Advancing Urban Traffic Prediction: Spatio-temporal Data Analysis based on Deep Learning Architectures

Junyan Yu Hongliang Lu Xiaotao Zhang

I. Topic Selection Background and Reasons

A. Background

Urban traffic congestion is an increasing problem worldwide due to rapid urbanization. This not only wastes time but also increases fuel consumption and pollution. With the rise of smart cities, intelligent transportation systems have become crucial, with traffic prediction playing a vital role in supporting traffic management and planning.

B. Significance of the Topic

Accurate traffic predictions enhance traffic management, reduce congestion, and improve road capacity. They also optimize traffic signal timing, reducing waiting times, and provide real-time traffic information to the public, improving travel choices and experience. Additionally, these predictions help minimize fuel consumption and pollution, contributing to environmental sustainability.

C. Specific Examples and Application Scenarios of Traffic Prediction

Real-time traffic prediction enables more efficient traffic light control, reducing peak hour congestion. Navigation systems use these predictions to provide optimal routing, saving drivers time by avoiding congested areas. Public transit systems improve punctuality and service by adjusting schedules based on predictions. Emergency systems use traffic predictions to manage the impact of sudden events, ensuring smoother traffic flow. These applications demonstrate the utility of traffic prediction in enhancing urban mobility and safety.

D. Data Handling

The input data format is a traffic flow data table with time as row labels and probes as column labels, and a relationship matrix of size (number of probes * number of probes) between the probes. In the preliminary analysis, we noticed that the initial data had a wide range and significant order of magnitude differences between data points, requiring the use of normalization methods during processing.

II. Models and Methods

A. LSTM/GRU

Firstly, our group considered using recurrent neural networks to process temporal information, and therefore, we adopted the LSTM/GRU algorithm.LSTM can remember numerical values of indefinite time length by introducing three gate functions: input gate, forget gate, and output gate to control the input value, memory value, and output value. GRU is a gate mechanism unit in RNN, similar to LSTM, but with only two gates: update gate and reset gate. The specific structure is shown in the following figure:

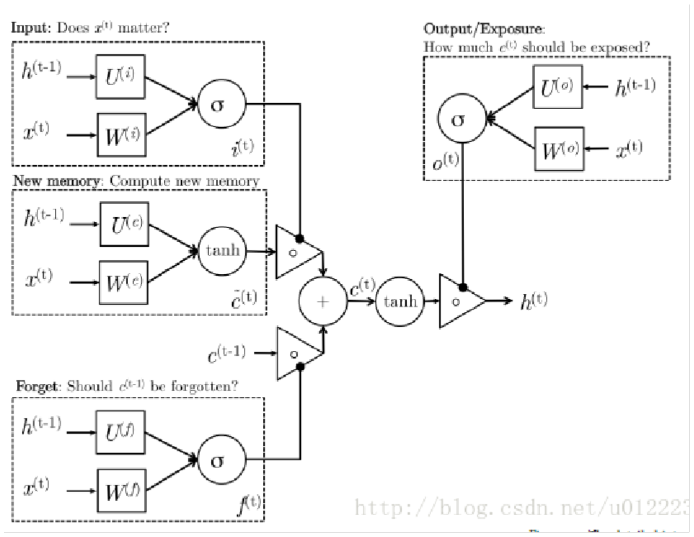
Input: Does $x^{(t)}$ matter?
$h^{(t-1)}$
$U^{(i)}$
$x^{(t)}$
$W^{(i)}$
σ
$i^{(t)}$
New memory: Compute new memory
$U^{(c)}$
$W^{(c)}$
tanh
$\tilde{c}^{(t)}$
$c^{(t-1)}$
Forget: Should $c^{(t-1)}$ be forgotten?
$U^{(f)}$
$W^{(f)}$
$f^{(t)}$
Output/Exposure: How much $c^{(t)}$ should be exposed?
$U^{(o)}$
$W^{(o)}$
$o^{(t)}$
$c^{(t)}$
$h^{(t)}$
http://blog.csdn.net/u012223

Fig. 1. LSTM Architecture

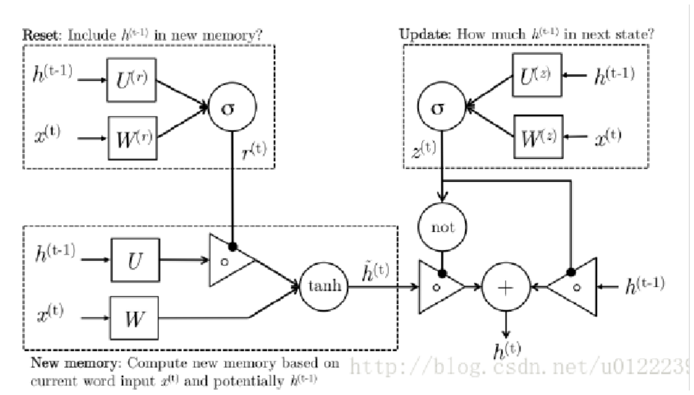
Reset: Include $h^{(t-1)}$ in new memory?
$h^{(t-1)}$
$U^{(r)}$
$x^{(t)}$
$W^{(r)}$
σ
$r^{(t)}$
Update: How much $h^{(t-1)}$ in next state?
$U^{(z)}$
$W^{(z)}$
$z^{(t)}$
not
U
W
tanh
$\tilde{h}^{(t)}$
$h^{(t)}$
New memory: Compute new memory based on current word input $x^{(t)}$ and potentially $h^{(t-1)}$
http://blog.csdn.net/u0122239

Fig. 2. GRU Architecture

Due to the high computational efficiency and faster convergence of GRU, we adopt the form of directly calling GRU packets to complete the code.

But during the experiment, We found that no matter how We adjust the learning rate as well as optimizing training methods, the validation loss is consistently high. So We're displaying the results to see if it's a problem with the model itself. Then We discover the key of the problem:In the case where the input training batches are completely different, the corresponding values between the output batches given by GRU are completely the same, which means that the model fails to learn the features of the time series.

