Inductive Graph Neural Network Kriging

Gang Su

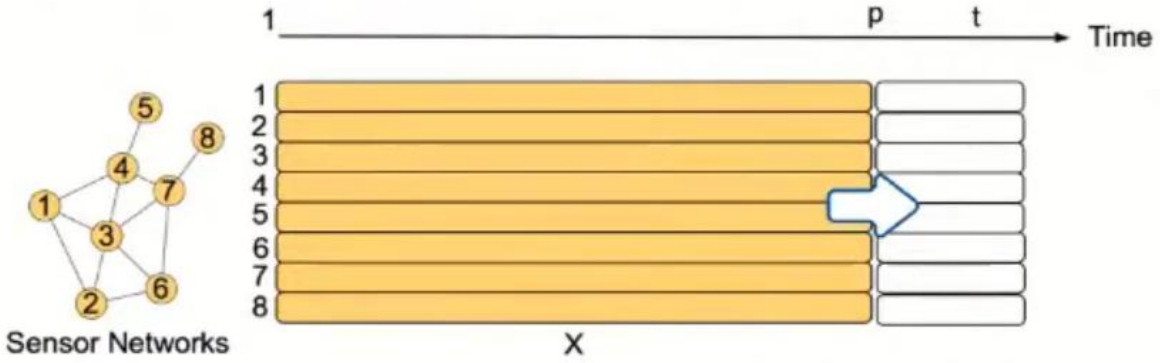
Agenda

1. Background and Motivation
2. Methodology
3. Experiments

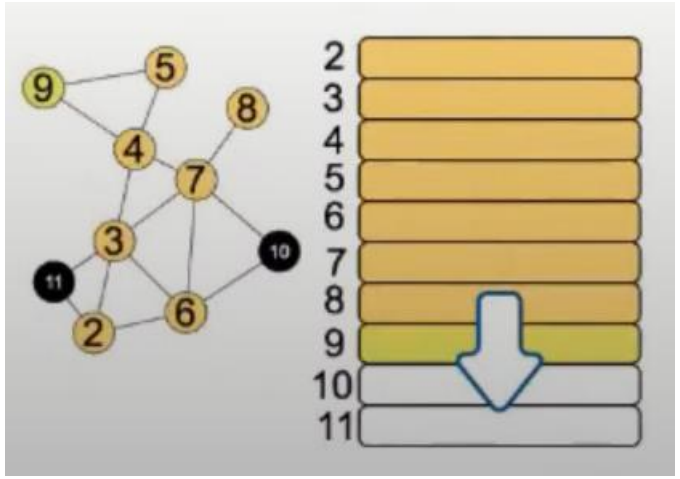
Background

- **Forecasting**: Estimation for future information given the observed signals from history.

[GNN&LSTM Cui 2018], [STGCN Yu 2018], [DCRNN Li 2018], [GraohWavenet Wu 2019], [Attention based STGCN Guo 2019]



- **Kriging**: Signal interpolation for unsampled locations given the observed signal from sampled locations during the same period.



Motivations

- **Applications in smart cities**: How to infer traffic states of unknown locations given limited number of sensors in the network?

Traditional Kriging Techniques:

1. Gaussian Process
2. Matrix/Tensor Completion

Problems:

1. They are essentially transductive: if adding a new sensor or some sensor failure, we have to completely retrain the model.
2. GP can not effectively deal with large-scale graph datasets.

Motivations

- Recent research on GNNs shows the power in characterizing complex spatial dependencies.

  [*How Powerful are Graph Neural Networks? (2018)]*

- GNNs also exhibit the inductive power to generalize the message assign mechanism to unseen nodes or even entirely new graphs.

  [GraphSAGE (2017), GAT (2018), GrahSAINT (2020)]

- GNNs can effectively deal with large-scale network.

  ***Node Sampling**: GraphSAGE (2017), VR-GCN(2018), PinSAGE (2018)*

  ***Layer Sampling**: FastGCN (2018), AS-GCN(2018), LADIES (2019)*

  ***Subgraph Sampling**: ClusterGCN (2019), **GrahSAINT** (2020)]*

Background: Spectral GCNs

- General Spectral Approach:

$$x \star_G g = \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(g)) = U(U^T x \odot U^T g)$$

$$\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

- ChebNet:

$$x \star_G g_\theta = U g_\theta U^T x = U \begin{pmatrix} \hat{g}(\lambda_1) & & \\ & \ddots & \\ & & \hat{g}(\lambda_n) \end{pmatrix} \begin{pmatrix} \hat{x}(\lambda_1) \\ \hat{x}(\lambda_2) \\ \vdots \\ \hat{x}(\lambda_n) \end{pmatrix} = U \begin{pmatrix} \hat{g}(\lambda_1) & & \\ & \ddots & \\ & & \hat{g}(\lambda_n) \end{pmatrix} U^T x$$

$$g_\theta = \begin{pmatrix} \hat{g}(\lambda_1) & & \\ & \ddots & \\ & & \hat{g}(\lambda_n) \end{pmatrix} \rightarrow g_\theta = \begin{pmatrix} \beta_0 T_0(\hat{\lambda}_1) + \beta_1 T_1(\hat{\lambda}_1) & & \\ & \ddots & \\ & & \beta_0 T_0(\hat{\lambda}_n) + \beta_1 T_1(\hat{\lambda}_n) \end{pmatrix}$$

$$x \star_G g_\theta = U g_\theta U^T x = \sum_{k=0}^{K} \beta_k T_k(\hat{L}) x$$

