Graph Convolutions:** Graph convolution allows filtering signals on graphs, defined in the spectral domain by $g_\theta * x = U g_\theta(\Lambda) U^T x$, where g_θ is a function of the Laplacian's eigenvalues, typically parameterized for computational efficiency.

2. **Attention Mechanisms** Attention mechanisms dynamically select and emphasize certain parts of the data, improving model interpretability and performance by focusing on relevant information.

   - **Spatial Attention:** In traffic networks, the relevance of one sensor/node to another can vary dynamically. Spatial attention allows the model to learn these varying dependencies by computing attention scores that modify the adjacency matrix of the graph convolution dynamically.

- **Temporal Attention:** Similar to spatial attention but applied over time, this mechanism identifies and emphasizes critical time steps that have more predictive power for future states, adapting to the changes in temporal patterns such as daily or weekly cycles.

3. **Multi-Component Framework for Spatial-Temporal Data**
   - **Decomposition of Temporal Data:** The paper introduces a multi-component approach where different components of the model capture distinct temporal dependencies (recent, daily-periodic, weekly-periodic). Each component uses both spatial and temporal attention mechanisms to focus on different aspects of the traffic data, ensuring that all relevant temporal patterns are captured.
   - **Integration and Fusion:** The outputs from the various components are not treated equally; they are fused together using learned weights that reflect the importance of each component's output in predicting future traffic states. This weighted fusion helps to synthesize a comprehensive prediction that takes into account short-term fluctuations, as well as daily and weekly patterns.

The theoretical framework of ASTGCN illustrates a sophisticated approach to handling spatial-temporal data by integrating advanced techniques from graph theory, signal processing, and neural networks. By applying both spatial and temporal attention mechanisms within a graph convolutional network structure, ASTGCN can effectively model complex interactions in traffic data, leading to more accurate and robust predictions. This integration not only enhances performance but also provides a framework for interpreting the dynamic influences within the network.

4.2.3 Mathematical Formulations

The mathematical formulations presented in this paper focus on graph convolutions, attention mechanisms, and the integration of these components within a spatio-temporal model. Here's a detailed breakdown of these formulations:

1. **Graph Convolutional Networks (GCNs)**
   The core of graph convolutions in the ASTGCN framework leverages the spectral approach, using the graph Laplacian matrix. The Laplacian matrix L is defined as:

   $$L = D - A$$

   where A is the adjacency matrix of the graph G representing the traffic network, and D is the degree matrix, which is diagonal and contains the sum of the weights of the edges connected to each vertex.
   Normalized Laplacian:

   $$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

   where I_N is the identity matrix.
   Graph Fourier Transform and **Graph Convolution** are defined using the eigen-decomposition of L: $L = U \Lambda U^T$, where U (matrix of eigenvectors) represents the graph Fourier basis, and Λ (diagonal matrix containing eigenvalues) represents the frequencies in the graph domain.
   The graph convolution of a signal x with a filter g_θ parameterized by θ is given by:

   $$g_\theta * x = U g_\theta(\Lambda) U^T x$$

   where $g_\theta(\Lambda)$ is typically implemented using Chebyshev polynomials for efficiency, avoiding the need to compute U explicitly.

2. **Attention Mechanisms**

   - **Spatial Attention:** Spatial attention dynamically adjusts the weights of the adjacency matrix, allowing the model to focus on significant nodes (locations) based on the traffic conditions.
     The spatial attention weights are computed as follows:

     $$S = V_s \cdot \sigma((X^{(r-1)}hW_1)W_2(W_3X^{(r-1)}h)^T + b_s)$$

     $$S'_{ij} = \frac{\exp(S_{ij})}{\sum_{j=1}^{N} \exp(S_{ij})}$$

     where σ is the activation function (e.g., sigmoid), $X^{(r-1)}h$ represents the input features at layer $r-1$, and V_s, W_1, W_2, W_3, b_s are trainable parameters.

   - **Temporal Attention:** Temporal attention focuses on important time steps, enhancing the model's ability to capture relevant temporal dynamics.
     The temporal attention weights are computed as:

     $$E = V_e \cdot \sigma(((X^{(r-1)}h)^T U_1)U_2(U_3X^{(r-1)}h) + b_e)$$

     $$E'_{ij} = \frac{\exp(E_{ij})}{\sum_{j=1}^{T_{r-1}} \exp(E_{ij})}$$

     where V_e, U_1, U_2, U_3, and b_e are trainable parameters.

3. **Integration and Fusion in the ASTGCN Model**
   The outputs from the different components (recent, daily, weekly) of the ASTGCN model are combined to form the final traffic flow prediction:

   $$\hat{Y} = W_h \odot \hat{Y}_h + W_d \odot \hat{Y}_d + W_w \odot \hat{Y}_w$$

   where $\odot$ denotes element-wise multiplication, and W_h, W_d, and W_w are trainable parameters that determine the contribution of each temporal component (recent, daily, weekly) to the final output.

The mathematical formulations in the ASTGCN model utilize these components to capture both the spatial and temporal dependencies in traffic flow data effectively, allowing it to adapt dynamically to changes in traffic conditions and thus providing accurate traffic flow predictions.