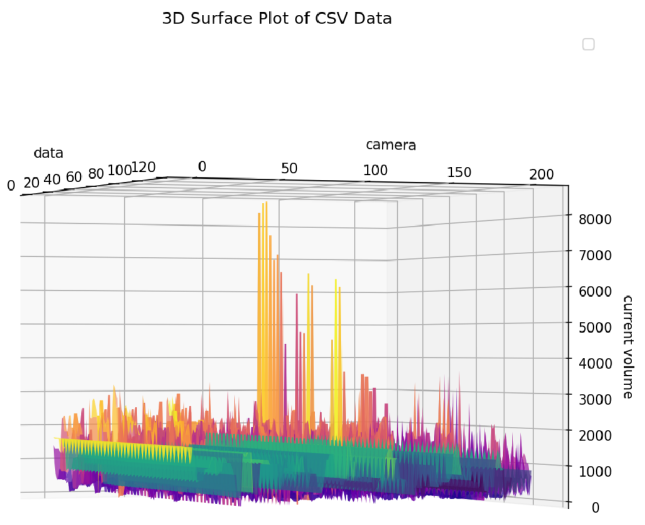

Fig. 3. 3D plan view of the model's output and truth value

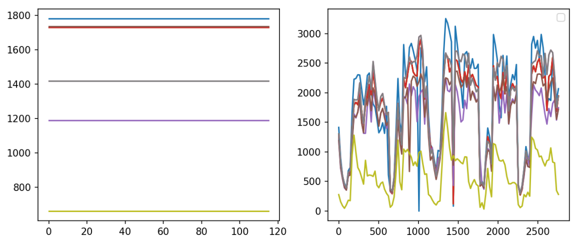

Fig. 4. 2D line chart of the model's output and truth value

In the 3D graph, the red values represent the values of the training set, while the blue values represent the output results of the model. It can be seen that the results are quite distinct in the dimension of the probe, but the values are all the same in the dimension of time. This indicates that the model has failed to learn the temporal features of the input sequence. In the same two-dimensional graph, we can also see similar results. For any specific probe, its corresponding predicted value at each time point is the same.

Then We analyzed the reason with the teaching assistant and found that when setting the number of layers to 2, one of the memory layers passed through died shortly after running. Eventually, we suspected that the problem was due to the forgetting mechanism, which caused the model to not learn the corresponding temporal features.

At Epoch 1, differences can still be observed between batches of output.

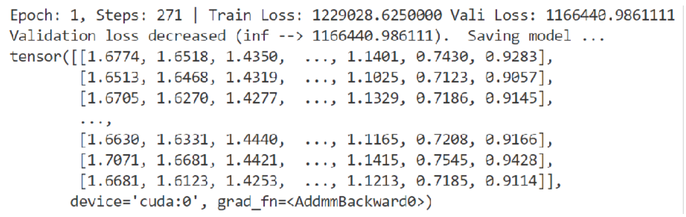
```
Epoch: 1, Steps: 271 | Train Loss: 1229028.6250000 Vali Loss: 1166440.9861111
Validation loss decreased (inf --> 1166440.986111).  Saving model ...
tensor([[1.6774, 1.6518, 1.4350,  ..., 1.1401, 0.7430, 0.9283],
        [1.6513, 1.6468, 1.4319,  ..., 1.1025, 0.7123, 0.9057],
        [1.6705, 1.6270, 1.4277,  ..., 1.1329, 0.7186, 0.9145],
        ...,
        [1.6630, 1.6331, 1.4440,  ..., 1.1165, 0.7208, 0.9166],
        [1.7071, 1.6681, 1.4421,  ..., 1.1415, 0.7545, 0.9428],
        [1.6681, 1.6123, 1.4253,  ..., 1.1213, 0.7185, 0.9114]],
       device='cuda:0', grad_fn=<AddmmBackward0>)
```

Fig. 5. Output at epoch 1

After ten epochs, the output batch is completely identical.

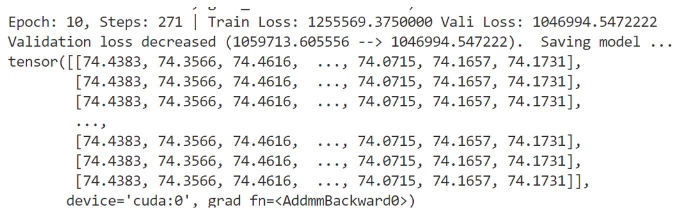
```
Epoch: 10, Steps: 271 | Train Loss: 1255569.3750000 Vali Loss: 1046994.5472222
Validation loss decreased (1059713.605556 --> 1046994.547222).  Saving model ...
tensor([[74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731],
        [74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731],
        [74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731],
        ...,
        [74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731],
        [74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731],
        [74.4383, 74.3566, 74.4616,  ..., 74.0715, 74.1657, 74.1731]],
       device='cuda:0', grad fn=<AddmmBackward0>)
```

Fig. 6. Output at epoch 10

To further investigate this issue, We attempted to output hidden weights.

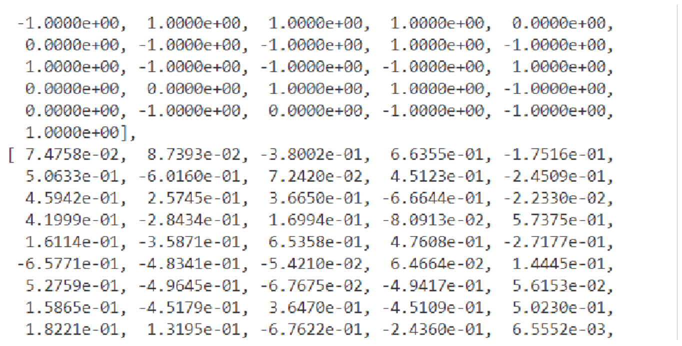
```
 -1.0000e+00,  1.0000e+00,  1.0000e+00,  1.0000e+00,  0.0000e+00,
  0.0000e+00, -1.0000e+00, -1.0000e+00,  1.0000e+00, -1.0000e+00,
  1.0000e+00, -1.0000e+00, -1.0000e+00, -1.0000e+00,  1.0000e+00,
  0.0000e+00,  0.0000e+00,  1.0000e+00,  1.0000e+00, -1.0000e+00,
  0.0000e+00, -1.0000e+00,  0.0000e+00, -1.0000e+00, -1.0000e+00,
  1.0000e+00],
[ 7.4758e-02,  8.7393e-02, -3.8002e-01,  6.6355e-01, -1.7516e-01,
  5.0633e-01, -6.0160e-01,  7.2420e-02,  4.5123e-01, -2.4509e-01,
  4.5942e-01,  2.5745e-01,  3.6650e-01, -6.6644e-01, -2.2330e-02,
  4.1999e-01, -2.8434e-01,  1.6994e-01, -8.0913e-02,  5.7375e-01,
  1.6114e-01, -3.5871e-01,  6.5358e-01,  4.7608e-01, -2.7177e-01,
 -6.5771e-01, -4.8341e-01, -5.4210e-02,  6.4664e-02,  1.4445e-01,
  5.2759e-01, -4.9645e-01, -6.7675e-02, -4.9417e-01,  5.6153e-02,
  1.5865e-01, -4.5179e-01,  3.6470e-01, -4.5109e-01,  5.0230e-01,
  1.8221e-01,  1.3195e-01, -6.7622e-01, -2.4360e-01,  6.5552e-03,
```