- GCN: (only one rank approximating)

$$\begin{aligned} x \star_G g_\theta &= \sum_{k=0}^{K} \beta_k T_k(\hat{L}) x = \sum_{k=0}^{1} \beta_k T_k(\hat{L}) x \\ &= \beta_0 T_0(\hat{L}) x + \beta_1 T_1(\hat{L}) x \\ &= (\beta_0 + \beta_1 \hat{L}) x \\ &= (\beta_0 + \beta_1 (L - I_n)) x \\ &= (\beta_0 - \beta_1 (D^{-1/2} W D^{-1/2})) x \end{aligned}$$

$$x \star_G g_\theta = (\beta_0 - \beta_1 (D^{-1/2} W D^{-1/2})) x = (\theta (D^{-1/2} W D^{-1/2} + I_n)) x$$

acency

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$I_n + D^{-1/2} W D^{-1/2} \rightarrow \tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2}$$

$$\tilde{W} = W + I_n \quad \tilde{D}_{ii} = \sum_i \tilde{W}_{ij}$$

$$0 \quad 0 \quad -1 \quad 3 \quad -1 \quad -1$$

$$x \star_G g_\theta = \theta \left(\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2} \right) x$$

Bruna, Joan et al. “Spectral Networks and Locally Connected Networks on Graphs.” (2014)

Micha ̈el Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks ongraphs with fast localized spectral filtering." (NIPS 2016).

Kipf, Thomas N., and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." (ICLR 2016).

Background: Spectral GCNs

- ChebNet:

```
def forward(self, X, A_hat):
    list_cheb = list()
    for k in range(self.orders):
        if (k==0):
            list_cheb.append(torch.diag(torch.ones(A_hat.shape[0],)))
        elif (k==1):
            list_cheb.append(A_hat)
        else:
            list_cheb.append(2*torch.matmul(A_hat, list_cheb[k-1])  - list_cheb[k-2])

    features = list()
    for k in range(self.orders):
        features.append(torch.einsum("kk,bkj->bkj", [list_cheb[k], X]))
    features_cat = torch.cat(features, 2)
    t2 = torch.einsum("bkj,jh->bkh", [features_cat, self.Theta1])
    t2 += self.bias
    if self.activation == 'relu':
        t2 = F.relu(t2)
    if self.activation == 'selu':
        t2 = F.selu(t2)
    return t2
```

$$x \star_G g_\theta = U g_\theta U^T x = \sum_{k=0}^{K} \beta_k T_k\left(\hat{L}\right) x$$

- GCN:

```
def forward(self, X, A_hat):
    """
    :param X: Input data of shape (batch_size, num_nodes, num_timesteps)
    :A_hat: The normalized adajacent matrix (num_nodes, num_nodes)
    :return: Output data of shape (batch_size, num_nodes, num_features)
    """
    features = torch.einsum("kk,bkj->bkj", [A_hat, X])
    t2 = torch.einsum("bkj,jh->bkh", [features, self.Theta1])
    t2 += self.bias
    if self.activation == 'relu':
        t2 = F.relu(t2)
    if self.activation == 'selu':
        t2 = F.selu(t2)

    return t2
```

$$x \star_G g_\theta = \theta\left(\tilde{D}^{-1/2}\tilde{W}\tilde{D}^{-1/2}\right)x$$

Limitations of Spectral Approaches

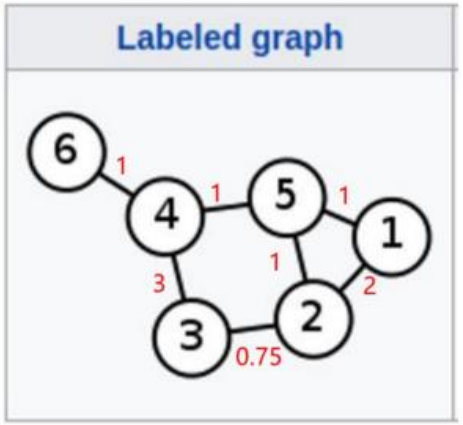
- Spectral models are less efficient than spatial models. Spectral models either need to perform eigenvector computation or handle the whole graph at the same time.