4.2.4 Adaptation of ASTGCN for Bike Demand Prediction

1. **Graph Construction:**

   - Define nodes as bike stations.
   - Set edges based on distances or historical movement patterns between stations (e.g., frequent bike rentals between specific stations).
   - Use additional features like station capacity or type of area (residential, commercial) as node features.

2. **Temporal Data Integration:**

- Incorporate historical bike demand data for each station into the model, formatted in time series.
- Include other temporal features that might affect demand, such as weather data, holidays, and events.

3. **Model Training:**
    - Train the ASTGCN model on historical data, where the output is the predicted bike demand for future time slots.
    - Use the spatial-temporal blocks to process the graph data, applying both spatial and temporal attention to learn the intricate dependencies.

4. **Deployment for Real-Time Predictions:**
    - Implement the model in a real-time setting where it receives continuous data updates and provides demand predictions, facilitating dynamic management of bike availability.
    - Utilize the model's predictions to optimize bike redistribution efforts, maintenance scheduling, and promotional strategies targeting increased bike usage.

4.2.5 Implementation Details

Describe how you implemented the models and algorithms. Include snippets of your code or pseudocode to illustrate key parts of the implementation.

Data Preprocessing

The key steps that I have taken to preprocess data for implementing ASTGCN on the Citibike dataset are as follows:

1. Loaded the necessary distance matrix and ride data that contains detailed records of all the bike rides for relevant JC stations.
2. Adjusted time range to focus on rides within the specified period, ensuring consistency in temporal data analysis.
3. Constructed an adjacency matrix indicating the number of rides between each pair of stations, which helps the model understand traffic flow within the network.
4. Created time series data frames for inbound and outbound rides, capturing the flow of bikes at 5-minute intervals.
5. Segmented data based on different time units such as recent hours, daily, and weekly intervals to capture varying influences on future events. This helps in understanding how past events at different scales affect future outcomes. For each forecast point, relevant historical segments are identified and extracted to create samples that include necessary contextual information from the past. These samples serve as the training and testing data for the model.
6. The extracted samples are standardized and organized into structured datasets, divided into training, validation, and testing sets to facilitate different phases of model development, including training, tuning, and evaluation.

Intuition behind the above-mentioned data segmentation
We have 42 nodes, and each node (station) has 2 features - the number of bikes leaving the station and the number of bikes entering the station. Each data point is collected every 30-minute interval. The idea is to sample data from the past segments of the data, that are predictors for the future segment. If we want to predict the bike counts in the next hour (e.g., Thursday 8:00 AM), we can use:

1. **The Recent Segment:** We use the last two hours. Based on the traffic from 6:00 to 8:00 AM, we will predict the traffic of 8:00 AM.
   Intuition: The formation and dispersion of traffic congestion are gradual. So, the just past traffic flows inevitably have an influence on the future traffic flows.

2. **The Daily-Periodic Segment:** We use the same hour in the last week and the same hour yesterday and the day before. Based on the traffic at 8:00 AM on Tuesday and Wednesday, we will predict the traffic at 8:00 AM on Thursday.
   Intuition: Due to the regular daily routine of people, traffic data may show repeated patterns, such as the daily morning peaks. The purpose of the daily-period component is to model the daily periodicity of traffic data.

3. **The Weekly-Periodic Segment:** We use the same hour in the last week and the same hour in the week before the last week. Based on the traffic at 8:00 AM on the previous Thursdays, we will predict the traffic at 8:00 AM.
   Intuition: Usually, the traffic patterns on Mondays have a certain similarity with the traffic patterns on Mondays in history but may be greatly different from those on weekends. Thus, the weekly-period component is designed to capture the weekly periodic features in traffic data.

ASTGCN Model The implementation of the model can be divided into the following key parts:

1. **Data Loading**
   Load and preprocess time-series data from a file for training, validation, and testing in a machine learning model for traffic or network predictions. The code reads data, splits it into respective sets, and formats it into tensors that are suitable for input into neural network models.

```
    def load_graphdata(graph_signal_matrix, num_of_hours, num_of_days,
       ↪  num_of_weeks, batch_size, shuffle=True, DEVICE = torch.
       ↪ device('cuda:0')):
        '''
        :param graph_signal_matrix_filename: str
        :param num_of_hours: int
        :param num_of_days: int
        :param num_of_weeks: int
        :param DEVICE:
        :param batch_size: int
        :return:
        three DataLoaders, each dataloader contains:
        test_x_tensor: (B, N_nodes, in_feature, T_input)
        test_decoder_input_tensor: (B, N_nodes, T_output)
        test_target_tensor: (B, N_nodes, T_output)
        '''
```

```
file = os.path.basename(graph_signal_matrix_filename).split('.
    ↪ ')[0]
filename = r'C:\Users\srush\bike share model\JC_ASTGCN\
    ↪ your_data_r1_d0_w0_astcgn'
print('load file:', filename)

file_data = np.load(filename + '.npz')
train_x = file_data['train_x']  # (10181, 307, 2, 12)
train_x = train_x[:, :, 0:1, :]
train_target = file_data['train_target']  # (10181, 307, 12)

val_x = file_data['val_x']
val_x = val_x[:, :, 0:1, :]
val_target = file_data['val_target']

test_x = file_data['test_x']
test_x = test_x[:, :, 0:1, :]
test_target = file_data['test_target']

mean = file_data['mean'][:, :, 0:1, :]  # (1, 1, 2, 1)
std = file_data['std'][:, :, 0:1, :]  # (1, 1, 2, 1)

# train_loader
train_x_tensor = torch.from_numpy(train_x).type(torch.
    ↪ FloatTensor).to(DEVICE)  # (B, N, F, T)
train_target_tensor = torch.from_numpy(train_target).type(
    ↪ torch.FloatTensor).to(DEVICE)  # (B, N, T)
train_dataset = torch.utils.data.TensorDataset(train_x_tensor,
    ↪  train_target_tensor)
train_loader = torch.utils.data.DataLoader(train_dataset,
    ↪ batch_size=batch_size, shuffle=shuffle)

# val_loader
val_x_tensor = torch.from_numpy(val_x).type(torch.FloatTensor)
    ↪ .to(DEVICE)  # (B, N, F, T)
val_target_tensor = torch.from_numpy(val_target).type(torch.
    ↪ FloatTensor).to(DEVICE)  # (B, N, T)
val_dataset = torch.utils.data.TensorDataset(val_x_tensor,
    ↪ val_target_tensor)
val_loader = torch.utils.data.DataLoader(val_dataset,
    ↪ batch_size=batch_size, shuffle=False)

# test_loader
test_x_tensor = torch.from_numpy(test_x).type(torch.
    ↪ FloatTensor).to(DEVICE) # (B, N, F, T)
test_target_tensor = torch.from_numpy(test_target).type(torch.
    ↪ FloatTensor).to(DEVICE)  # (B, N, T)
test_dataset = torch.utils.data.TensorDataset(test_x_tensor,
    ↪ test_target_tensor)
test_loader = torch.utils.data.DataLoader(test_dataset,
    ↪ batch_size=batch_size, shuffle=False)

# print
```

```
        print('train:', train_x_tensor.size(), train_target_tensor.
            ↪ size())
        print('val:', val_x_tensor.size(), val_target_tensor.size())
        print('test:', test_x_tensor.size(), test_target_tensor.size()
            ↪ )

        return train_loader, train_target_tensor, val_loader,
            ↪ val_target_tensor, test_loader, test_target_tensor, mean
            ↪ , std
```

Also, created a weighted adjacency matrix based on distances between the stations.

2. **Network Architecture and Initialisation**
In the implementation of the ASTGCN model for predicting bike demand, the network is configured with two primary blocks, each designed to capture complex spatial and temporal dependencies within the data. Each block integrates temporal and spatial attention mechanisms, enhanced by Chebyshev convolution to leverage the graph structure of the data. This is supplemented by convolution operations across the temporal dimension and layer normalization to stabilize learning. The model is parameterized with filters and strides to optimize the processing of time-series data related to bike station nodes, ensuring robust feature extraction for accurate future demand prediction.

```
    nb_block = 2
    in_channels = 1
    K = 3
    nb_chev_filter = 64
    nb_time_filter = 64
    time_strides = num_of_hours
    num_for_predict = 12
    len_input = 12
    net = ASTGCN( nb_block, in_channels, K, nb_chev_filter,
        ↪ nb_time_filter, time_strides, num_for_predict, len_input,
        ↪ num_of_vertices).to(DEVICE)
    print(net)

    ASTGCN(
    (_blocklist): ModuleList(
        (0): ASTGCNBlock(
          (_temporal_attention): TemporalAttention()
          (_spatial_attention): SpatialAttention()
          (_chebconv_attention): ChebConvAttention(1, 64, K=3,
              ↪ normalization=None)
          (_time_convolution): Conv2d(64, 64, kernel_size=(1, 3),
              ↪ stride=(1, 1), padding=(0, 1))
          (_residual_convolution): Conv2d(1, 64, kernel_size=(1, 1),
              ↪ stride=(1, 1))
          (_layer_norm): LayerNorm((64,), eps=1e-05,
              ↪ elementwise_affine=True)
        )
        (1): ASTGCNBlock(
          (_temporal_attention): TemporalAttention()
          (_spatial_attention): SpatialAttention()
          (_chebconv_attention): ChebConvAttention(64, 64, K=3,
              ↪ normalization=None)
```

```
            (_time_convolution): Conv2d(64, 64, kernel_size=(1, 3),
               ↪ stride=(1, 1), padding=(0, 1))
            (_residual_convolution): Conv2d(64, 64, kernel_size=(1, 1),
               ↪ stride=(1, 1))
            (_layer_norm): LayerNorm((64,), eps=1e-05,
               ↪ elementwise_affine=True)
          )
        )
        (_final_conv): Conv2d(12, 12, kernel_size=(1, 64), stride=(1, 1)
           ↪ )
      )
```

3. **Setting Up the Network and Optimizer**
   Initialize the ASTGCN network with specified parameters and configure the Adam optimizer for training. It prints the network's parameters and the total count, giving an overview of the model's complexity and the computation resource allocation.

```
    learning_rate = 0.001
    optimizer = optim.Adam(net.parameters(), lr=learning_rate)
    print('Net\'s state_dict:')
    total_param = 0
    for param_tensor in net.state_dict():
        print(param_tensor, '\t', net.state_dict()[param_tensor].size
           ↪ (), '\t', net.state_dict()[param_tensor].device)
        total_param += np.prod(net.state_dict()[param_tensor].size())
    print('Net\'s total params:', total_param)
```