Fig. 7. hidden weights during training

It can be observed that after running the program for a period of time, one layer of hidden weights has completely changed to 1, 0, -1, indicating that all the information in this layer has died. This suggests that the GRU model may have poor predictive ability for this dataset, and after multiple rounds of learning, it has forgotten all the previously input information.

It can be seen that the GRU model alone has poor prediction performance for this dataset. In the future, it can be considered to combine Transformer and other models for experiments. As it turned out, GRU generally performs better in music, speech signal processing, and NLP.

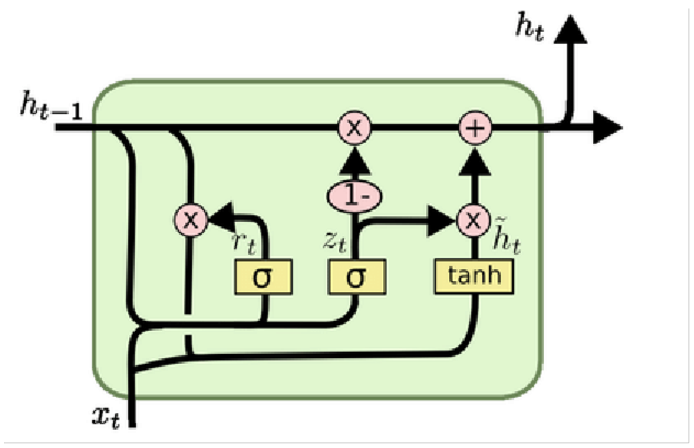

Fig. 8. Working Diagram of GRU

B. Encoder-based Transformer

The example code implements an Encoder Only Transformer, where SelfAttention extracts dependencies between probes, and the FeedForward neural network extracts temporal dependencies.

This model can learn features from hundreds of days of traffic flow data to perform traffic prediction, with the final validation loss on the validation set around 48000, and the predictions are relatively accurate.

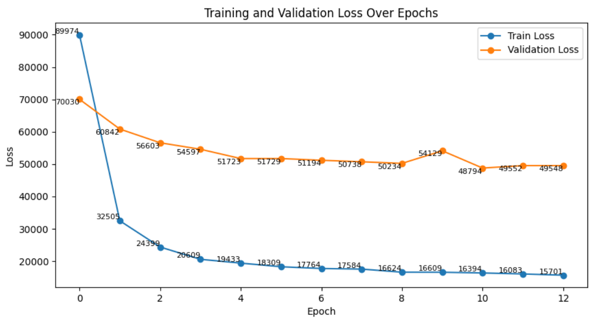

Fig. 9. Learning curve of the Encoder-based Transformer

C. GAT+Transformer

We consider introducing Graph Attention Networks (GAT) on top of the Transformer to extract spatial features from the adjacency matrix representing probe distances [1].

Node features $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\}$, $\vec{h}_i \in \mathbb{R}^F$ are input to the GAT layer. Each node is multiplied by a learnable weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$, and the attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ (consisting of a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$ and the activation function LeakyReLU) is applied to compute the attention coefficients

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) = \text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]\right).$$

These are normalized using the softmax function to get

$$\alpha_{ij} = \text{softmax}_j(e_{ij}).$$

Applying multi-head attention, the normalized coefficients of K attention mechanisms are used to compute the linear combination of neighboring node features, concatenated to form the output of the layer

$$\vec{h}'_i = \Vert_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j,$$

where k denotes the normalized coefficients and weight matrix of the k-th attention mechanism, $\Vert$ denotes concatenation, and $\mathcal{N}_i$ are the neighbors of node i.

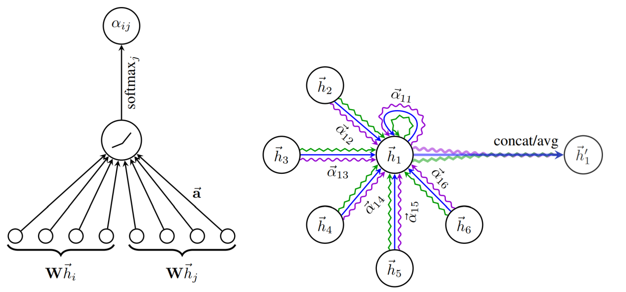

Fig. 10. Left: Calculation of the normalized attention coefficients α_{ij}; Right: Multi-head attention on node 1 across its neighbors (with $K = 3$ heads)

To combine the effects of GAT and Transformer, the output of the GAT layer is combined with the projected traffic flow time series information using a linear sum with a combination coefficient c

$$\mathbf{h}_{\text{Encoder input}} = c \cdot \mathbf{h}'_{\text{GAT}} + (1 - c) \cdot \mathbf{h}_{\text{proj}}$$

and input into the Transformer encoder to extract spatiotemporal features (see Figure 11).

