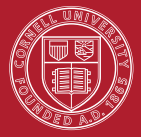


Cornell University

Lecture 22 - Graph Neural Networks Applications

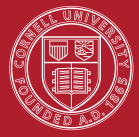
ECE 5260 – ORIE 5735

Anna Scaglione

Cornell Tech

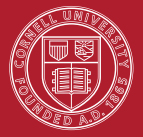
April 28, 2026

Content



Cornell University

Outline



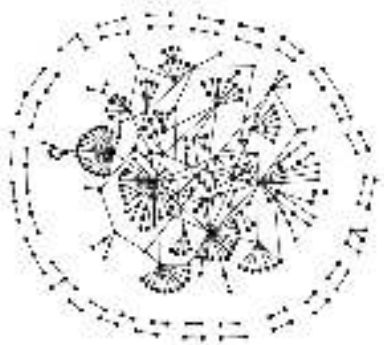
Cornell University

Application of GCN

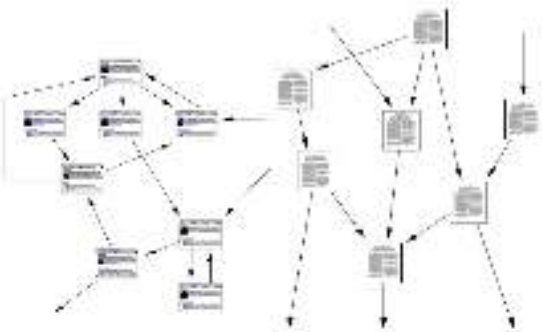
- Graph neural networks have many applications in different areas:

Social networks

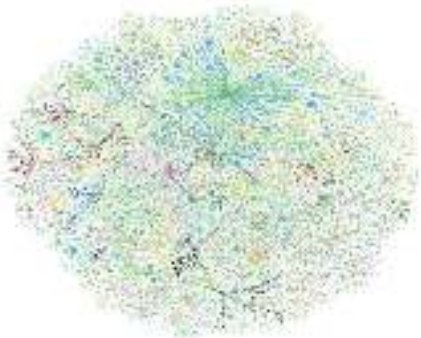
Economic networks



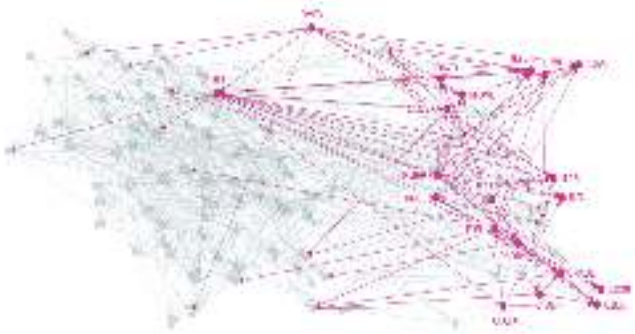
Biomedical networks



Information networks:
Web & citations



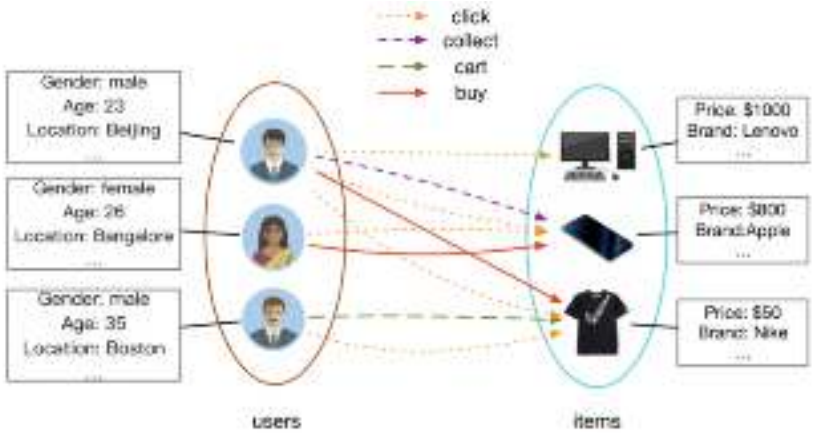
Internet



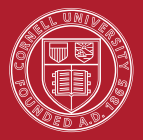
Networks of neurons

Application of GCN: Recommended system

- Naturally, graphs appear in the context of user interaction with products in an e-commerce platform.
- Therefore, many companies use GNN for product recommendation, such as Alibaba, Amazon, Pinterest.