4. **Loss Function Configuration**
   Define different loss functions, including masked MAE, to handle missing or incomplete data effectively during training and validation phases. It selects the appropriate loss function based on configuration.

```
    def masked_mae(preds, labels, null_val=np.nan):
        if np.isnan(null_val):
            mask = ~torch.isnan(labels)
        else:
            mask = (labels != null_val)
        mask = mask.float()
        mask /= torch.mean((mask))
        mask = torch.where(torch.isnan(mask), torch.zeros_like(mask),
           ↪ mask)
        loss = torch.abs(preds - labels)
        loss = loss * mask
        loss = torch.where(torch.isnan(loss), torch.zeros_like(loss),
           ↪ loss)
        return torch.mean(loss)
```

5. **Training and Validating the Model**
   Build training and validation loop and save the model parameters when a new best validation loss is found.

```
    # train model
    for epoch in range(20):
        params_filename = os.path.join('./', 'epoch_%s.params' % epoch
           ↪ )
        masked_flag = 1
        if masked_flag:
            val_loss = compute_val_loss_mstgcn(net, val_loader,
               ↪ criterion_masked, masked_flag,missing_value,sw,
               ↪ epoch,edge_index_data)
        else:
            val_loss = compute_val_loss_mstgcn(net, val_loader,
               ↪ criterion, masked_flag, missing_value, sw, epoch,
               ↪ edge_index_data)

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_epoch = epoch
            torch.save(net.state_dict(), params_filename)
            print('save parameters to file: %s' % params_filename)

        net.train()  # ensure dropout layers are in train mode

        for batch_index, batch_data in enumerate(train_loader):
            encoder_inputs, labels = batch_data   # encoder_inputs
               ↪ torch.Size([32, 307, 1, 12])  label torch.Size([32,
               ↪ 307, 12])
            optimizer.zero_grad()
            outputs = net(encoder_inputs, edge_index_data) # torch.
               ↪ Size([32, 307, 12])

            if masked_flag:
                loss = criterion_masked(outputs, labels,missing_value)
            else :
                loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            training_loss = loss.item()
            global_step += 1
            sw.add_scalar('training_loss', training_loss, global_step)

            if global_step % 200 == 0:
                print('global step: %s, training loss: %.2f, time: %.2
                   ↪ fs' % (global_step, training_loss, time() -
                   ↪ start_time))
```

6. **Testing and Reporting**
   After training, the model is evaluated on the test set to measure its performance.

```
    net.train(False)  # ensure dropout layers are in evaluation mode
    with torch.no_grad():
        test_loader_length = len(test_loader)  # nb of batch
        tmp = []  # batch loss
        for batch_index, batch_data in enumerate(test_loader):
            encoder_inputs, labels = batch_data
            outputs = net(encoder_inputs, edge_index_data)
            loss = criterion(outputs, labels)
            tmp.append(loss.item())
            if batch_index % 100 == 0:
                print('test_loss batch %s / %s, loss: %.2f' % (
                    ↪ batch_index + 1, test_loader_length, loss.item()
                    ↪ ))
        test_loss = sum(tmp) / len(tmp)
        sw.add_scalar('test_loss', test_loss, epoch)
    print(test_loss)
```

4.2.6 Results

The model achieved a Mean Absolute Error (MAE) of 0.96, indicating a high level of accuracy in forecasting demand across different stations and times. This performance underscores the model's capability in capturing complex spatial-temporal relationships inherent in urban transportation data. The effectiveness of the ASTGCN model suggests that it can significantly aid in the operational planning and management of bike-sharing systems, potentially leading to optimized resource allocation and enhanced user satisfaction. Further research could explore the integration of additional dynamic factors such as weather conditions and urban events, which could further refine the accuracy of demand predictions.

4.3 Spatio-temporal Neural Structural Causal Models for Bike Flow Prediction

4.3.1 Summary

This paper introduces a novel Spatio-temporal Neural Structure Causal Model (STNSCM) designed to enhance bike-sharing system flow predictions. This model addresses shortcomings in existing prediction methods by incorporating a causal graph and applying the frontdoor criterion to reduce confounding biases in feature extraction. It also features a counterfactual representation reasoning module that predicts future scenarios based on current data, improving adaptability and accuracy under varying conditions. The study demonstrates that integrating causal reasoning with machine learning significantly boosts the model's performance against traditional methods, especially in handling external environment fluctuations.

4.3.2 Theoretical Background

1. **Causality and Structural Causal Models (SCM)**
   Causality in this context refers to the identification and use of causal relationships rather than mere correlations for prediction tasks. SCMs are used to model and understand causality in statistical data. SCMs are graphical models that describe the causal processes of the system through directed acyclic graphs (DAGs). Each node in a DAG represents a variable, and each edge represents a causal influence from one variable to another. This model allows the incorporation of domain knowledge about the causal structure of the problem, which is crucial for predicting outcomes under interventions or changed conditions.