- Spectral-based model cannot operate on direct graph ($w_{ij} \neq w_{ji}$). (Graph Fourier Transform)

- Spectral-based model is transductive (cannot be applied when graph changed).

Spatial Approaches (Sampling)

- Uniform Sampling (GraphSAGE)
- Random Walk (Diffusion CNN)



A

0	2.0000	0	0	1.0000	0
2.0000	0	0.7500	0	1.0000	0
0	0.7500	0	3.0000	0	0
0	0	3.0000	0	1.0000	1.0000
1.0000	1.0000	0	1.0000	0	0
0	0	0	1.0000	0	0

D

3.0000	0	0	0	0	0
0	3.7500	0	0	0	0
0	0	3.7500	0	0	0
0	0	0	5.0000	0	0
0	0	0	0	3.0000	0
0	0	0	0	0	1.0000

P = $D^{-1}A$

0	0.6667	0	0	0.3333	0
0.5333	0	0.2000	0	0.2667	0
0	0.2000	0	0.8000	0	0
0	0	0.6000	0	0.2000	0.2000
0.3333	0.3333	0	0.3333	0	0
0	0	0	1.0000	0	0

P^3

0.1185	0.3970	0.0889	0.1659	0.2074	0.0222
0.3176	0.1185	0.2462	0.0593	0.2086	0.0498
0.0711	0.2462	0	0.6471	0.0356	0
0.0996	0.0444	0.4853	0	0.2213	0.1493
0.2074	0.2607	0.0444	0.3689	0.1185	0
0.0667	0.1867	0	0.7467	0	0

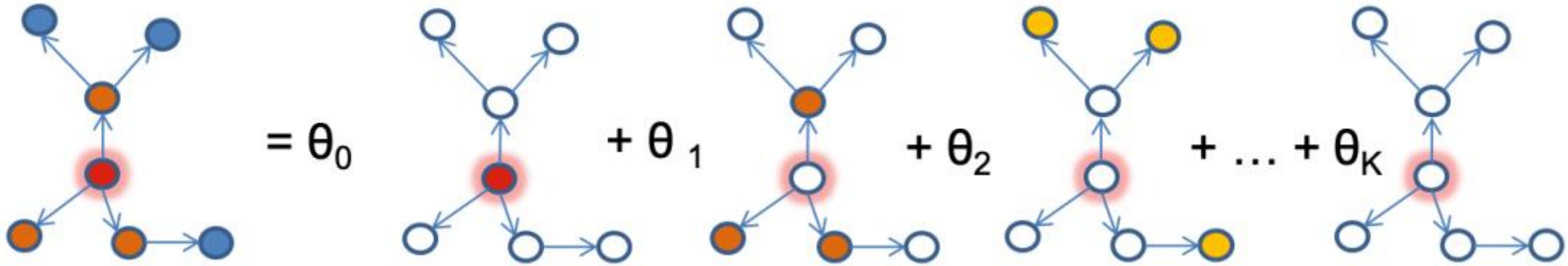
Spatial Approaches (Aggregate)

- Mean, LSTM and Pool Aggregate (GraphSAGE)
- Concatenate all hidden matrix H (DCNN)
- Sum up all hidden matrix H (DCRNN)

$$\mathbf{H}^{(k)} = f(\mathbf{W}^{(k)} \odot \mathbf{P}^k\mathbf{X}),$$

Note in Diffusion Process, the hidden presentation matrix $H^{(k)}$ remains the same dimension as the input feature matrix X and is not a function of its function of its previous hidden representation matrix $H^{(k-1)}$.

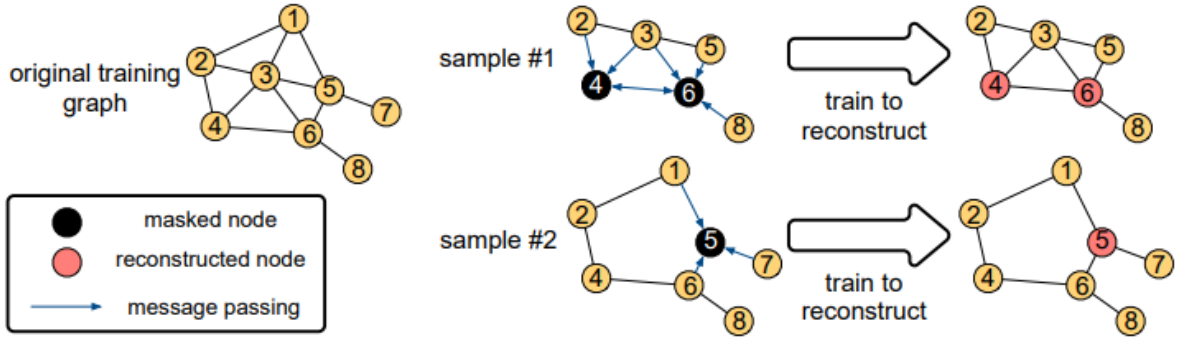
$$\mathbf{H} = \sum_{k=0}^{K} f(\mathbf{P}^k\mathbf{X}\mathbf{W}^{(k)}),$$



