Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting (*Yaguang Li et al., ICRL 2018)*

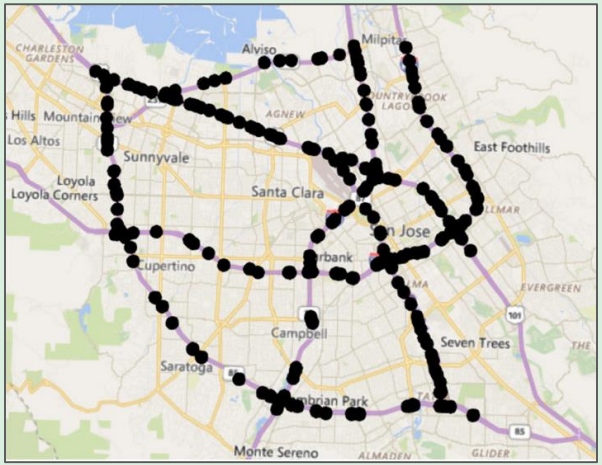
Presented by: LE VAN DUC (2019-38262)

Outline

- Introduction
- Graph Diffusion Convolution Definition
- Diffusion Convolutional Recurrent Neural Network (DCRNN)
- Detailed Implementation Explanation
- Training & Testing
- Real Experiments Running
- More Evaluations w/ SOTA methods
- Conclusions

Introduction

- The paper tries to solve **traffic forecasting** problems by:
  - A Diffusion Process on a Directed Graph to model the Spatial Correlations in Traffic Flow => **Diffusion Convolution**
  - An Encoder-Decoder architecture to capture the Temporal Dependency => **Diffusion Convolutional Gated Recurrent Unit (DCGRU)**



Traffic Sensor Network

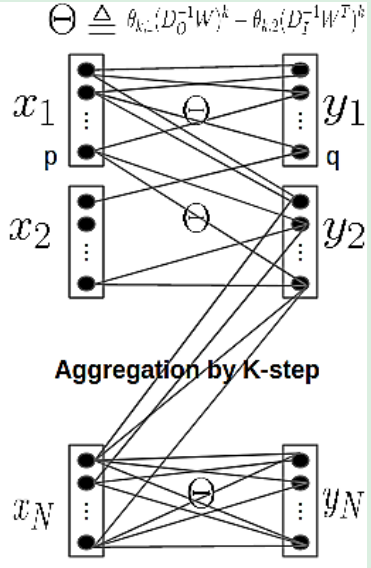
Problem Statement & Diffusion Convolution Definition

- **Traffic Forecasting Problem**:
  - Sensor networks as a weighted directed graph $G = (V, E, W), |V| = N, W \in R^{N \times N}$
  - V = {traffic sensors}, E = {roads between sensors},
  - Wij = road network distance
- **Diffusion Convolution**:
  - Model the spatial correlations of traffic flow as a Diffusion process by a Random Walk on G with a state transition matrix $D_O^{-1}W$
  - $D_O = diag(W)$ is the out-degree diagonal matrix.
  - Define the diffusion convolution over a graph signal X and a filter f as:

$$X *_G f_\theta = \sum_{k=0}^{K-1} (\theta_{k,1}(D_O^{-1}W)^k + \theta_{k,2}(D_I^{-1}W^T)^k)X$$

  - Θ are the learnable parameters, 2 transition matrices represent the bidirectional diffusion process. K is the diffusion steps of the random walk.

Diffusion convolution



Diffusion Convolutional Layer

- **Diffusion Convolutional (DiffConv) Layer**:
  - Build a diffusion convolutional layer that maps P-input features to Q-output features.