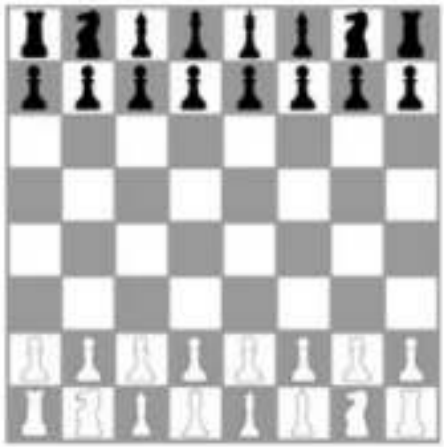
Application of GCN to combinatorics



Cornell University

- The solution of combinatorial optimization (CO) problems is the main force of many important applications in finance, logistics, energy, life sciences and hardware design.
- Google Brain team used GNN to optimize new hardware (e.g. Google's Tensor Processing Unit), i.e., the power, area and performance of the chip block.

Chess

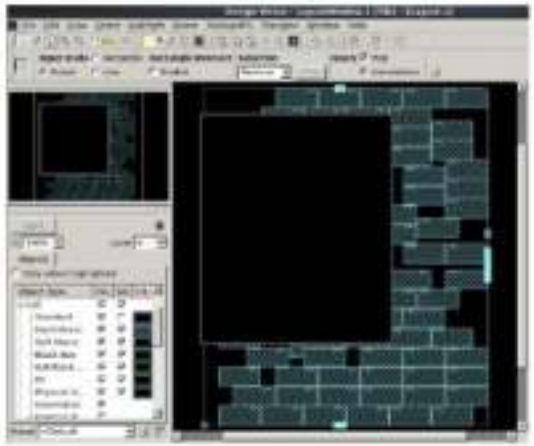