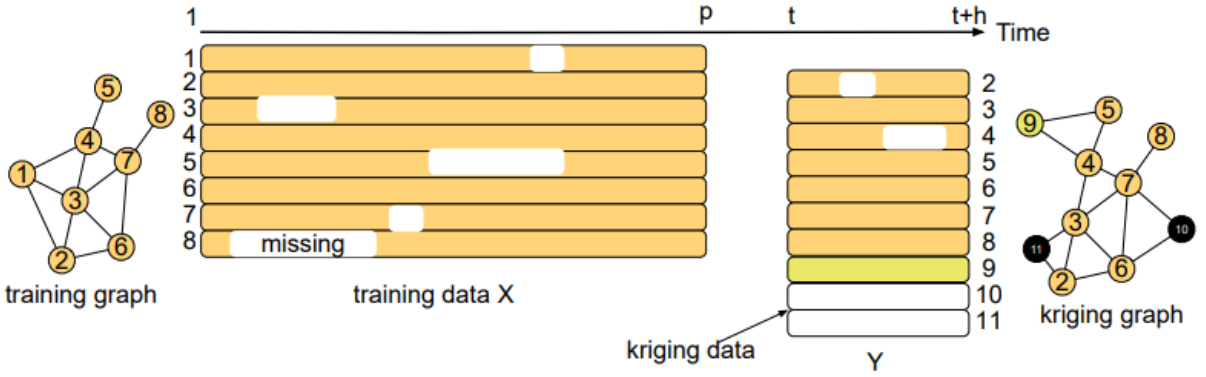
Methodology

- Inductive Graph Neural Networks for Kriging (IGNNK)

$$J = \sum \|\hat{X}_{\text{sample}} - X_{\text{sample}}\|_F^2.$$



(a) Training process of IGNNK

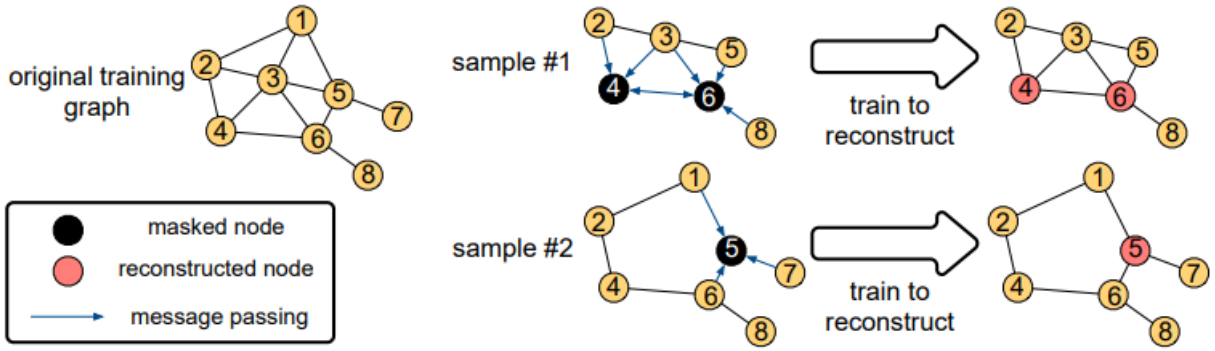


(b) Kriging process of IGNNK

- Reconstruct information on random subgraph structures, instead of a fixed graph.
- Generalized to unseen nodes/graph.
- To make the learned networks more generalized for all nodes, we use the total reconstruction errors on all nodes as objective function.

Algorithm

- 1. Randomly sampling a subgraph of the available spatiotemporal datasets.
- 2. Constructing an adjacency matrix according to the spatial relationships of sampled locations.
- 3. Randomly masking signals of parts nodes as 0s (simulating the unknown locations)
- 4. Train a GNN to reconstruct the whole sampled graph signal.



(a) Training process of IGNNK

GNN Architecture

$$H_{l+1} = \sum_{k=1}^{K} T_k(\bar{W}_f) H_l \Theta_{b,l}^k + T_k(\bar{W}_b) H_l \Theta_{f,l}^k,$$