2. **Causal Graphs and Frontdoor Criterion**
   The causal graph in STNSCM is used to map out the relationships between different factors affecting bike flow, such as spatial and temporal conditions, bike station status, and contextual variables like weather or urban dynamics. The frontdoor criterion is applied here as a method to adjust for confounding variables when direct causal effects from an exposure to

an outcome need to be measured. This is particularly useful in scenarios where backdoor adjustments (another method for controlling confounds) are not applicable due to unmeasured confounding.

3. **Counterfactual Representation Reasoning**
   One of the novel aspects of the proposed model is its use of counterfactual reasoning to enhance prediction accuracy. Counterfactual reasoning in this context involves imagining different scenarios ("what if" scenarios) based on modifications of the current factual data. This approach allows the model to predict how changes in conditions (like weather changes or special events) could affect bike flow, enabling more dynamic and responsive predictions.

4. **Frontdoor Adjustment Technique**
   This technique is a key element of the model's theoretical underpinning. It breaks down the causal effect of an exposure on an outcome into several stages that can be individually examined and adjusted. This is particularly effective in overcoming confounders that might affect both the exposure and the outcome, allowing the model to isolate and identify the direct effects of input variables on bike flow predictions.

5. **Graph Neural Networks (GNNs) for Spatio-Temporal Data**
   The model utilizes graph neural networks (GNNs) to handle the non-Euclidean data inherent in network-based systems like urban bike-sharing programs. GNNs are adept at managing data with complex relationships and dependencies, such as those found in transportation networks. The spatio-temporal aspect of the model addresses the dynamics of bike flow over time and across different urban areas, incorporating both spatial dependencies and temporal patterns.

4.3.3 Methodology of STNSCM with Mathematical Formulations

The framework of STNSCM various components, each specializing in handling different aspects of the data and model dynamics. These components include:
1. Input Gate: Processes and normalizes input data, integrating both historical data and contextual information (such as weather conditions or time of day). This component ensures that the input features are appropriately conditioned before they enter the main processing units of the model.
2. Dynamic Causality Generator: This component is crucial for embedding the temporal dynamics and causal relationships within the data into a dynamically evolving graph structure. It uses advanced neural network architectures to continually update its understanding of how different variables interact over time.
3. Spatio-temporal Evolutionary Graph Convolution Network (STEGCN): Applies graph convolution techniques to the data structured by the Dynamic Causality Generator. This network captures the spatial dependencies within the data, allowing for effective modeling of how conditions in one location influence or are correlated with conditions in other locations.
4. Counterfactual Representation Reasoning Module: Uses the processed and structured data to simulate and predict future states under hypothetical scenarios. This module enables the model to forecast outcomes based on potential changes in the input conditions, providing valuable insights into "what-if" analyses.
5. Output Layer: The final predictions are generated here, based on the data processed through the aforementioned components. This layer typically involves regression or classification layers, depending on the specific task (e.g., predicting traffic volume or determining the likelihood of traffic jams).

Detailed explanation of each component:

1. **Structural Causal Model (SCM)**
SCMs are mathematical models that use graphs to depict and analyze the causal relationships between variables. In these models, each node represents a variable, and each directed edge represents a direct causal effect from one variable to another. The SCM framework aims to not just capture associative relationships (as correlations do) but to model the generation of data based on underlying causal processes. Mathematically, an SCM consists of:

    - **Variables:** These are the nodes in the graph, which in the context of the paper include X_t (bike flow), C_t (contextual conditions), H_t (spatio-temporal states), and Y_{t+1} (predicted future states).
    - **Functional Relationships:** These are the edges. Each variable X in the graph is generated by a function f of its parent variables P_AX and possibly an external noise term U_X that captures random factors affecting X not explained by its parents.

    For example, in a simple SCM involving these variables, the functional relationships could be expressed as:

$$\begin{aligned} X_t &= f_X(C_t, U_{X_t}) \\ H_t &= f_H(X_t, C_t, U_{H_t}) \\ Y_{t+1} &= f_Y(H_t, U_{Y_{t+1}}) \end{aligned}$$

    where f_X, f_H, and f_Y are deterministic functions, and U_{X_t}, U_{H_t}, $U_{Y_{t+1}}$ are noise variables that are independent of each other.

2. **Causal Intervention via Frontdoor Criterion**
The Frontdoor Criterion is employed to estimate the causal effect of contextual conditions C_t on the spatio-temporal states H_t while controlling for the mediating effects of X_t (bike flow data) and S_t, T_t (spatio-temporal features). This method is particularly useful in scenarios where backdoor paths (which could introduce confounders) are present and cannot be adjusted directly. The criterion allows for isolating and analyzing the direct causal path from X_t through mediators to H_t, bypassing potential confounders. The mathematical formulation of the Frontdoor Criterion can be expressed as:

$$P(H_t \mid \mathrm{do}(X_t)) = \sum_{S_t,T_t} P(S_t, T_t \mid \mathrm{do}(X_t)) P(H_t \mid \mathrm{do}(S_t, T_t))$$

    which simplifies to:

$$= \sum_{S_t,T_t} P(S_t, T_t \mid X_t) \sum_{X'_t} P(H_t \mid S_t, T_t, X'_t) P(X'_t)$$

    Where:

    - $P(S_t, T_t \mid X_t)$ indicates the probability of observing particular spatio-temporal features given the bike flow data.
    - $P(H_t \mid S_t, T_t, X'_t)$ represents how these features along with modified input data X'_t influence the spatio-temporal state H_t.