Number of states ~ 10^{123}

Go

Number of states ~ 10^{360}

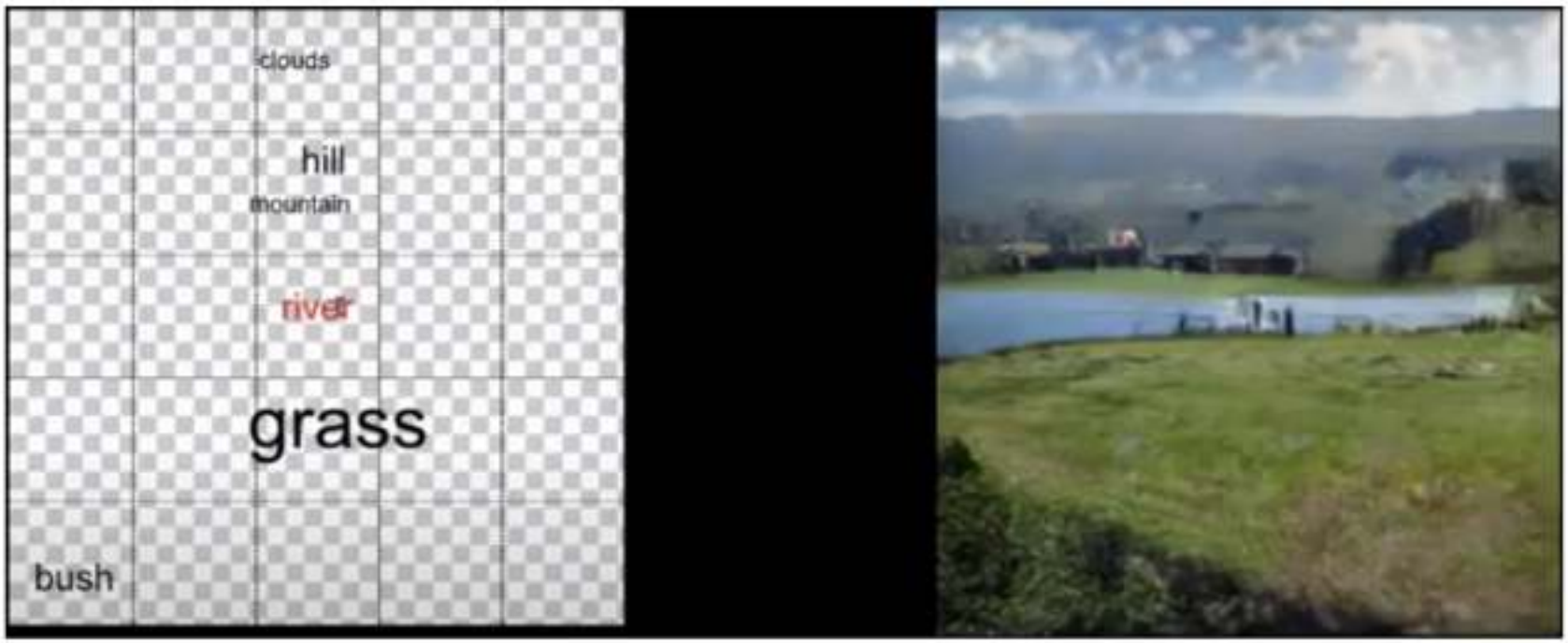
Chip Placement



Number of states ~ 10^{9000}

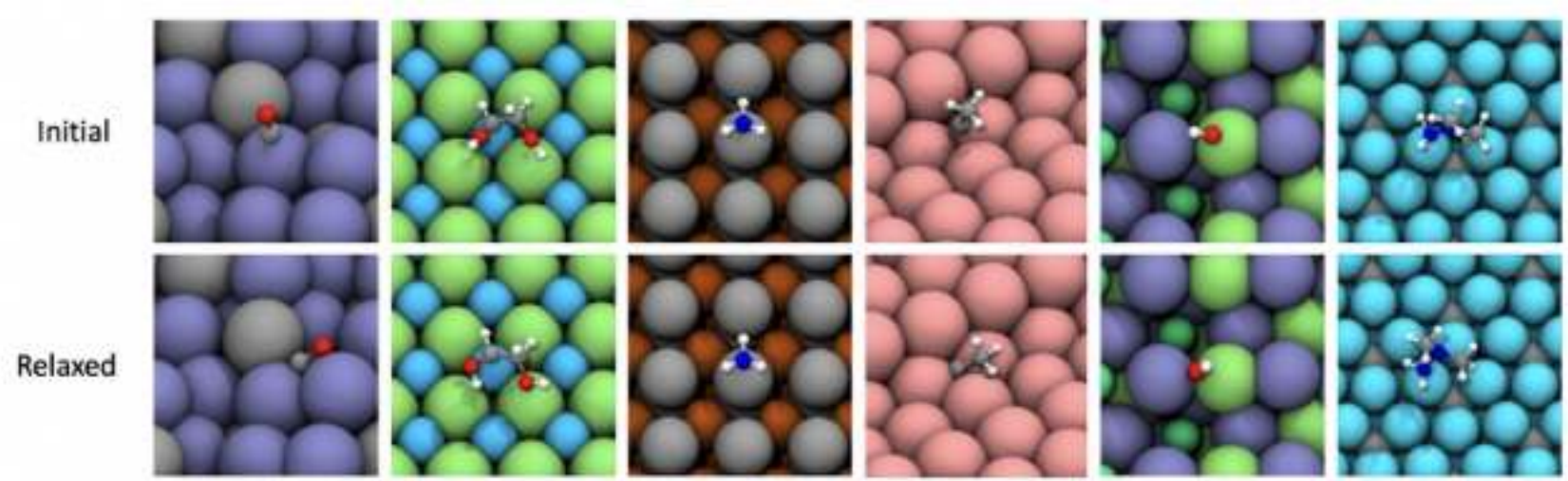
Application of GCN: Computer vision

- 3D Graphics Company Magic Leap Released a product called SuperGlue. The GNN architecture, which can perform graphics matching in real-time video, is used to complete tasks such as 3D reconstruction, location recognition, localization, and mapping (SLAM).