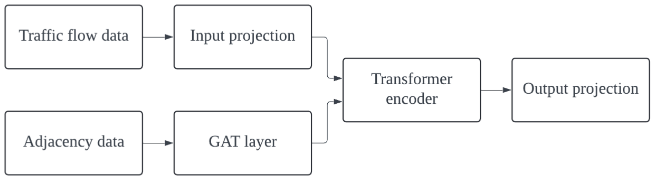

Fig. 11. Combination of GAT and Transformer

After completing the training of the model, we placed the output on a complete time series spanning from August 8, 2016, at 00:00, to August 14, 2016, at 23:00. To visually represent the performance, we selected data from five strategic traffic probes for analysis and illustrated these results in Figure 12. Throughout this week-long period, it was observed that the traffic flow exhibited seven distinct peaks, each showing varying trends according to the specific characteristics of the location and time. Notably, the traffic flow curves corresponding to different probes showed significant variability, demonstrating the complex dynamics of urban traffic patterns across different areas of the city.

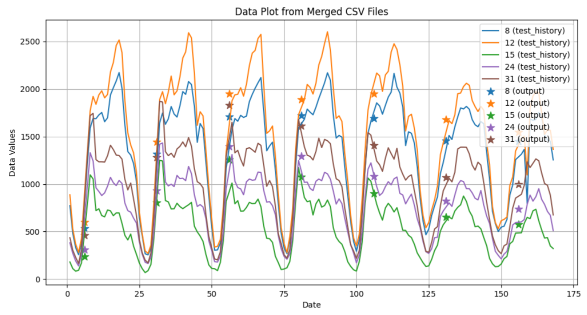

Fig. 12. Prediction results of GAT+Transformer

The integration of Graph Attention Networks (GAT) proved essential for capturing the spatial relationships inherent in the traffic data, allowing the Transformer encoder to leverage these spatial features effectively. As a result, the model could provide more nuanced and accurate predictions, clearly marked by stars in the graphical representation in Figure 12. This enhanced predictive capability was crucial for forecasting peak traffic times and potential congestion points.

Overall, the final validation loss of the model was significantly reduced to approximately 45744, a marked improvement over the baseline provided by an Encoder-based Transformer model. This improvement underscores the effectiveness of our approach in utilizing the distance information between probes to enhance prediction accuracy. Such advancements not only refine the model's ability to predict future traffic conditions but also contribute to the broader field of intelligent transportation systems by enhancing our understanding and management of urban traffic flows.

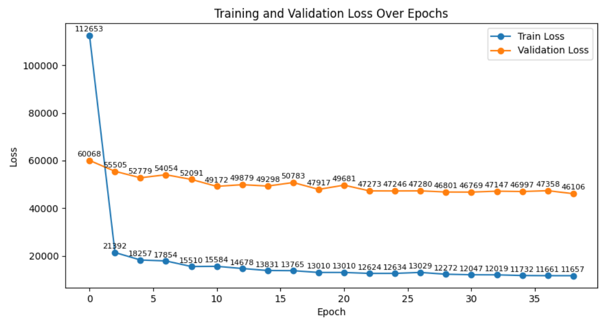

Fig. 13. Learning curve of GAT+Transformer

Our implementation of this model is available on Github.

D. AlexNet+Transformer

Besides the graph analysis models mentioned above, our group has also attempted other models.

AlexNet is a classic CNN model comprising eight layers. In the 2012 ImageNet Large Scale Visual Recognition Challenge, AlexNet achieved the lowest Top-5 error rate of 15.3%, which was 10.8% lower than the second place. In the architecture of AlexNet, the first five layers are convolutional layers, followed by some max pooling layers, with the last three layers being fully connected. It employs the unsaturated ReLU activation function, demonstrating better training performance than tanh and sigmoid.

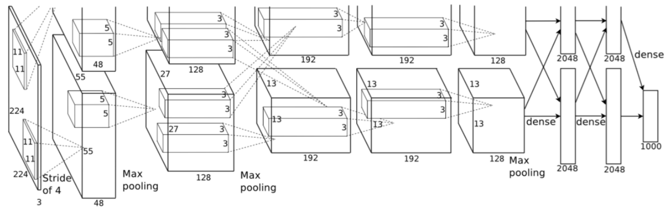

Fig. 14. AlexNet Architecture

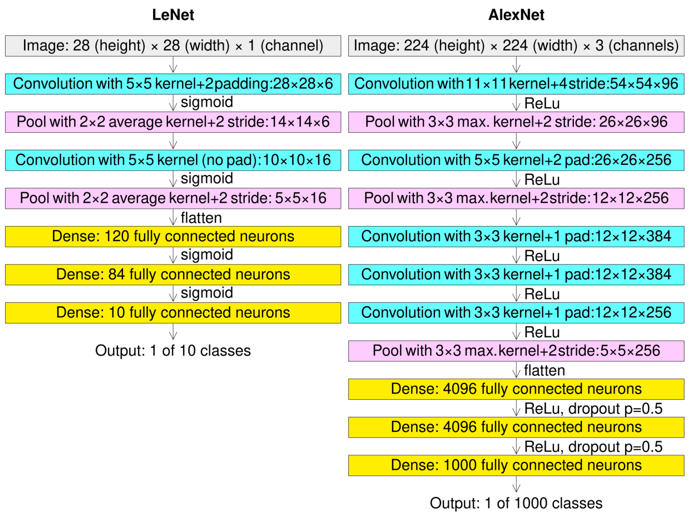
LeNet
AlexNet
Image: 28 (height) × 28 (width) × 1 (channel)
Convolution with 5×5 kernel+2padding:28×28×6
sigmoid
Pool with 2×2 average kernel+2 stride:14×14×6
Convolution with 5×5 kernel (no pad):10×10×16
sigmoid
Pool with 2×2 average kernel+2 stride: 5×5×16
flatten
Dense: 120 fully connected neurons
sigmoid
Dense: 84 fully connected neurons
sigmoid
Dense: 10 fully connected neurons
Output: 1 of 10 classes
Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
Output: 1 of 1000 classes

Fig. 15. Comparison of Architecture between AlexNet and LeNet

We add the features extracted by AlexNet on top of the Transformer input sequence, so as to merge graph features into the process of prediction.

After training, the validation loss of the model is as low as around 47100, which is better than the simple Transformer performance in the example code.

(We only present the true values of the corresponding probes over time in the training data once, and subsequent images are drawn based on the same set of labels.)