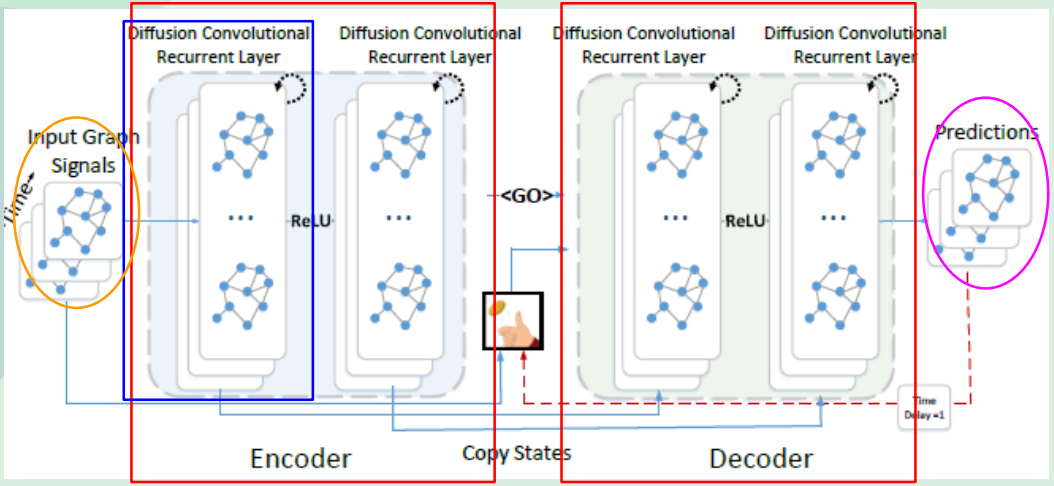
$$H_{:,q} = a(\sum_{p=1}^{P} X_{:,p} *_G f_\theta), p \in \{1, ..., P\}, q \in \{1, ..., Q\}$$

  - **X** is input, **H** is output, **$\{f_\theta\}$** are filters, **a** is the activation function (e.g. ReLU, Sigmoid).
  - This layer can be trained by a stochastic gradient based method.
- **Relation with Spectral Graph Convolution**:
  - Spectral Graph Convolution (e.g. ChebNet) can be considered a special case of DiffConv.
  - This paper proved that, the spectral graph convolution defined as

$$X_{:,p} \star_{\mathcal{G}} f_\theta = \Phi \, F(\theta) \, \Phi^\intercal X_{:,p}$$

  - With **$L = \Phi\Lambda\Phi^\intercal$** & **$F(\theta) = \sum_0^{K-1} \theta_k \Lambda^k$** (polynomial filter) is equivalent to graph DiffConv up to a similarity transformation, when G is undirected graph.

Diffusion Convolutional Recurrent Neural Network (DCRNN)



- Use Recurrent Neural Networks (e.g Gated Recurrent Unit, GRU) to model the temporal dependency.
- Replace the matrix multiplications in a GRU cell with DiffConv => **DiffConv GRU cells**.

$$r^{(t)} = \sigma(\Theta_r \star_{\mathcal{G}} [X^{(t)}, H^{(t-1)}] + b_r) \qquad u^{(t)} = \sigma(\Theta_u \star_{\mathcal{G}} [X^{(t)}, H^{(t-1)}] + b_u)$$
$$C^{(t)} = \tanh(\Theta_C \star_{\mathcal{G}} [X^{(t)}, (r^{(t)} \odot H^{(t-1)})] + b_c) \qquad H^{(t)} = u^{(t)} \odot H^{(t-1)} + (1 - u^{(t)}) \odot C^{(t)}$$

- For multiple times ahead traffic flow prediction, employ an Encoder-Decoder model with both networks are **DiffConv Recurrent layers** => **DCRNN model**.

Code Link & Experiments Results

- **Code Link: https://github.com/vanduc103/gcn2020_project**
- **You can check README file in the Code Link to see my Training files that show the Experiments Results.**
- **The instructions to prepare the data, construct the Graph, training, and testing are in the README file in the Code Link.**
- **Model training:**
  - **python dcrnn_train.py --config_filename=data/model/dcrnn_la.yaml**
- **Model testing (with 1 pre-trained model):**
  - **python run_demo.py --config_filename=data/model/dcrnn_DR_1_h_12_64-64_lr_0.01_bs_64_0527211535/config_100.yaml**

Detailed Implementation Explanation

- **Diffusion Filter** (file lib/utils.py)
  - Random Walk filter ($D_O^{-1}W$): see **code** on the right
  - Bi-directional RW filter ($D_I^{-1}W$ & $D_O^{-1}W$):
    - compute RW filter 2 times with adj_mx & adj_mx transpose.
  - Scaled Laplacian filter: compute Spectral Graph Conv (to compare w/ DiffConv)
  - Identity filter: ignore spatial filter by multiplying w/ identity matrices.

```
def calculate_random_walk_matrix(adj_mx):
    adj_mx = sp.coo_matrix(adj_mx)
    d = np.array(adj_mx.sum(1))
    d_inv = np.power(d, -1).flatten()
    d_inv[np.isinf(d_inv)] = 0.
    d_mat_inv = sp.diags(d_inv)
    random_walk_mx = d_mat_inv.dot(adj_mx).tocoo()
    return random_walk_mx
```

represent adjacency matrix in sparse form

compute out-degree diagonal matrix (D_O^{-1})