Application of GCN: Chemistry

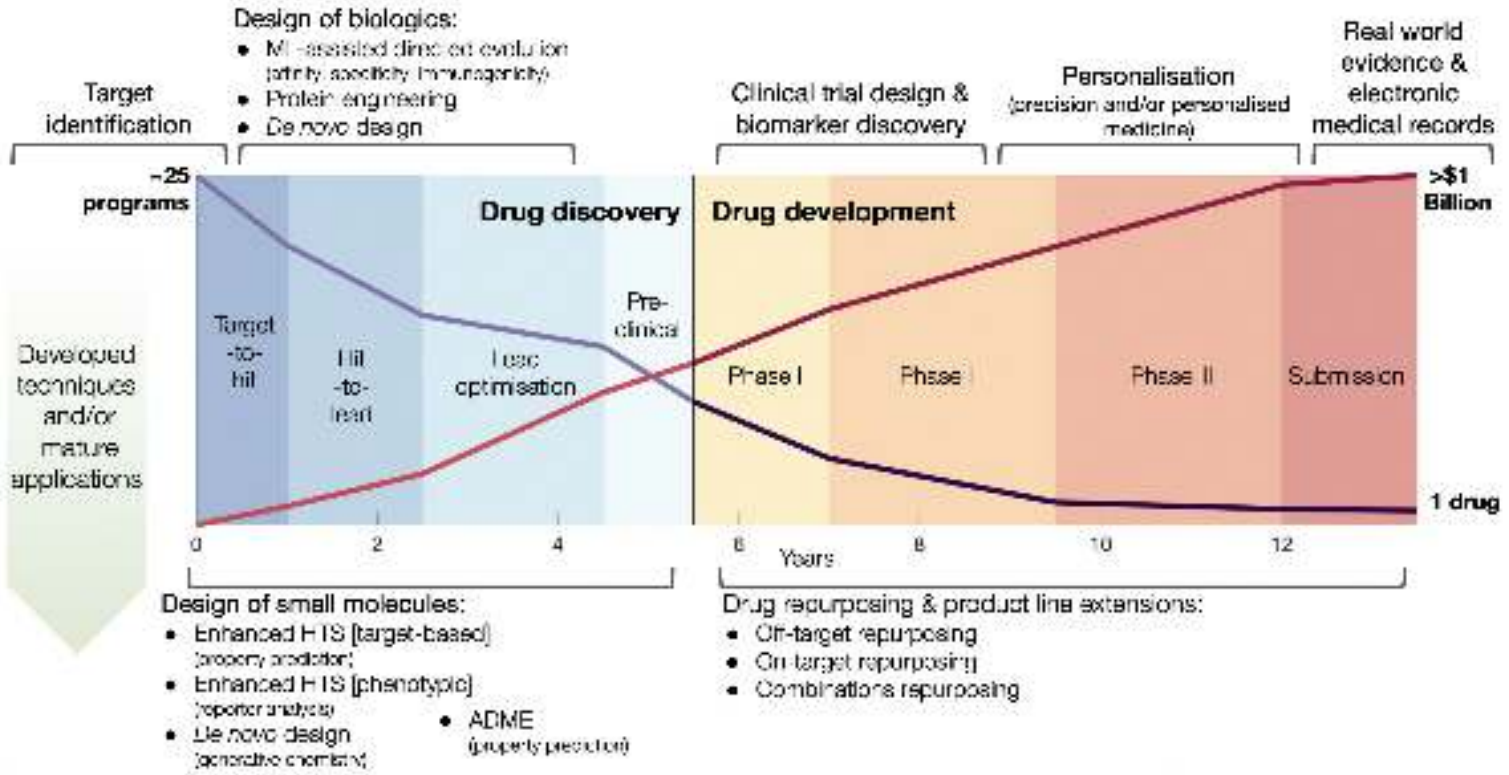
- The interactions between particles or molecules represent the graph → GNN to predict the properties of such systems, such as Facebook and CMU Open Catalyst project
- The goal is to find new low-cost molecules.



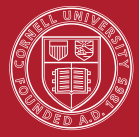
Examples of the initial state and relaxed state of adsorbates (small linking molecules) and catalyst surfaces.In order to find the relaxed state of a pair of adsorbent catalysts, expensive DFT simulations must be performed, which may take several days. Zitnick et al. 2020 year

Application of GCN: Drug discovery

- One of the most promising results of using GNN for drug discovery is that of MIT researchers namely **Chemprop**.