E. Encoder-Decoder Architecture Transformer

Based on the encoder-only Transformer example code provided by the instructor, our team implemented a

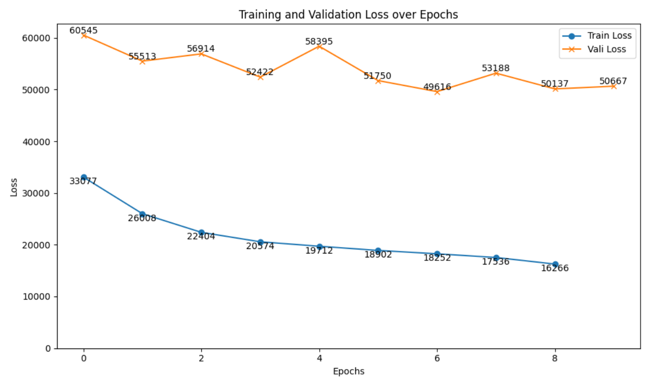

Fig. 16. the Learning Curve of AlexNet+Transformer

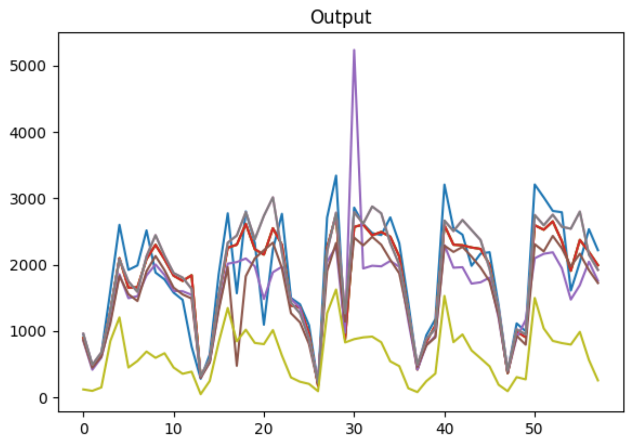

Fig. 17. AlexNet+Transformer Prediction Results

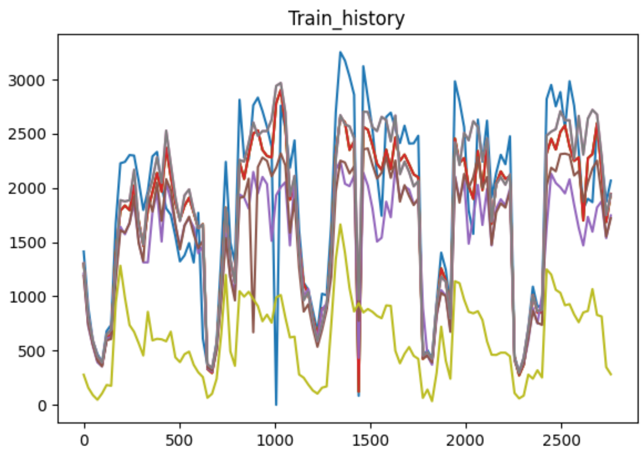

Fig. 18. Truth value of train data

complete encoder-decoder Transformer architecture, additionally incorporating positional encoding and masked softmax methods. The Transformer model architecture is illustrated in Figure 19. For specific implementation details and mathematical induction of the Transformer model, please refer to the paper ”Attention is all you need” [2].

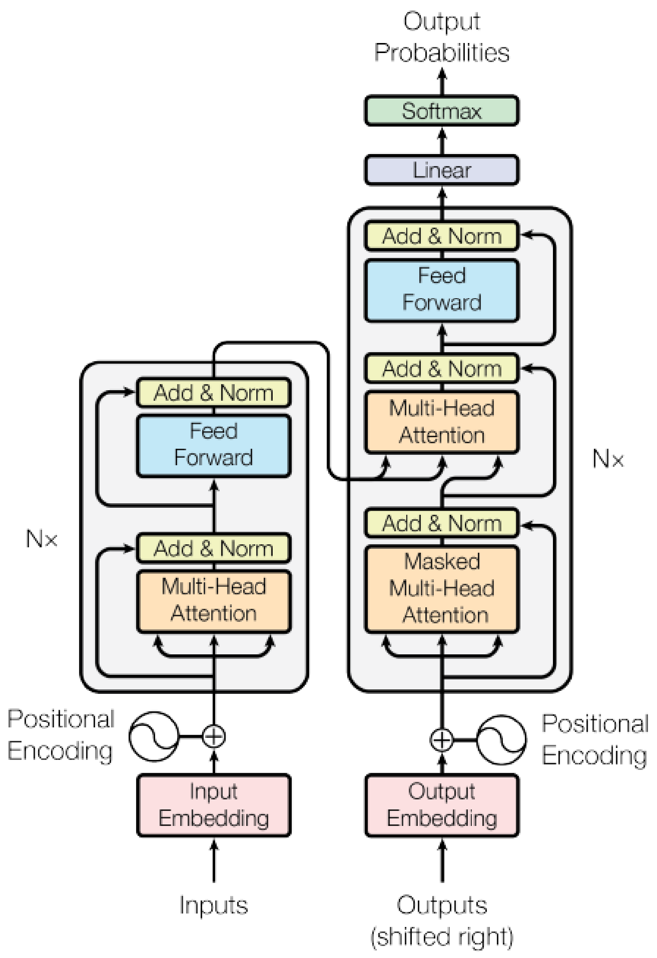

Fig. 19. Transformer Architecture

After implementing the complete Transformer architecture, the team proposed two approaches for training the model on the Paris traffic dataset:

1. Following the original Transformer training method for sequence-to-sequence tasks, the encoder input sequence for each iteration has a shape of [batch_size, seq_len, features], where data from 208 sensors at a single time step are used as features for that time step. The decoder input is the target sequence shifted forward by one time step, with a shape of [batch_size, pred_len, features]. The goal is to use teacher forcing to quickly teach the model to map the training sequence to the target sequence. In each iteration, the training data shape is [16, 24, 208], and the target sequence shape is [16, 1, 208]. The model consists of 2 encoder layers and 2 decoder layers, each with a hidden

dimension of 208. The multi-head attention mechanism uses 8 heads, the Positional FFN layer has a hidden dimension of 400, and the dropout rate is 0.1. During training, the Adam optimizer is used with a learning rate of 0.001. To evaluate the model performance compared to the example code, MSE loss is used as the loss function.

