GENERATIVE AI FOR SUSTAINABLE CITIES

July 16, 2024

Abstract

The rapid urbanization and population growth in cities have led to significant challenges in traffic management, contributing to congestion and environmental concerns. This paper presents a novel approach to traffic prediction and generation using Generative AI, combining Graph Convolutional Networks (GCNs), Gated Recurrent Units (GRUs), and Generative Adversarial Networks (GANs). By leveraging historical traffic data, our model captures both spatial and temporal dependencies, producing accurate and realistic traffic predictions. This approach aims to optimize traffic flow, reduce congestion, and promote sustainable urban development, thereby enhancing the quality of urban life.

1 Introduction

Urbanization and population growth are accelerating globally, leading to increased demand for effective traffic management solutions in cities. Traditional methods often struggle to cope with the dynamic and complex nature of urban traffic systems, resulting in congestion, longer travel times, and higher environmental pollution. To address these challenges, we propose an innovative approach that utilizes Generative Artificial Intelligence (Generative AI) to predict and generate traffic patterns based on historical data.

Our approach integrates several advanced deep learning techniques: Graph Convolutional Networks (GCNs) to capture the spatial dependencies in the traffic network, Gated Recurrent Units (GRUs) to model the temporal dynamics, and Generative Adversarial Networks (GANs) to generate realistic traffic scenarios. The GCN processes the spatial structure of the traffic network, capturing dependencies between different locations. The output of the GCN serves as input to the GRU, which learns the temporal patterns in the traffic data. Finally, the GRU's output is fed into the GAN, which generates future traffic states.

The GCN is further enhanced with Graph Attention Network (GAT) convolutions to incorporate edge features, enabling the model to capture more complex spatial interactions. The GAN architec-

ture employs a Variational Autoencoder (VAE)-based generator and a discriminator to ensure that the generated traffic scenarios closely resemble real data. The generator's encoder compresses the input graph data into a latent representation, while the decoder reconstructs the graph data from this latent space. The discriminator processes both the adjacency matrix and the feature vector generated by the generator to validate their realism.

Through extensive experiments on real-world traffic datasets, our model demonstrates its ability to accurately predict and generate traffic patterns. By optimizing traffic flow and reducing congestion, our approach contributes to the broader goal of creating sustainable cities with improved quality of life for their inhabitants. This paper outlines the theoretical foundations of our model, its architecture, and the results of our experimental validation, highlighting its potential to transform urban traffic management.

2 GCN (Graph Convolutional Network)

In this project, the GCN (Graph Convolutional Network) serves as the initial component responsible for capturing the spatial dependencies in the traffic network. The GCN processes the raw traffic data, encoding the relationships between different locations in the network. Additionally, we use GAT (Graph Attention Network) convolutions to incorporate edge features, enhancing the model's ability to capture more complex spatial interactions.

2.1 Input Data

The input to the GCN consists of traffic data represented as a graph, where each node corresponds to a location in the traffic network, and edges represent the connections or interactions between these locations. The input can be denoted as $X \in \mathbb{R}^{N \times F}$, where N is the number of nodes and F is the number of features per node.

2.2 GCN Architecture

The GCN architecture comprises multiple layers of graph convolutions, which aggregate information from neighboring nodes to learn spatial features. The basic operation of a single GCN layer can be described as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{1}$$

where:

- $\mathbf{H}^{(l)}$ is the input feature matrix at layer l ($\mathbf{H}^{(0)} = \mathbf{X}$),
- $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the normalized adjacency matrix with added self-loops,
- $\mathbf{A}$ is the adjacency matrix of the graph,
- $\mathbf{D}$ is the degree matrix of $\mathbf{A}$,
- $\mathbf{W}^{(l)}$ is the weight matrix at layer l,
- σ is an activation function, such as ReLU.

2.3 GAT Convolutions

To further enhance the GCN, we incorporate Graph Attention Network (GAT) convolutions. GAT convolutions introduce an attention mechanism that allows the model to weigh the importance of neighboring nodes differently, based on their features and connectivity. The attention coefficients α_{ij} are computed as follows:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j\right]\right) \tag{2}$$

where:

- $\mathbf{h}_i$ and $\mathbf{h}_j$ are the feature vectors of nodes i and j,
- $\mathbf{W}$ is a shared weight matrix,
- $\mathbf{a}$ is the weight vector for computing attention coefficients,
- $\parallel$ denotes concatenation.

The attention coefficients α_{ij} are obtained by normalizing e_{ij} using the softmax function:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{3}$$

The output features of each node are then aggregated as a weighted sum of its neighbors' features:

$$\mathbf{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right) \tag{4}$$

2.4 Integration with GRU and GAN

2.4.1 From GCN to GRU

- The GCN processes the spatial structure of the traffic network, capturing dependencies between different locations.
- The output of the GCN, denoted as $\mathbf{E}_{\text{GCN}} \in \mathbb{R}^{T \times N \times F_{\text{in}}}$, serves as the input to the GRU. This embedding contains spatial features over a series of time steps.