$D_O^{-1}W$ as sparse-dense matrix multiplication

- **DiffConv GRU Cell** (file model/dcrnn_cell.py)
  - GRU Gates computation by DiffConv:
    - Reset Gate (**r**), Update Gate (**u**)
    - Candidate (**C**) and Output (**H**)
  - Input, Output:
    - Input: Batch of Historical Traffic Time series
    - Input Shape: (B, num_nodes * input_dim)
    - Output: Batch of Predicted Values
    - Output Shape: (B, output_size)

```
value = tf.nn.sigmoid(fn(inputs, state, output_size, bias_start=1.0))
value = tf.reshape(value, (-1, self._num_nodes, output_size))
r, u = tf.split(value=value, num_or_size_splits=2, axis=-1)
```

```
with tf.variable_scope("candidate"):
    c = self._gconv(inputs, r * state, self._num_units)
    if self._activation is not None:
        c = self._activation(c)
output = new_state = u * state + (1 - u) * c
```

```
fn = self._gconv
```

```
for support in self._supports:
    x1 = tf.sparse_tensor_dense_matmul(support, x0)
    x = self._concat(x, x1)

    for k in range(2, self._max_diffusion_step + 1):
        x2 = 2 * tf.sparse_tensor_dense_matmul(support, x1) - x0
        x = self._concat(x, x2)
        x1, x0 = x2, x1
```

Code for DiffConv: Recursive Sparse-Dense Matrix Multiplication with Max-Diffusion-Step (1,2,3…) => **O(KN)**

Diffusion Convolutional Recurrent Neural Network Model (DCRNN Model)

- **DCRNN Model** (file model/dcrnn_model.py)
  - Build DiffConv GRU Cell and stack many cells to build Encoder and Decoder layers:

```
cell = DCGRUCell(rnn_units, adj_mx, max_diffusion_step=max_diffusion_step, num_nodes=num_nodes,
encoding_cells = [cell] * num_rnn_layers
decoding_cells = [cell] * (num_rnn_layers - 1) + [cell_with_projection]
encoding_cells = tf.contrib.rnn.MultiRNNCell(encoding_cells, state_is_tuple=True)
decoding_cells = tf.contrib.rnn.MultiRNNCell(decoding_cells, state_is_tuple=True)
```

  - Some **important parameters**:
    - **max_diffusion_steps**: defines number of diffusion steps (K-hop) we would run by random walk from a node. Default is 2 diffusion steps.
    - **filter_type**: defines which Diffusion Filter we will use. Default is "dual_random_walk" that applies bi-directional RW filter. There are 3 other filters: "random_walk", "scaled_laplacian" (for Spectral GNN computation), "identity" (for not applying spatial learning in DCRNN model)
    - **horizon**: number of time intervals will be forecasting (each time interval is 5 minutes). Default is 12 horizons that means forecasting from 5 minutes to 1 hour.

Training and Testing

- **Data Preparation**: collect traffic sensors data and generate training set
  - Traffic sensors data: collect traffic sensors data from Los Angeles (METR-LA) and the Bay Area (PEMS-BAY). Each sensor has time interval 5 minutes.
  - Generate training data: generate train/val/test data by aggregating some time windows (here is 12 time windows of each 5 minutes = total 1 hour prediction)
- **Graph Construction**:
  - Generate Adjacency Matrix as Weighted Distance Matrix between pairwise road sensors. Choose threshold K (=0.1) to make graph sparse.
- **Model Training**
  - Training configuration: in file .yaml, including model and training hyperparameters.
  - Loss function: Use MAE (Mean Absolute Error) Loss
  - Training monitoring: in file info.log
- **Model Evaluation**
  - Baseline methods evaluation: HA (Historical Average), VAR (Vector Auto-Regressive), Static Prediction (fixed prediction)
  - Metric: there are 3 evaluation metrics: MAE metric, MAPE (Mean Absolute Percentage Error) metric, RMSE (Root Mean Squared Error) metric.

Running Experiments

- Recover "**comparison performance**" of DCRNN model (with forecasting time T = 15', 30', 1h) as in the paper: use dual_random_walk filter, max_diffusion_step = 2, horizon = 12.