2. Referring to the training approach in the instructor-provided encoder-only Transformer example, the dataloader returns data in the shape of [batch_size, features, seq_len] for each iteration, which is then transformed by a linear layer to [batch_size, features, num_hiddens], enhancing the dimensions of key, query, and value in the self-attention layer to improve its learning capacity. Subsequently, a three-layer encoder is used to extract spatial relationships through the attention layer and temporal information through the FFN layer. Finally, the encoder output is used as input to a single-layer decoder, whose output, after being transformed, passes through a fully connected layer to obtain the prediction results. Other hyperparameters are consistent with those in approach 1.

The two approaches yielded different training results:

Approach 1: The large dimensions of query, key, and value caused slow convergence during training and gradient explosion problems, which were partially mitigated by introducing gradient clipping. The simple feedforward network structure failed to effectively capture the complex spatial features within the input data, leading to poor fitting of the output to the actual data. Despite repeated hyperparameter tuning, the model's MSE loss on the validation set could only converge to 90000-100000. Severe overfitting was also observed, which was partially alleviated by using the AdamW optimizer with a weight decay of 0.01. These issues indicate that the model did not achieve state-of-the-art (SOTA) results.

Approach 2: The model effectively captured both spatial and temporal features without requiring excessive hyperparameter tuning. The MSE loss on the validation set converged to 40000-50000, with no significant overfitting observed. Compared to approach 1, the model performance improved significantly, achieving SOTA results for traditional Transformers on single-step time series prediction tasks.

The following experimental results illustrate the significant differences in optimization performance between the two approaches:

In addition to these two approaches, the team also considered the following approach: directly importing the paris_adj adjacency matrix from the dataset and using it as input to the encoder. This approach is similar to Graph Attention Networks (GAT) and Graph Neural Networks (GNN), utilizing the Transformer's attention mechanism to fully capture the spatial relationships between sensors before passing the result to the decoder. The decoder input is the input sequence, which, after processing through the attention layers and feedforward networks, yields the output that is then transformed through a linear layer to

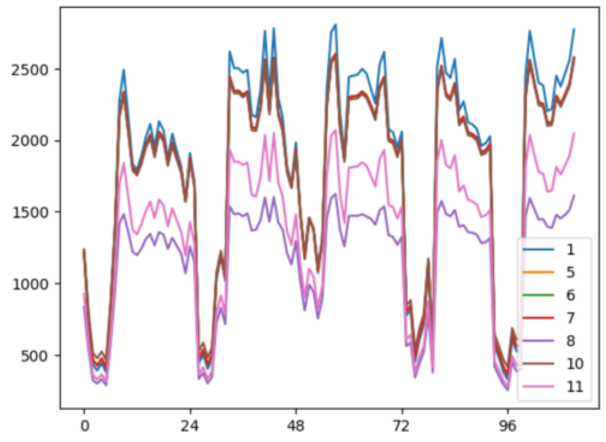

Fig. 20. Approach 1 prediction results

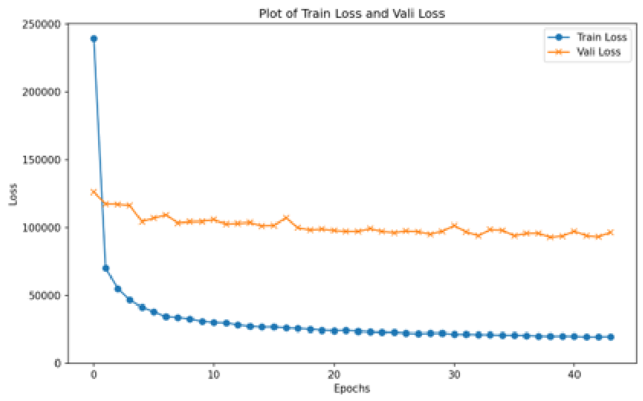

Fig. 21. Approach 1 training and validation losses

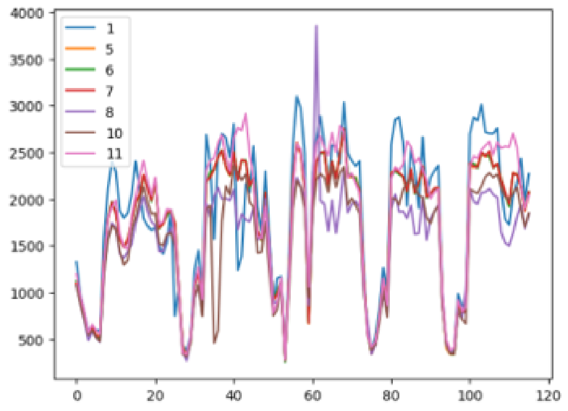

Fig. 22. Approach 2 prediction results

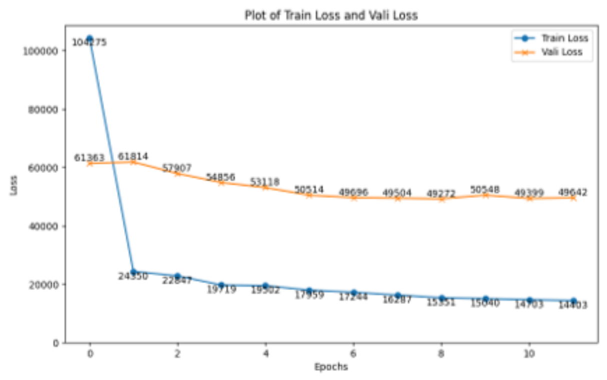

Fig. 23. Approach 2 training and validation losses

obtain the prediction results. However, due to the large dimension of the adjacency matrix (208*208), the convergence speed during model training was significantly slow, and the gradient explosion issue was severe. Consequently, the model performance was suboptimal, and hence, the results of this approach are not presented here.