Outline



Cornell University

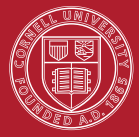
Introduction of Recommendation System

- Algorithms for recommending movies on Netflix, friends on Facebook, furniture on Amazon, and jobs on LinkedIn, etc.

Recommendation as Matrix Completion

- Recommendation as "Matrix Completion" task
  - Columns and rows represent users and items
  - matrix values represent scores determining whether a user would like an item or not (or).
- An example of Netflix challenge:
  - Carrying a 1M$ prize for the algorithm that can best predict user ratings.
  - The size of the Netflix matrix is 480k movies $\times$ 18k users (8.5B entries), with only 0.011% known entries.

Matrix completion problem



Cornell University

- Matrix completion problem: recovering the missing values of a matrix given a small fraction of its entries
  - It is common to assume that the matrix is of low rank:

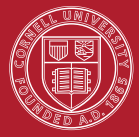
$$\min_{\mathbf{X}} \operatorname{rank}(\mathbf{X}) \quad \text{s.t.} \quad x_{ij} = y_{ij}, \forall ij \in \Omega, \tag{1}$$

  - where $\mathbf{X}$ denotes the matrix to recover, Ω is the set of the known entries and y_{ij} are their values.
  - Rank minimization is an NP-hard combinatorial problem.
- Note, $\|\mathcal{M}(\boldsymbol{X}) - \boldsymbol{Y}\| \equiv \|\boldsymbol{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|$ where $\boldsymbol{\Omega}$ is one if y_{ij} is known and zero else. The convex relaxation of (1) is to replace the rank with the nuclear norm $\|\cdot\|_\star$ equal to the sum of its singular values:

$$\min_{\mathbf{X}} \|\mathbf{X}\|_\star + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|_F^2 \tag{2}$$

$\frac{\mu}{2}$ is a constant penalty and $\|\cdot\|_F$ is Frobenius Norm.

Matrix completion problem



- Matrix completion problem: recovering the missing values of a matrix given a small fraction of its entries
  - It is common to assume that the matrix is of low rank:

$$\min_{\mathbf{X}} \operatorname{rank}(\mathbf{X}) \quad \text{s.t.} \quad x_{ij} = y_{ij}, \forall ij \in \Omega, \tag{1}$$

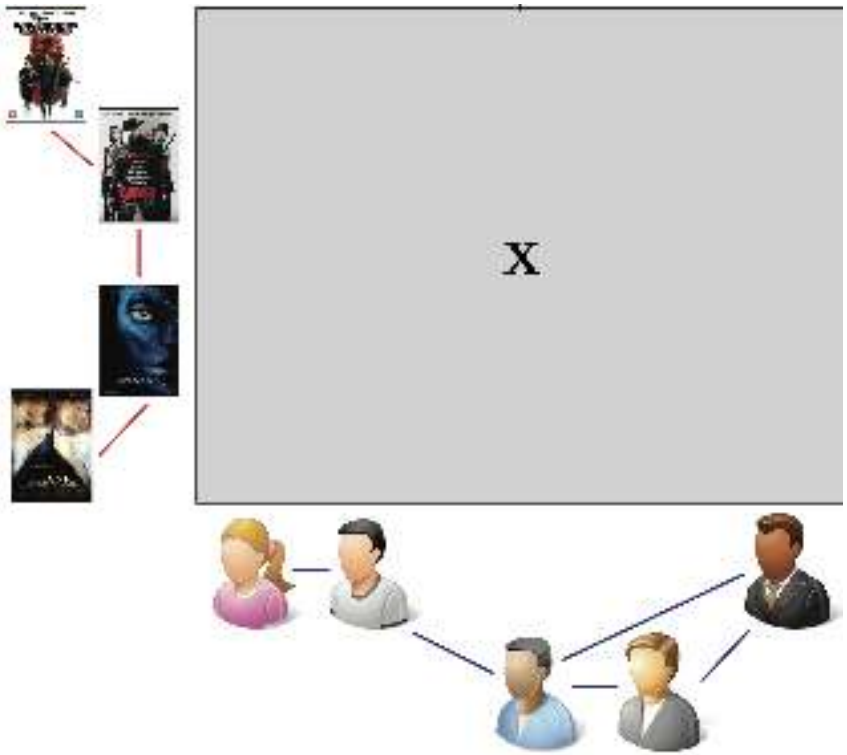
  - where $\mathbf{X}$ denotes the matrix to recover, Ω is the set of the known entries and y_{ij} are their values.
  - Rank minimization is an NP-hard combinatorial problem.
- Note, $\|\mathcal{M}(\boldsymbol{X}) - \boldsymbol{Y}\| \equiv \|\boldsymbol{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|$ where $\boldsymbol{\Omega}$ is one if y_{ij} is known and zero else. The convex relaxation of (1) is to replace the rank with the nuclear norm $\|\cdot\|_\star$ equal to the sum of its singular values:

$$\min_{\mathbf{X}} \|\mathbf{X}\|_\star + \frac{\mu}{2}\|\boldsymbol{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|_F^2 \tag{2}$$

$\frac{\mu}{2}$ is a constant penalty and $\|\cdot\|_F$ is Frobenius Norm.

Geometric matrix completion

- Assume there is a user graph and a item graph:

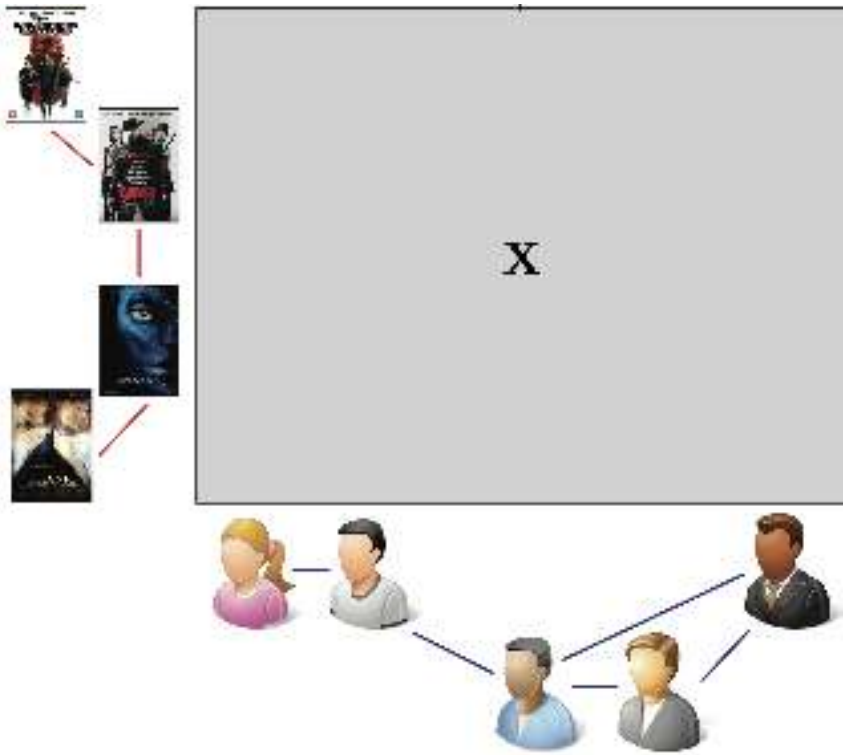


- The Laplacian Matrix (also GSO) $\mathbf{S}_r$ for the user graph, and $\mathbf{S}_c$ for the item graph
- Their smoothness can be expressed as the Dirichlet norm for the graph signals for the columns (respectively, rows) $\|\mathbf{X}\|^2_{\mathbf{S}_c} = \text{trace}\left(\mathbf{X}^\top \mathbf{S}_c \mathbf{X}\right)$ (respectively, $\|\mathbf{X}\|^2_{\mathbf{S}_r} = \text{trace}\left(\mathbf{X}^\top \mathbf{S}_r \mathbf{X}\right)$)
- The geometric matrix completion problem is to minimize:

$$\min_{\mathbf{X}} \|\mathbf{X}\|^2_{\mathbf{S}_r} + \|\mathbf{X}\|^2_{\mathbf{S}_c} + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|^2_{\mathrm{F}} \tag{3}$$