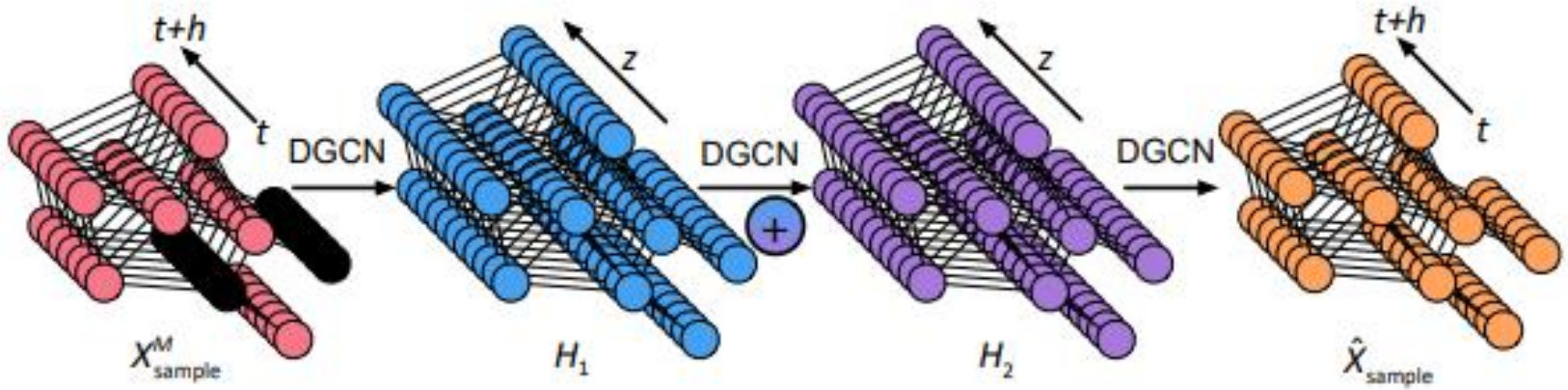


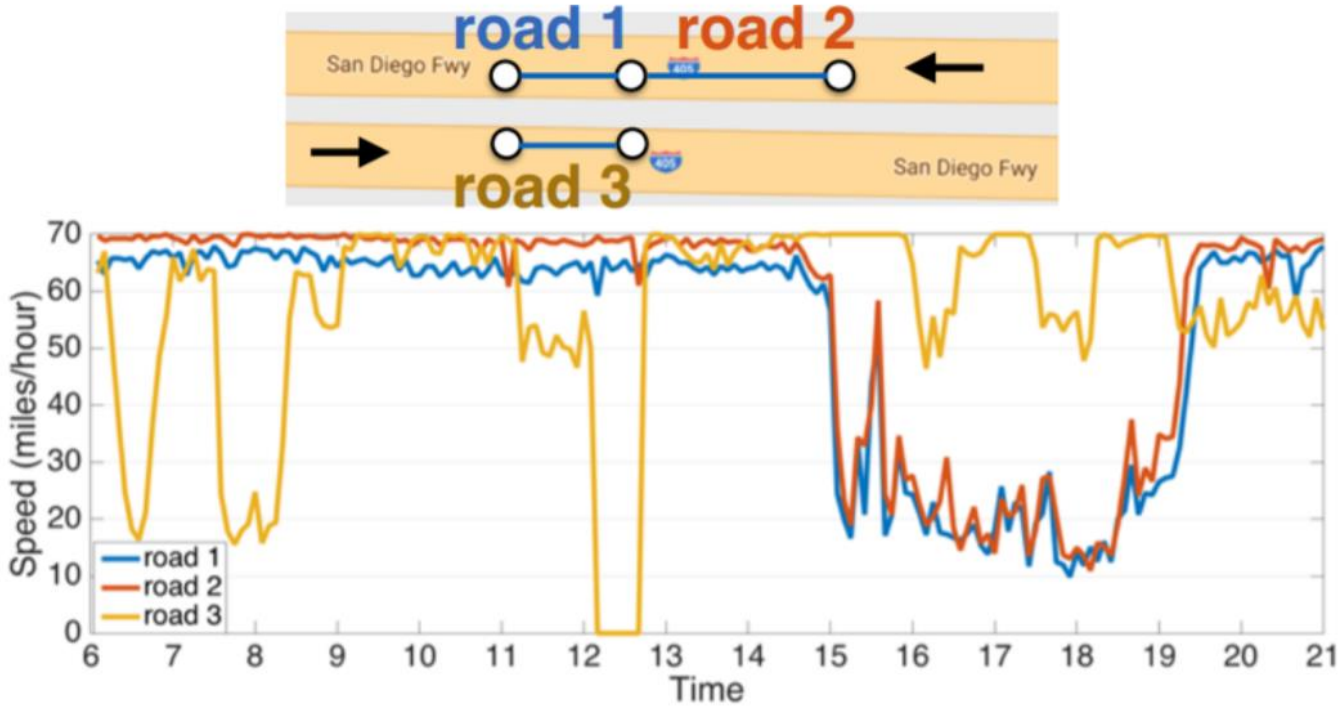
Figure 2: Graph neural network structure of IGNNK

$$T_k(X) = 2XT_{k-1}(X) - T_{k-2}(X) \qquad T_0(X) = I \text{ and } T_1(X) = X$$

$$W_{ij} = \exp\left(-\left(\frac{\text{dist}(v_i, v_j)}{\sigma}\right)^2\right)$$

Wf = forward transition matrix
Wb = backward transition matrix

Asymmetric distance matrix



$$W_{ij} = \exp\left(-\left(\frac{\text{dist}\,(v_i, v_j)}{\sigma}\right)^2\right)$$

Wf = forward transition matrix
Wb = backward transition matrix

- Road 1 and road 3 locate in the opposite direction of the highway. Though close to each other in the Euclidean space, their road network distance is large, and their traffic speeds differ significantly.