F. The Improvement of the Standard Transformer Architecture

After implementing and comparing the above approaches, the team reviewed some shortcomings of the traditional Transformer. The standard PositionWiseFFN struggles to capture the complex nonlinear temporal features within the data and has low computational efficiency. Additionally, the traditional Transformer architecture, without special optimization, suffers from the issue of residual connection failure, making it challenging to simply stack encoder and decoder layers to enhance model performance. With fewer layers, relying on simple self-attention layers makes it difficult to extract spatial relationships within the data. Therefore, the team made the following improvements to the model, inspired by Tsinghua University's Haixu Wu's Time-Series-Library project on GitHub:

1. Replacing the traditional PositionWiseFFN with 2 1D Convolutional layers. By leveraging the inherent advantages of 1D Convolutional layers in .processing time-series data (the sliding convolutional kernels can effectively capture temporal dependencies within the sequence), this block improved computational efficiency and the ability to extract nonlinear features. What's more, a combination of 1D Convolutional layer, 1D BatchNorm layer, ELU layer, and 1D MaxPool layer for down sampling is optional.

2. Employing more modern self-attention mechanisms such as DSAttention, ProbAttention, FullAttention, etc. Referencing implementations from the Time-Series-Library, the team opted for DSAttention in their model to enhance the ability to capture spatial relationships within the data. Unlike the ordinary Attention mechanism that simply uses the scaled dot product of $\mathbf{Q}$ and $\mathbf{K}$ as the attention score, the DSAttention mechanism introduces two learnable parameters τ and δ to dynamically adjust the attention distribution, making it more flexible and capable of capturing complex, non-stationary patterns in the data.

$$\text{scores} = \mathbf{Q}\mathbf{K}^T$$

$$\text{scores} = \text{scores} \cdot \tau + \delta$$

$$\mathbf{A} = \text{Dropout}(\text{softmax}(\text{scores}))$$

$$\mathbf{V}' = \mathbf{A}\mathbf{V}$$

3. Modifying the Transformer's architecture to suit the current task (e.g., long/short-term sequence prediction, imputation, anomaly detection, classification, etc.) to improve the model's generalization ability.

For forecasting tasks, both long-term and short-term, the model employs a full encoder-decoder structure. This configuration includes embedding layers for both the encoder and decoder, multiple layers of attention mechanisms, and feed-forward networks. The encoder captures the temporal dependencies and contextual information from the input sequence, while the decoder generates forecasted values based on the encoder's output and its own previous outputs.

In contrast, for data imputation and anomaly detection tasks, the model simplifies to using only the encoder. The encoder processes the input data and directly projects the output through a linear layer to generate imputed values or detect anomalies. This streamlined approach allows for efficient handling of these tasks without the need for a complex decoder structure.

For classification tasks, the model also utilizes only the encoder. The encoder's output is processed through an activation function and dropout layer before being projected to the final classification output. This configuration ensures that the model can effectively handle classification tasks by leveraging the encoder's ability to capture rich features from the input sequence.

In addition to these improvements, the team is considering further incorporating other Transformer architecture variants in future research, such as the Temporal Fusion Transformer and Non-Stationary Transformer. The performance of these models will be presented in future research.

G. DLinear

So under the guidance of the teaching assistant, we also tried a quite different model: DLinear.

Are Transformers Effective for Time Series Forecasting? "[3]Is an article published in AAAI in 2023. The article discusses the effectiveness of Transformers in temporal prediction and proposes a simple network model that is not part of the Transformer series.

The main idea of the model is to decompose the time-series data into trend and residual terms, and pass them through fully connected layers respectively, and finally sum them up to obtain the prediction results. The trend term is obtained by averaging and pooling the original data, while the residual term is the difference between the original data and the pooled data. The most significant advantage of this model is its low memory usage, fast inference speed.

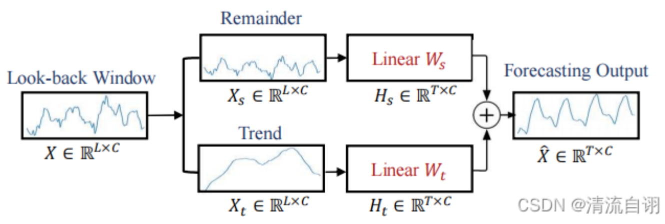
Remainder
Look-back Window
$X \in \mathbb{R}^{L\times C}$
$X_s \in \mathbb{R}^{L\times C}$
Linear W_s
$H_s \in \mathbb{R}^{T\times C}$
Forecasting Output
$\hat{X} \in \mathbb{R}^{T\times C}$
Trend
$X_t \in \mathbb{R}^{L\times C}$
Linear W_t
$H_t \in \mathbb{R}^{T\times C}$
CSDN @清流自诩

Fig. 24. DLinear Architecture

From the two charts of ouput, we can see that the DLinear model performs quite well in time series prediction: compared with previous transformer based models, it can provide relatively small validation losses even when the training epochs are small.

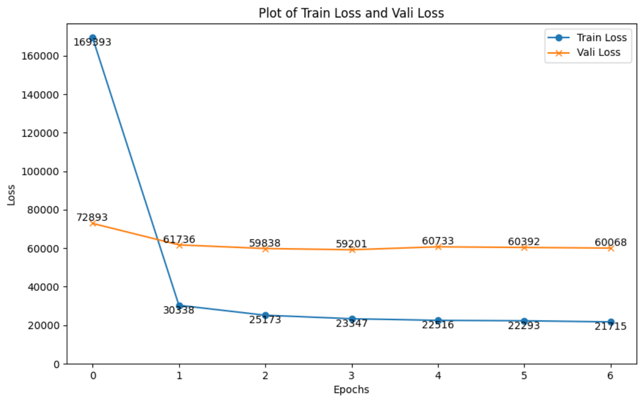

Fig. 25. the Learning Curve of DLinear Architecture

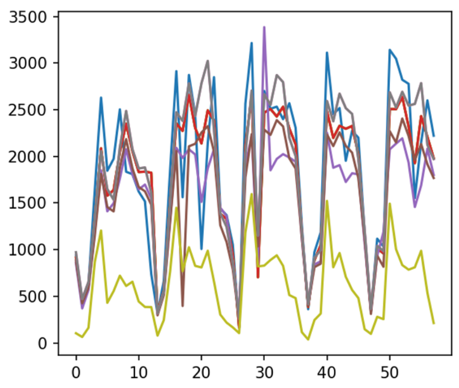

Fig. 26. the output of DLinear Architecture

We also conducted an analysis of the parameter selection for this model. Firstly, we fixed the prediction sequence length to 1 and plotted a testloss heatmap of the input sequence length and pooling kernel size. It can be found that for the given dataset, when $seg\_len = 24$ and the kernel size is relatively large, the loss is small. This indicates that increasing the pooling kernel size of seg_len at an appropriate size can to some extent remove the influence of noise.