Geometric matrix completion

- Assume there is a user graph and a item graph:

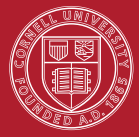


- The Laplacian Matrix (also GSO) $\mathbf{S}_r$ for the user graph, and $\mathbf{S}_c$ for the item graph
- Their smoothness can be expressed as the Dirichlet norm for the graph signals for the columns (respectively, rows) $\|\mathbf{X}\|^2_{\mathbf{S}_c} = \text{trace}\left(\mathbf{X}^\top \mathbf{S}_c \mathbf{X}\right)$ (respectively, $\|\mathbf{X}\|^2_{\mathbf{S}_r} = \text{trace}\left(\mathbf{X}^\top \mathbf{S}_r \mathbf{X}\right)$)

- The geometric matrix completion problem is to minimize:

$$\min_{\mathbf{X}} \|\mathbf{X}\|^2_{\mathbf{S}_r} + \|\mathbf{X}\|^2_{\mathbf{S}_c} + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{X} - \mathbf{Y})\|^2_{\mathrm{F}} \tag{3}$$

Factorized models

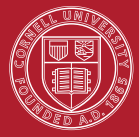


- The variables in the full $m \times n$ matrix $\mathbf{X}$: hard to scale up to large matrices such as the Netflix challenge.
- A solution is to use a factorized representation $\mathbf{X} = \mathbf{M}\mathbf{N}^\top$, where $\mathbf{M}$ and $\mathbf{N}$ are $m \times r$ and $n \times r$ matrices $r \ll \min\{m, n\}$.
- The factorized formulation of the graph-based minimization in (3) is

$$\min_{\mathbf{M},\mathbf{N}} \|\mathbf{M}\|_{\mathbf{S}_r}^2 + \|\mathbf{N}\|_{\mathbf{S}_c}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{M}\mathbf{N}^\top - \mathbf{Y})\|_{\mathrm{F}}^2 \tag{4}$$

- However, $\|\mathbf{M}\|_{\mathbf{S}_r}^2$ and $\|\mathbf{N}\|_{\mathbf{S}_r}^2$ only consider the first-order of GSO $\mathbf{S}$.
- How to use Chebyshev graph filter to capture both the user graph and item graph?

Factorized models

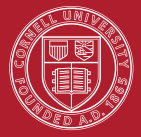


- The variables in the full $m \times n$ matrix $\mathbf{X}$: hard to scale up to large matrices such as the Netflix challenge.
- A solution is to use a factorized representation $\mathbf{X} = \mathbf{M}\mathbf{N}^\top$, where $\mathbf{M}$ and $\mathbf{N}$ are $m \times r$ and $n \times r$ matrices $r \ll \min\{m, n\}$.
- The factorized formulation of the graph-based minimization in (3) is

$$\min_{\mathbf{M},\mathbf{N}} \|\mathbf{M}\|_{\mathbf{S}_r}^2 + \|\mathbf{N}\|_{\mathbf{S}_c}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{M}\mathbf{N}^\top - \mathbf{Y})\|_{\mathrm{F}}^2 \tag{4}$$

- However, $\|\mathbf{M}\|_{\mathbf{S}_r}^2$ and $\|\mathbf{N}\|_{\mathbf{S}_r}^2$ only consider the first-order of GSO $\mathbf{S}$.
- How to use Chebyshev graph filter to capture both the user graph and item graph?

Factorized models

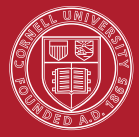


- The variables in the full $m \times n$ matrix $\mathbf{X}$: hard to scale up to large matrices such as the Netflix challenge.
- A solution is to use a factorized representation $\mathbf{X} = \mathbf{M}\mathbf{N}^\top$, where $\mathbf{M}$ and $\mathbf{N}$ are $m \times r$ and $n \times r$ matrices $r \ll \min\{m, n\}$.
- The factorized formulation of the graph-based minimization in (3) is

$$\min_{\mathbf{M},\mathbf{N}} \|\mathbf{M}\|_{\mathbf{S}_r}^2 + \|\mathbf{N}\|_{\mathbf{S}_c}^2 + \frac{\mu}{2}\|\mathbf{\Omega} \circ (\mathbf{M}\mathbf{N}^\top - \mathbf{Y})\|_{\mathrm{F}}^2 \tag{4}$$

- However, $\|\mathbf{M}\|_{\mathbf{S}_r}^2$ and $\|\mathbf{N}\|_{\mathbf{S}_r}^2$ only consider the first-order of GSO $\mathbf{S}$.
- How to use Chebyshev graph filter to capture both the user graph and item graph?