Experimental Results

Model	METR-LA			NREL			USHCN			SeData		
	RMSE	MAE	R^2	RMSE	MAE	R^2	RMSE	MAE	R^2	RMSE	MAE	R^2
IGNNK	**9.048**	**5.941**	**0.827**	**3.261**	**1.597**	**0.885**	**3.205**	2.063	**0.771**	**6.863**	**4.241**	**0.537**
kNN	11.071	6.927	0.741	4.192	2.850	0.810	3.400	2.086	0.742	-	-	-
KPMF	12.851	7.890	0.652	8.771	7.408	0.169	6.663	4.847	0.011	13.060	8.339	-0.673
GLTL	9.668	6.559	0.803	4.840	3.372	0.747	5.047	3.396	0.432	6.989	4.285	0.520
OKriging	-	-	-	3.470	2.381	0.869	3.231	**1.999**	0.767	-	-	-

Table 1: Kriging performance comparison of different models on four datasets.

- **METR-LA** is a traffic speed dataset from 207 sensors in Los Angeles (Mar 1, 2012 to Jun 30, 2012);
- **NREL** registers solar power out put by 137 photovoltaic power plants in Alabama state in 2006;
- **USHCN** contains monthly precipitation of 1218 locations from 1899 to 2019;
- **SeData** is also a traffic speed dataset collected from 323 loop detectors in the Seattle highway network

- **KPMF:** Kernelized Probabilistic Matrix Factorization
- **GLTL:** Greedy Low-rank Tensor Learning

Experiments Transfer to the same type dataset: PeMS

- Transfer to the same type dataset: PeMS (Traffic speed datasets)

Model	Gaussian				Binary			
	RMSE	MAE	MAPE	R^2	RMSE	MAE	MAPE	R^2
IGNNK	**6.093**	**3.663**	**8.16%**	**0.574**	9.245	5.394	13.26%	0.161
kNN	7.431	4.245	9.13%	0.458	-	-	-	-
KPMF	7.332	4.293	9.21%	0.472	10.065	5.985	16.03%	0.005
GLTL	8.846	4.486	10.25%	0.232	8.504	4.962	12.24%	0.290
IGNNK Transfer	METR-LA				SeData			
	6.713	**4.173**	**9.19%**	**0.525**	11.484	6.456	15.10%	-0.388

Table 2: Kriging performance of different models on PeMS-Bay. The last two rows shows the transferability of the two IGNNK models trained on METR-LA and SeData.

- In general, Gaussian kernel offers better performance than Binary adjacency matrix. The results confirm the critical role of distance in kriging task.

$$W_{ij} = \exp\left(-\left(\frac{\text{dist}(v_i, v_j)}{\sigma}\right)^2\right) \qquad W_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are neighbors,} \\ 0, & \text{otherwise.} \end{cases}$$