Dataset	T	Metric	Static	HA	VAR	DCRNN
METR-LA	15'	MAE RMSE MAPE	4.02 8.69 9.39%	4.15 7.77 12.90%	4.37 7.78 10.08%	**2.67** **5.19** **6.88%**
	30'	MAE RMSE MAPE	5.09 11.13 12.21%	4.15 7.77 12.90%	5.40 9.37 12.75%	**3.08** **6.33** **8.41%**
	1h	MAE RMSE MAPE	6.79 14.21 16.71%	4.15 7.77 12.90%	6.50 10.68 15.84%	**3.58** **7.56** **10.33%**

- Experiment to investigate the **effect of diffusion convolution** (spatial dependence modeling): NoDiffConv (use Identity filter), UniDiffCov (use RW filter), FullDiffConv (use Bi-directional RW filter)

Dataset	T	Metric	NoDiffConv	UniDiffConv	FullDiffConv
METR-LA	15'	MAE RMSE MAPE	2.98 5.90 7.90%	2.72 5.25 7.06%	**2.67** **5.19** **6.88%**
	30'	MAE RMSE MAPE	3.58 7.29 10.22%	3.14 6.39 8.67%	**3.08** **6.33** **8.41%**
	1h	MAE RMSE MAPE	4.45 8.99 13.77%	3.64 7.62 10.72%	**3.58** **7.56** **10.33%**

- Experiment to study the **effect of graph construction**: Undirected (use Laplacian filter or Spectral GNN) vs. Directed Graph (use Laplacian filter or DiffConv with Bi-directional RW filter)

Dataset	T	Metric	Undirected (Laplacian)	Directed (Laplacian)	Directed (DiffConv)
METR-LA	15’	MAE RMSE MAPE	2.72 5.25 7.06%	2.68 5.22 6.91%	**2.67** **5.19** **6.88%**
	30’	MAE RMSE MAPE	3.14 6.39 8.67%	**3.08** 6.34 8.46%	**3.08** **6.33** **8.41%**
	1h	MAE RMSE MAPE	3.64 7.62 10.72%	**3.57** **7.56** 10.40%	3.58 **7.56** **10.33%**

- Experiment to examine the **effect of Diffusion Steps** (K-hop): vary max_diffusion_step from 1 to 5.

Dataset	T	Metric	K = 1	K = 2	K = 3	K = 4	K = 5
METR-LA	15’	MAE RMSE MAPE	2.68 **5.16** **6.87%**	**2.67** 5.19 6.88%	2.69 5.23 6.92%	2.74 5.33 7.04%	2.97 5.74 8.19%
	30’	MAE RMSE MAPE	3.09 **6.27** 8.48%	**3.08** 6.33 **8.41%**	3.09 6.35 8.44%	3.15 6.47 8.59%	3.50 7.24 10.70%
	1h	MAE RMSE MAPE	**3.58** **7.49** 10.50%	**3.58** 7.56 **10.33%**	**3.58** 7.58 10.34%	3.63 7.70 10.50%	4.22 9.06 14.07%

More Evaluations

- **Compare with 2 recent SOTA methods**
  - Yu et al., Spatio-Temporal Graph Convolutional Networks: A De Forecasting, IJCAI 2018
    - Use common Spectral filter (e.g in ChebNet) and generalize to multi-dimensional tensors.
    - Do not use RNN because of time-consuming recurrent computation. Use only Conv Block with the combination of Graph Spatial Conv and Gated Temporal Conv.
  - Diao et al., Dynamic Spatial-Temporal Graph Convolutional Neural Networks for Traffic Forecasting, AAAI 2019
    - Learn the Laplacian Matrix for traffic networks by specific time-of-day dynamically (not assume adjacency matrix unchanged as previous) => Dynamic Laplacian Matrix Estimator.
    - Other layers similar to previous methods (a ConvBlock = Graph Conv + Gated Temporal Conv)
- **Apply with other spatiotemporal datasets** (crowd flows)
  - Dataset description: Taxi Trajectory Dataset in Beijing (2014). Data = Inflow + Outflow at a Region at a Time.
  - Data preparation (in folder data/beijing2014):
    - Generate training data by time windows 30 minutes.
    - Compute Adjacency Matrix as Weighted Distance Matrix of pairwise region distance.
    - Training & Testing with taxi trajectory data (future plan)

Conclusions

- This is one of the first papers that proposes to apply Graph Diffusion Convolutional Networks into a spatiotemporal problem like Traffic Forecasting.
- The leverage of Diffusion Convolution has given good forecasting results.
- Recent SOTA methods have pointed out and solved some weak points of this models, e.g. a time-consuming recurrent network computation or an impractical static adjacency matrix for dynamic problems like traffic.
- **GNN** has promising power in solving real-life problems. I hope that we can successfully apply GNN in many real-world problems in the future.

Thank you for reading!

HAVE A GOOD DAY!