2.4.2 GCN Processing

- The GCN takes raw traffic data and processes it to produce embeddings that capture spatial dependencies in the traffic network.
- This embedding $\mathbf{E}_{\text{GCN}}$ encapsulates the spatial features and is fed into the GRU to learn the temporal dynamics of the traffic patterns.

3 GRU (Gated Recurrent Unit)

In this project, the GRU (Gated Recurrent Unit) serves as the intermediate component that processes the embeddings generated by the Graph Convolutional Network (GCN) and prepares them for the Generative Adversarial Network (GAN). This module leverages the temporal dynamics of traffic data to produce meaningful embeddings that encapsulate both spatial and temporal information, enhancing the predictive capability of the entire system.

3.1 Input and Output Embeddings

3.1.1 Input Embedding

The input to the GRU is an embedding generated by the GCN. This embedding encapsulates the spatial dependencies in the traffic network and has a specific dimensionality, denoted as $E_{\text{GCN}} \in \mathbb{R}^{T \times N \times F_{\text{in}}}$, where T is the number of time steps, N is the number of nodes (or locations) in the graph, and F_{in} is the feature dimension of the GCN output.

3.1.2 Output Embedding

The output from the GRU is a transformed embedding that will be fed into the GAN. This embedding captures the temporal evolution of the traffic patterns and is of size $E_{\text{GRU}} \in \mathbb{R}^{T \times N \times F_{\text{out}}}$, where F_{out} is the feature dimension required by the GAN.

3.2 GRU Architecture

The GRU is a type of recurrent neural network (RNN) that is designed to handle sequential data and capture long-term dependencies efficiently. It consists of a series of gates that regulate the flow of information, allowing the model to retain and forget information as needed.

3.2.1 GRU Cell Structure

- **Update Gate**: Decides how much of the past information needs to be passed to the future.
- **Reset Gate**: Decides how much of the past information to forget.
- **Current Memory Content**: Computes the new memory content.
- **Final Memory at Current Time Step**: Combines the old and new memory content based on the update gate's decision.

Mathematically, the operations in a GRU cell at time step t can be described as follows:

$$\text{Update gate}: \quad z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{5}$$

$$\text{Reset gate}: \quad r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{6}$$

$$\text{New memory content}: \quad \tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{7}$$

$$\text{Final memory}: \quad h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{8}$$

where:

- σ is the sigmoid activation function,
- $\tanh$ is the hyperbolic tangent activation function,
- $*$ denotes element-wise multiplication,
- W_z, W_r, W are the weight matrices for the respective gates.

3.3 Integration with GCN and GAN

3.3.1 From GCN to GRU

- The GCN processes the spatial structure of the traffic network, capturing dependencies between different locations.
- The output of the GCN, E_{GCN}, serves as the input to the GRU. This embedding contains spatial features over a series of time steps.

3.3.2 GRU Processing

- The GRU takes E_{GCN} and processes it sequentially over time steps T.
- It learns the temporal patterns and dependencies in the traffic data, producing an output embedding E_{GRU} that integrates both spatial and temporal information.

3.3.3 To GAN

- The GRU's output embedding, E_{GRU}, is then used as input to the GAN.
- The GAN utilizes this enriched embedding to generate future traffic states, leveraging both the spatial relationships captured by the GCN and the temporal dynamics modeled by the GRU.

4 GAN Architecture

The proposed VAE-based generator consists of an encoder and a decoder, both incorporating convolutional layers to process graph data and fully connected layers to manage feature vectors.

4.1 Encoder

The encoder compresses the input graph data into a latent representation. It comprises three convolutional layers (conv1, conv2, and conv3), followed by a fully connected layer (fc_enc) that outputs the mean (μ) and log variance ($\log \sigma^2$) of the latent space. This architecture is designed to capture intricate patterns in the graph adjacency matrix and the associated feature vectors.

4.2 Decoder

The decoder reconstructs the graph data from the latent representation. It includes a fully connected layer (fc_dec_adj) to reshape the latent vector into a higher-dimensional space, followed by three transposed convolutional layers (deconv1, deconv2, and deconv3) that generate the reconstructed adjacency matrix. Additionally, a separate fully connected layer (fc_dec_vec) generates the associated feature vector.

4.3 Reparameterization Trick

To enable backpropagation through stochastic latent variables, we employ the reparameterization trick. This involves sampling $\epsilon \sim \mathcal{N}(0, I)$ and computing the latent variable as follows:

$$z = \mu + \sigma \cdot \epsilon,$$

where

$$\sigma = \exp\left(0.5 \cdot \log \sigma^2\right).$$

4.4 Forward Pass

The input to the generator is a 1-D vector of shape (batch size, number of nodes) provided by the GRU, and the current adjacency matrix of shape (batch size, edge features, number of nodes, number of nodes).