Transfer Performance Visualization

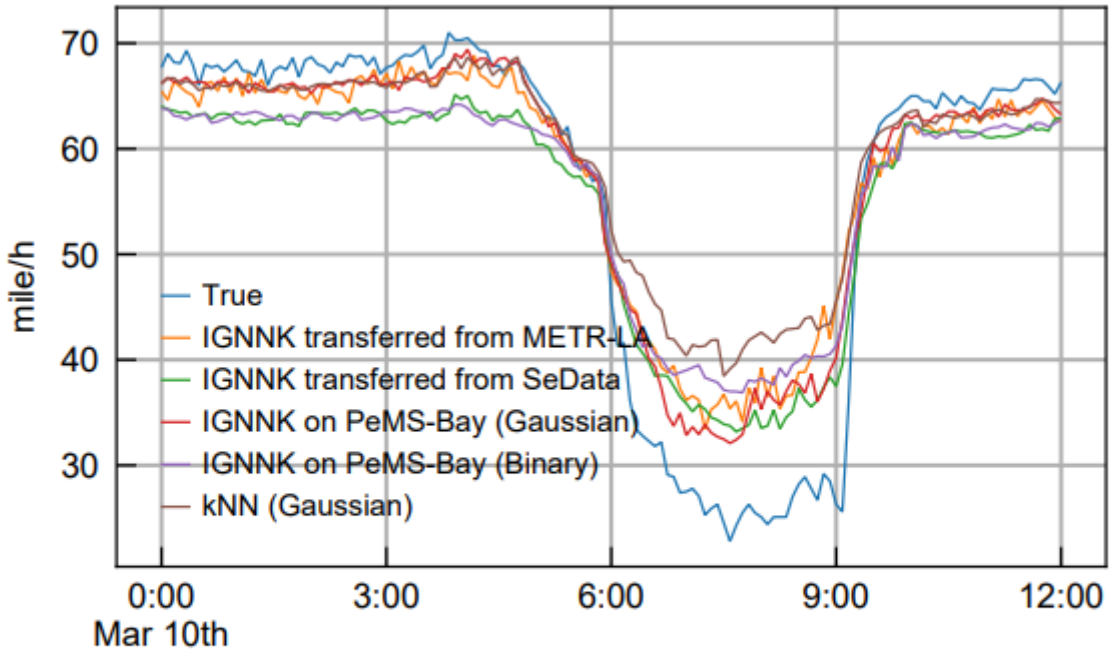
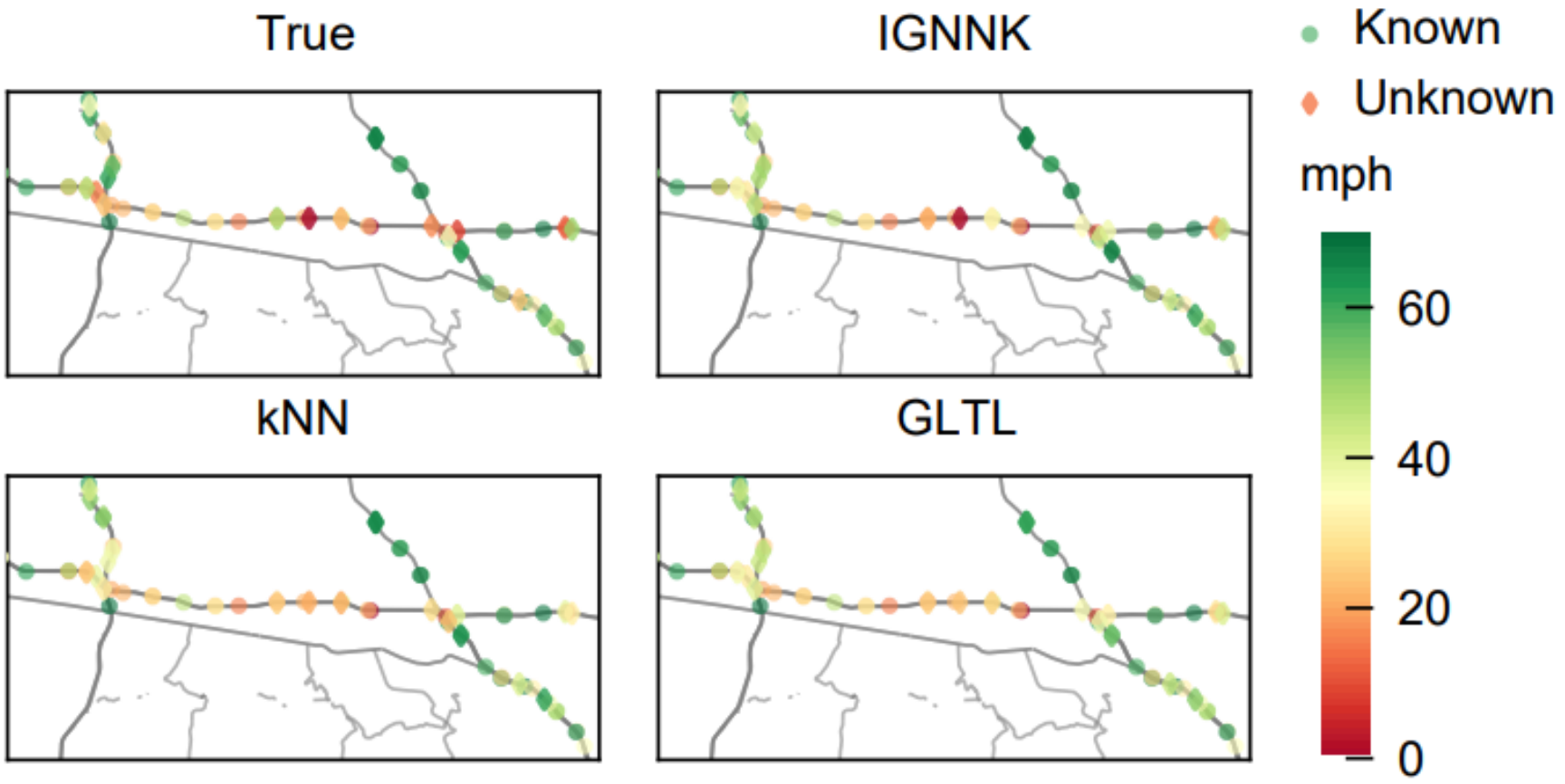


Figure 4: Kriging performance on unknown nodes in PeMS-Bay dataset.