- $P(X'_t)$ is the prior distribution of modified input data, reflecting historical or contextual influences that aren't directly observed.

Input Gate The Input Gate plays a pivotal role in processing and conditioning the input data for subsequent layers. It manipulates data to fit the prior distribution, ensuring that the transformed data (used as mediators in the Frontdoor Criterion) accurately reflects the causal relationships of interest.

Detailed Functionality:

Data Collection: It gathers historical bike flow data across different periods: weekly, daily, and by the hour. Concatenation and Transformation: The gate combines these different time slices with contextual conditions corresponding to the same periods. It processes this data through fully connected layers separately, then concatenates these processed features. Mathematical Representation of the Input Gate:

$$X'_t = X_{\text{in},t} + \tanh(X_{\text{in},t}\Theta_1 + a) \odot \sigma(X_{\text{in},t}\Theta_2 + b)$$

where:

- $X_{\text{in},t}$ represents the concatenated input features (both periodic flow data and external conditions).
- Θ_1, Θ_2 are weight matrices for transforming these inputs.
- a, b are bias vectors.
- tanh and σ (sigmoid function) are non-linear activation functions that modulate the impact of input features, ensuring that the transformations accommodate non-linear relationships in the data.
- The element-wise product $\odot$ combines these two transformed streams, merging the capacity to handle both linear and non-linear interactions within the data.

The combination of the Frontdoor Criterion and the sophisticated functionalities of the Input Gate ensures that the STNSCM effectively captures and utilizes the causal relationships in predicting bike flow. This approach mitigates potential biases from unmeasured confounders and enhances the model's ability to generalize across different environmental and contextual conditions by accurately reflecting the dynamics of bike usage influenced by both observed and latent variables.

3. **Dynamic Causality Generator**
   This component is responsible for processing the output from the Input Gate to create a representation that integrates spatio-temporal dynamics effectively. It uses a series of mathematical transformations to enhance the input data with causal understanding, which is essential for predicting future states based on historical and current inputs.

   - **Initialization and Input Combination**

     $$I_t = (X'_t \parallel H_{t-1}) \cdot \theta_{dym} + b_{dym}$$

     - X'_t is the context-conditioned feature output from the Input Gate.
     - H_{t-1} is the spatio-temporal state from the previous time step, providing historical continuity.

- θ_{dym} and b_{dym} are trainable parameters (weights and biases) of the dynamic causality generator.
- $\|$ denotes concatenation, combining the current input features with historical spatio-temporal states to form a comprehensive input vector.
- I_t is the intermediate tensor that will be further processed to capture dynamic causality.

- **Squeeze Operation**

$$z_s = \frac{1}{d} \sum_{c=1}^{d} I_t[:, c]$$

  - This operation calculates the global average pooling over the channel dimension c of I_t, compressing the feature maps by taking the average of each feature channel across all data points.
  - This results in a vector z_s that represents a squeezed, summarized view of the input features, reducing dimensionality and focusing on the most salient features.

- **Excitation Operation**

$$z_e = \sigma(\theta_{e2} \cdot \mathrm{ReLU}(\theta_{e1} \cdot z_s))$$

  - θ_{e1} and θ_{e2} are weights of a two-layer neural network that processes the squeezed features.
  - ReLU (Rectified Linear Unit) introduces non-linearity, enhancing the model's ability to capture complex patterns.
  - σ (sigmoid function) normalizes the output, scaling the excitation values between 0 and 1 , effectively determining the importance of each feature.

- **Feature Rescaling**

$$DX_t = I_t \odot z_e$$

  - DX_t represents the dynamically scaled input features.
  - The element-wise product $\odot$ with the excitation output z_e allows the model to emphasize or de-emphasize certain features based on their learned importance, thus dynamically modifying the input tensor based on the content of the data itself.

- **Causality Embedding via Self-Attention**

$$A_{dyn,t} = \mathrm{ReLU}\left(\tanh\left(\frac{DX_t DX_t^T}{\sqrt{d}}\right)\right)$$

  - This equation constructs a self-attention mechanism where $DX_t DX_t^T$ computes the similarity matrix between different features, which is scaled by $\sqrt{d}$ to stabilize gradients during training.
  - tanh provides a non-linear transformation to scale the values, and ReLU ensures that only positive influences are considered, effectively embedding dynamic causality into the causal graph.
  - $A_{dyn,t}$ represents the dynamically updated adjacency matrix, indicating how different features should influence each other in the subsequent graph convolution operations, reflecting the changing relationships over time.

The Dynamic Causality Generator is central to the model's ability to understand and predict changes over time by learning from the evolving patterns in the data. Through a series of transformations, it adjusts the representation of data to encapsulate both the temporal dynamics and the causal interactions. This allows the STNSCM to not only react to changes in input data but to anticipate future states based on the learned causal structure, greatly enhancing the predictive power and accuracy of the model in dynamic environments such as urban traffic and transportation systems.