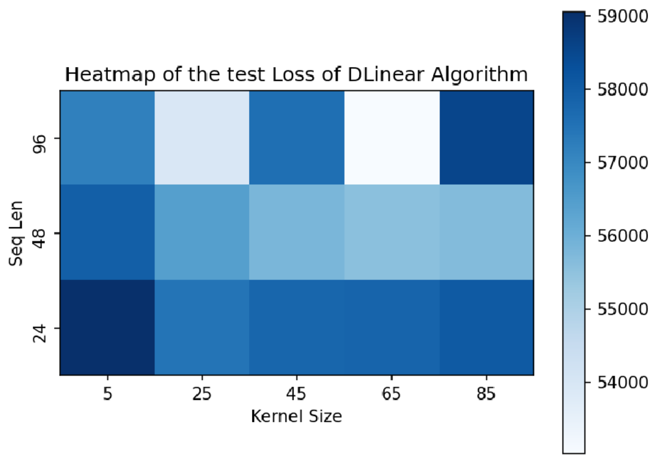

Fig. 27. Testloss Heatmap of the input Sequence Length and Pooling Kernel Size

Then we fixed input sequence length at 48, pooling kernel size at 65 and made a line chart about prediction length and validation loss. increased, the loss also gradually increased, which is very logical. Given the same amount of information, the reliability of the predicted value will also decrease.

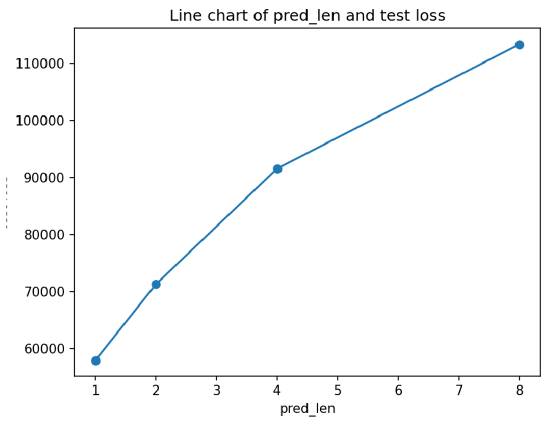

Fig. 28. the output of DLinear Architecture

III. Conclusions

A. Limitations of Traditional RNNs (LSTM/GRU) in Time Series Prediction

Traditional recurrent neural networks (RNNs) like LSTM and GRU are not ideally suited for time series prediction tasks. The information in the input sequences often gets lost after passing through several forget gates, preventing the model from capturing long-term dependencies effectively. Thus, specialized architectures are required when employing RNNs for time series prediction. In our future research, we plan to integrate proven models such as STGAN[4] to address these issues.

B. Effectiveness of Traditional Transformer Architecture

The traditional Transformer architecture has proven to be effective for simple time series prediction tasks. With the right training approach, its combination of multi-head self-attention mechanisms and feed-forward networks can efficiently capture both spatial and temporal information within sequences. Additionally, we have several strategies to further enhance model performance, such as incorporating GAT and AlexNet for extracting road network information before inputting it into the model to assist in spatial information learning; replacing the original fully connected feed-forward network with 1D convolutional layers to improve computational efficiency and non-linear feature extraction capabilities; and introducing more learnable parameters into the attention mechanism to enhance the model's expressiveness.

However, the Transformer model still faces several limitations. Increasing the model's depth can lead to gradient vanishing/explosion and residual connection failures, preventing simple depth augmentation for better fitting capacity. Moreover, the high computational complexity of the Transformer's self-attention mechanism necessitates consideration of computational efficiency in practical applications.

C. Emergence of New Fully Connected Network-Based Time Series Prediction Models

Recently, newly proposed fully connected network-based time series prediction models, such as TimeMixer and DLinear, have demonstrated superior performance compared to Transformer-based models that utilize attention mechanisms. In our study on the DLinear model, multi-scale sampling allows the model to better capture information at different levels of the sequence. Additionally, due to the use of fully connected layers, the model achieves higher computational efficiency compared to self-attention networks and does not suffer from residual connection failures, enabling increased model depth and improved fitting performance. Our research shows that after selecting the appropriate input sequence length and convolution kernel size, DLinear maintains good performance even as the predicted sequence length increases, highlighting the model's advantages in handling long sequences.

IV. Lessons Learned

In this project, we not only implemented and adjusted the model based on the characteristics of the dataset, but also gained a deeper understanding of the features and applicability of various neural network architectures, thus gaining a more comprehensive and modern understanding of deep learning. Through in-depth exploration of various models, including Transformer, we not only understand the theoretical principles of these models, but also experience the challenges of debugging in practice. In addition, we attempted to use various models, adjusted key parameters such as learning rate, and mastered how to control the decay of learning rate and gradient truncation techniques. We also practiced visualizing data and gained rich overall results, which significantly increased our team's technical capabilities.

We would like to express our special thanks to the teaching assistants and teachers for their patient guidance and professional advice. Their support has played a crucial role in helping us overcome the difficulties and challenges we encountered during the learning process.

V. Division of labour

Junyan Yu: GAT+Transformer, presentation preparation and report writing

Hongliang Lu: AlexNet+Transformer, LSTM/GRU, DLinear, presentation preparation and report writing

Xiaotao Zhang: Encoder-Decoder Architecture Transformer, improvement of standard Transformer, presentation preparation and report writing

References

[1] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks," in Proc. ICLR 2018, 2018.

[2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems, 2017, pp. 5998-6008

[3] Ailing Zeng and Muxi Chen and Lei Zhang and Qiang Xu "Are Transformers Effective for Time Series Forecasting?",Proceedings of the AAAI Conference on Artificial Intelligence,2023

[4] Zhang, Chenhan, James J. Q. Yu, and Yi Liu. 2019. "Spatial-Temporal Graph Attention Networks: A Deep Learning Approach for Traffic Forecasting." IEEE Access 7: 166246-166256.