Congestion Recognition



Only IGNNK can recognize the **congested** point.

GAT

$$e_{ij} = \alpha\left(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j\right) = \vec{a}^T\left[\mathbf{W}\vec{h}_i || \mathbf{W}\vec{h}_j\right]$$

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$$

$$\vec{h}'_i = f\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

```
def forward(self, input, adj):

      # num of nodes
    h = input
    B = h.size()[0]
    N = h.size()[1]

    a_input = torch.cat([h.repeat(1, 1, N).view(B, N * N, self.in_channels),
                         h.repeat(1, N, 1)], dim=2).view(B, N, N, 2 * self.in_channels)
    e = self.leakyrelu(torch.matmul(a_input, self.a).squeeze(3))
    zero_vec = -9e15*torch.ones_like(e)

    #>threshold for attention connection
    attention = torch.where(adj.unsqueeze(0).repeat(B, 1, 1) > self.threshold, e, zero_vec)

    attention = F.softmax(attention, dim=2)

    h_prime = torch.matmul(attention, h)

    if self.concat:
        return F.elu(h_prime)
    else:
        return h_prime
```

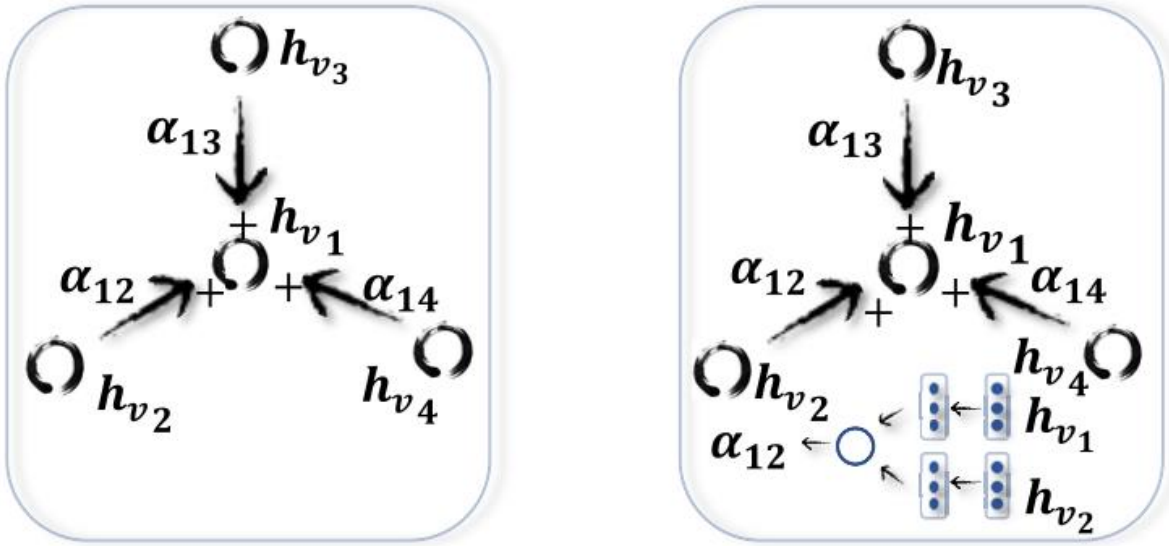


Fig. 4. Differences between GCN [22] and GAT [43]. (a) GCN [22] explicitly assigns a nonparametric weight $a_{ij} = (1/(\deg(v_i)\deg(v_j))^{1/2})$ to the neighbor v_j of v_i during the aggregation process. (b) GAT [43] implicitly captures the weight a_{ij} via an end-to-end neural network architecture so that more important nodes receive larger weights.

Reference

- **IGNNK** Inductive Graph Neural Networks for Spatiotemporal Kriging 2021 https://arxiv.org/pdf/2006.07527.pdf
- **GCN** SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS 2017 https://arxiv.org/pdf/1609.02907v4.pdf
- **GraphSAGE** Inductive Representation Learning on Large Graphs (NIPS 2017) https://arxiv.org/pdf/1706.02216.pdf
- **DCNN** Diffusion-Convolutional Neural Networks 2016 https://arxiv.org/pdf/1506.05163.pdf
- **DCRNN** DIFFUSION CONVOLUTIONAL RECURRENT NEURAL NETWORK: DATA-DRIVEN TRAFFIC FORECASTING 2018 https://arxiv.org/pdf/1707.01926.pdf
- **GAT** Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. "Graph attention networks." (NIPS 2017)
- Appleby, Gabriel et al. "Kriging Convolutional Networks." *AAAI* (2020). https://ojs.aaai.org//index.php/AAAI/article/view/5716