4. **Spatio-temporal Evolutionary Graph Convolution**
STEGCN is crafted to capture complex dependencies in spatio-temporal data by integrating information across both spatial and temporal dimensions. It efficiently models how conditions in one location influence another over time, which is critical for forecasting systems with inherent spatial interactions (like traffic flows).

   - **Static Graphs Construction**

$$A_{trans,k\ell} = \begin{cases} \exp\left(-\frac{d_{k\ell}^2}{\sigma^2}\right) & \text{if } d_{k\ell} > \epsilon_{geo} \\ 0 & \text{otherwise} \end{cases}$$

     – A_{geo} represents the geographical distance graph, where $d_{k\ell}$ is the physical distance between regions k and ℓ, σ is a scaling parameter, and ϵ_{geo} is a threshold distance beyond which no edge is assumed.
     – A_{trans} is the transition probability graph based on the historical traffic flow $T_{k\ell}$ from region k to ℓ, normalized by the total outflow from region k.

   - **Graph Convolution Layers**

$$X_t^{(n)} = \alpha_0 X_t^{(n-1)} + \alpha_1 A_{geo} X_t^{(n-1)} + \alpha_2 A_{trans} X_t^{(n-1)} + \alpha_3 A_{dyn,t} X_t^{(n-1)}$$

     – $X_t^{(n)}$ is the feature representation at layer n and time t.
     – A_{geo}, A_{trans}, and $A_{dyn,t}$ are normalized adjacency matrices corresponding to geographical, transition, and dynamically updated causal relationships, respectively.
     – $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ are coefficients that weight the contributions of each component to the node features in the next layer.

   - **Propagation and Feature Transformation**

$$X_{out}^t = \text{ReLU}\left(\sum_{k=0}^{n} X_t^{(k)} W^{(k)} + b^{(k)}\right)$$

     – This represents the aggregation and transformation process where features from different graph convolution layers $X_t^{(k)}$ are combined using learned weights $W^{(k)}$ and biases $b^{(k)}$.
     – The ReLU activation function ensures non-linearity in feature transformation, allowing the model to capture more complex patterns.

   The Spatio-temporal Evolutionary Graph Convolution effectively synthesizes data across time and space using graph-based techniques. It allows the model to consider how conditions in different areas are interconnected and how these connections evolve over time, which is crucial for predicting dynamics in systems where past interactions inform future states.

5. **Spatio-temporal Neural Structural Causal Unit (STNSCU)**
STNSCU is engineered to capture and model the intricate causal relationships within the data by using a series of graph convolutions and gating mechanisms. It utilizes the structural causal modeling approach to integrate and reason about the information encoded in the inputs, thereby enabling the unit to predict future states effectively. The STNSCU typically involves multiple steps, including gating mechanisms, graph convolutional operations, and a combination of these features to predict the next state.
The STNSCU Formulation involves several gating mechanisms and graph convolution operations. Here we break down the equation and its components:

   - **Gating Equations:**
     - Reset gate:
       $$r_t = \sigma(\Theta_r * G(X_t \parallel H_{t-1}) + b_r)$$
     - Update gate:
       $$z_t = \sigma(\Theta_z * G(X_t \parallel H_{t-1}) + b_z)$$
     - Candidate state:
       $$\hat{h}_t = \phi(\Theta_h * G(X_t \parallel (r_t \odot H_{t-1})) + b_h)$$
   - **State Update Equation:**
     $$H_t = z_t \odot H_{t-1} + (1 - z_t) \odot \hat{h}_t$$
   - **Component Descriptions:**
     - X_t' is the input at time t, processed by previous layers such as an input gate or a dynamic causality generator.
     - H_{t-1} is the spatio-temporal state from the previous timestep, providing historical context.
     - $\Theta_r, \Theta_z, \Theta_h$ are parameter matrices for the reset gate, update gate, and transformation gate, respectively.
     - b_r, b_z, b_h are bias terms corresponding to each gate.
     - G denotes a graph convolution operation that processes the input data according to the learned graph structure.
     - σ is the sigmoid activation function, used here to normalize the outputs of the reset and update gates.
     - ϕ is typically a non-linear activation function like the hyperbolic tangent (tanh), used to introduce non-linearity in the transformation of inputs.
     - r_t (reset gate) controls how much of the past state is forgotten.
     - z_t (update gate) determines how much of the new state $\hat{h}_t$ is used to update the state.
     - $\hat{h}_t$ is the candidate state, proposed as the new state.
     - $\odot$ represents element-wise multiplication, crucial for the gating processes.

   Here are detailed explanations of the operations utilized in graph convolutional networks:

   - **Graph Convolution (G):**

    - The graph convolution operation G processes the inputs (both the current and the past states) in the context of the graph's topology. This operation allows the unit to account for the spatial relationships encoded in the data, ensuring that spatial dependencies are considered when updating states.