- The 1-D vector input first passes through a fully connected layer and is then reshaped to (batch size, 1, number of nodes, number of nodes). It is subsequently processed by a convolutional layer to produce an output vector that can be added to the current adjacency matrix.
- After the addition of the output vector and the adjacency matrix, the resultant matrix passes through the encoder layer, which outputs μ and σ. These are reparameterized to provide the adjacency matrix and 1-D vector for the next time step.
- Shape of the adjacency matrix: (batch size, edge features, number of nodes, number of nodes)
- Shape of the 1-D vector: (batch size, number of nodes)

4.5 Discriminator

The discriminator's primary goal is to ensure that the outputs generated by the generator closely resemble real data. It separately processes both the adjacency matrix and the 1-D vector generated by the generator for the next time step.

- The 1-D vector is passed through a series of linear layers.
- The adjacency matrix is processed by a convolutional layer, followed by a block consisting of a convolutional layer, Leaky ReLU activation, and batch normalization. It is then passed through a classifier layer that includes pooling, flattening, and linear layers.

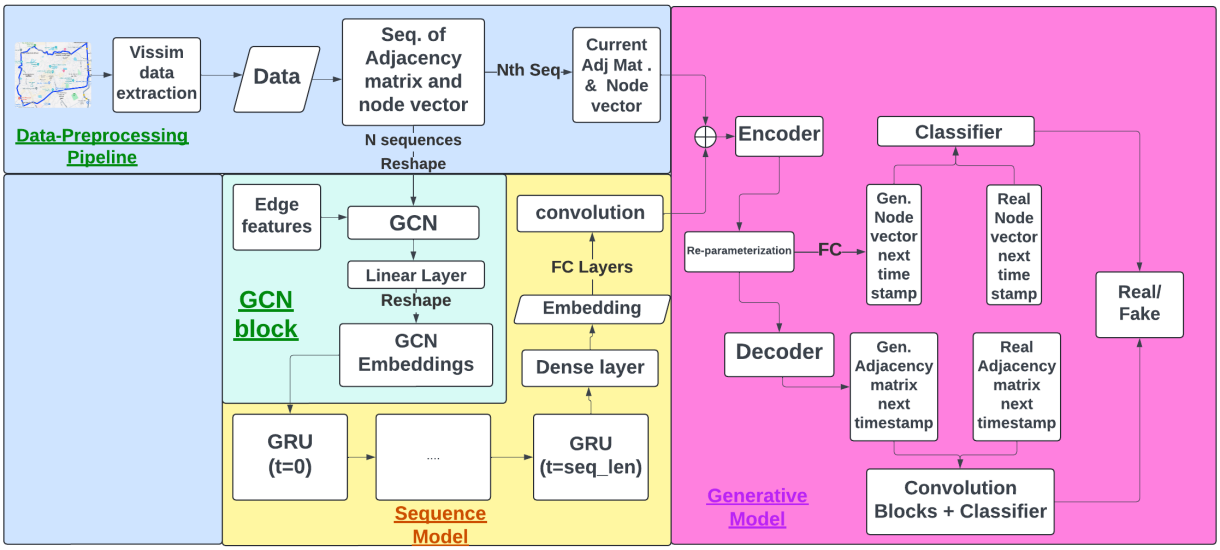

Figure 1: Overview

5 Loss Function

Combining multiple loss functions into a single loss function for a neural network is a common practice when you need to optimize for several different objectives simultaneously. Given the various loss components, let's define the combined loss function. Each component will be weighted appropriately to ensure that they contribute to the final loss according to their importance.

5.1 Loss Components

- **MSE Loss (Turning Ratio)**: Measures the mean squared error of the predicted turning ratios.
- **Categorical Loss (Road Blocking)**: Measures the cross-entropy loss for categorical data related to road blocking.
- **KLD Loss (Parameterization)**: Measures the Kullback-Leibler divergence for parametrization.
- **MSE Loss (Node Vector)**: Measures the mean squared error for node vectors.
- **TRPC Loss (Turning Ratio Permission Compatibility)**: Ensures that if a categorical value is 0, the turning ratio should be zero.
- **TRSC Loss (Turning Ratio Sum Constraint)**: Ensures that the sum of incoming and outgoing turning ratios is one.
- **Simulation Loss (Temporary Vehicle Passing Constraint)**: Ensures that all vehicles pass through the network as expected.

5.2 Combined Loss Function

We can combine these losses as follows, using weighting factors $\lambda_1, \lambda_2, \ldots, \lambda_7$ to balance their contributions:

$$\begin{aligned}\text{Total Loss} = {} & \lambda_1 \cdot \text{MSE\_Loss}_{\text{turning ratio}} + \lambda_2 \cdot \text{Categorical\_Loss}_{\text{road blocking}} \\ & + \lambda_3 \cdot \text{KLD\_Loss} + \lambda_4 \cdot \text{MSE\_Loss}_{\text{node vector}} \\ & + \lambda_5 \cdot \text{TRPC\_Loss} + \lambda_6 \cdot \text{TRSC\_Loss} \\ & + \lambda_7 \cdot \text{Simulation\_Loss}\end{aligned}$$