- **Gating Mechanisms:**
    - **Reset Gate (r_t):** This gate determines how much of the previous state to retain. By applying a sigmoid function, it ensures that the values are between 0 and 1, where closer to 1 means more of the past state is kept.
    - **Update Gate (z_t):** This gate decides how much of the candidate state $\hat{h}_t$ is used to update the new state H_t. It acts similarly to the forget gate in LSTM units, controlling the flow of information across timesteps.
    - **Candidate State ($\hat{h}_t$):** This is a new state proposed by the model, calculated by transforming the input and applying a non-linear activation. It is modulated by the reset gate, which allows the model to forget irrelevant past information selectively.
- **State Update:**
    - The final state H_t is computed as a weighted sum of the previous state and the candidate state, where the weights are given by the update gate. This step allows the unit to smoothly integrate new information while retaining important information from the past.

6. **Counterfactual Representation Reasoning**
   This module leverages the concept of counterfactual thinking—considering "what might have been" if different decisions had been made or different conditions had occurred. It essentially helps the model to extrapolate beyond the observed data to forecast what could happen under different sets of assumptions or inputs.

- **Counterfactual Scenario Forecasting**

$$P(Y_{C_{t+1}} \mid X_t) = \sum_{H'_t} P(Y_{t+1} \mid \mathrm{do}(H'_t), \mathrm{do}(C_{t+1}), X_t) P(H'_t \mid X_t, \mathrm{do}(C_{t+1}))$$

    - $Y_{C_{t+1}}$ represents the outcome variable (e.g., future bike flow) conditioned on a different set of contextual conditions C_{t+1}.
    - X_t is the current observed state or input data.
    - H'_t denotes the hypothetical or counterfactual spatio-temporal states under the new conditions C_{t+1}.
    - The first probability term captures the likelihood of observing Y_{t+1} given the intervention on both H'_t and C_{t+1}, controlled by the current state X_t.
    - The second probability term is the likelihood of the hypothetical state H'_t given the current state and an intervention on C_{t+1}.
    - This equation allows the model to compute the likelihood of various future outcomes by considering how changes in external conditions might influence the future states and outcomes.
- **Attention Mechanism for Counterfactual Reasoning**

$$Q = C_{pred} W_Q, \quad K = C_{his} W_K, \quad V = H_{his} W_V$$
$$H_{pred} = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}} V\right)$$

- C_{pred} represents the predicted external conditions for the future.
- C_{his} are the historical external conditions.
- H_{his} denotes historical spatio-temporal states.
- The attention mechanism, through the operations Q, K, and V, computes the relevance of different historical contexts to predict future states effectively using learned weights.
- W_Q, W_K, and W_V are learnable weights for the query, key, and value matrices in the attention mechanism, respectively. These weights are crucial for transforming input data into the appropriate formats for subsequent attention calculations.
- Q, K, and V are the query, key, and value components of the attention mechanism. They play distinct roles:
  * Q (Query): Represents the element or set of elements for which the attention mechanism tries to find relevant information across other elements.
  * K (Key): Corresponds to elements that are compared against the query to compute attention scores.
  * V (Value): Holds the actual information that will be aggregated based on the attention scores derived from the query-key comparisons.
- H_{pred} is the predicted spatio-temporal state vector, computed using a scaled dot-product attention mechanism:

  $$H_{\text{pred}} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

  where:
  * The softmax function applied over QK^T scaled by $\sqrt{d}$ normalizes the attention weights. The scaling factor d is typically the dimensionality of Q and K, which helps in stabilizing the gradients during training.
  * V provides the values to be aggregated according to these normalized attention weights, effectively determining the output of the attention mechanism based on the input data's contextual relationships.

4.3.4 Implementation Details

As explained above, the models are implemented to the Jersey City Citibike dataset similar to the previous 2 models explained. But, have also used weather data features. The code is available here.

4.3.5 Results

The MAE was 2.67, showing good results compared to others when contextual features are considered.

5 Conclusion

In this study, I have implemented and evaluated three distinct models to predict Citibike demand, each utilizing advanced graph convolutional techniques to model spatio-temporal data complexities. The models tested included the Spatio-Temporal Graph Convolutional Networks (STGCN), Attention Based Spatial-Temporal Graph Convolutional Networks (ASTGCN), and a third model specializing in capturing dynamic spatial-temporal correlations within traffic data.
My findings indicate varied performance across the models, as evidenced by the Mean Absolute Error (MAE) metric. The STGCN model, designed to capture comprehensive spatio-temporal correlations through its convolutional framework, demonstrated substantial efficacy in understanding traffic dynamics, suggesting its robustness in handling spatial and temporal data simultaneously. The ASTGCN model, which integrates attention mechanisms, provided enhanced capability in prioritizing crucial temporal and spatial features, potentially offering improved accuracy in scenarios where traffic patterns exhibit high non-linear characteristics.
The comparative analysis of MAEs underscores the importance of model selection based on specific traffic forecasting scenarios. Each model's ability to minimize error and capture intricate patterns in data presents valuable insights into the practical applications of graph convolutional networks in real-world traffic management systems. These findings could aid in the development of more accurate and reliable traffic prediction tools, contributing to better traffic management and planning.
Future research could explore the integration of these models with real-time data and the inclusion of additional variables such as weather conditions and special events, which significantly influence traffic flows. Moreover, further tuning and experimentation with hybrid models combining the strengths of convolutional and attention mechanisms may yield even more precise predictions, thereby enhancing the operational efficiency of traffic management systems.

6 References

- Link to the papers I have referred to: Literature Survey
- Github link to all the codes: Citibike Demand Prediction using GCN's