Multi-Graph GNNs



Cornell University

- Recall the signal graph filters of order K supported on $\mathbf{S}$ at layer ℓ ($\boldsymbol{x}_0 = \boldsymbol{x}$):

$$\boldsymbol{x}_\ell = \sigma\left[\boldsymbol{z}_\ell\right] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\mathbf{S}^k\boldsymbol{x}_{\ell-1}\right]$$

- When the graph signals $\mathbf{X}$ are associated with two graphs, the multi-graph version of the spectral convolution is given by

$$\tilde{\mathbf{X}} = \sigma\left[\mathbf{Z}\right] = \sigma\left[\sum_{k=0}^{K'-1}\sum_{k'=0}^{K-1} h_{kk'}\mathbf{S}_r^k\mathbf{X}\mathbf{S}_c^{k'}\right], \tag{5}$$

- Similar to the multilayer GCN with multiple channels of the previous lecture, several layers of the multi-graph version are stacked together, called Multi-Graph CNN (MGCNN) [Monti et al., 2017].

Multi-Graph GNNs

- Recall the signal graph filters of order K supported on $\mathbf{S}$ at layer ℓ ($\boldsymbol{x}_0 = \boldsymbol{x}$):

$$\boldsymbol{x}_\ell = \sigma\left[\boldsymbol{z}_\ell\right] = \sigma\left[\sum_{k=0}^{K-1} h_{\ell k}\mathbf{S}^k \boldsymbol{x}_{\ell-1}\right]$$

- When the graph signals $\mathbf{X}$ are associated with two graphs, the multi-graph version of the spectral convolution is given by

$$\tilde{\mathbf{X}} = \sigma\left[\mathbf{Z}\right] = \sigma\left[\sum_{k=0}^{K'-1}\sum_{k'=0}^{K-1} h_{kk'}\mathbf{S}_r^k \mathbf{X} \mathbf{S}_c^{k'}\right], \tag{5}$$

- Similar to the multilayer GCN with multiple channels of the previous lecture, several layers of the multi-graph version are stacked together, called Multi-Graph CNN (MGCNN) [Monti et al., 2017].

Multi-Graph GNNs for Separable convolution

- The multi-graph convolution is obtained considering the factorized form of the matrix $\mathbf{X} = \mathbf{M}\mathbf{N}^\top$ and applying one-dimensional convolutions to the respective graph to each factor.
- We define $\mathbf{m}_l, \mathbf{n}_l$ denote the l th columns of factors $\mathbf{M}, \mathbf{N}$ and $\boldsymbol{h}^r = \left(h_0^r, \ldots, h_{K-1}^r\right)$ and $\boldsymbol{h}^c = \left(h_0^c, \ldots, h_{K'-1}^c\right)$
- We can express the filters resorting to Chebyshev polynomials:

$$\tilde{\mathbf{n}} = \sigma\left[\mathbf{z}_l^r\right] = \left[\sum_{k=0}^{K-1} h_k^r \mathbf{S}_r^k \mathbf{n}_l\right], \quad \tilde{\mathbf{m}} = \sigma\left[\mathbf{z}_l^c\right] = \left[\sum_{k'=0}^{K'-1} h_{k'}^c \mathbf{S}_c^k \mathbf{m}_l\right], \quad l = 1, \ldots, r \tag{6}$$

- Then, we use the activation function $\sigma(\cdot)$ for the output signals $\mathbf{z}_l^r$ and $\mathbf{z}_l^c$. We call such an architecture a separable MGCNN or sMGCNN [Monti et al., 2017].

Multi-Graph GNNs for Separable convolution

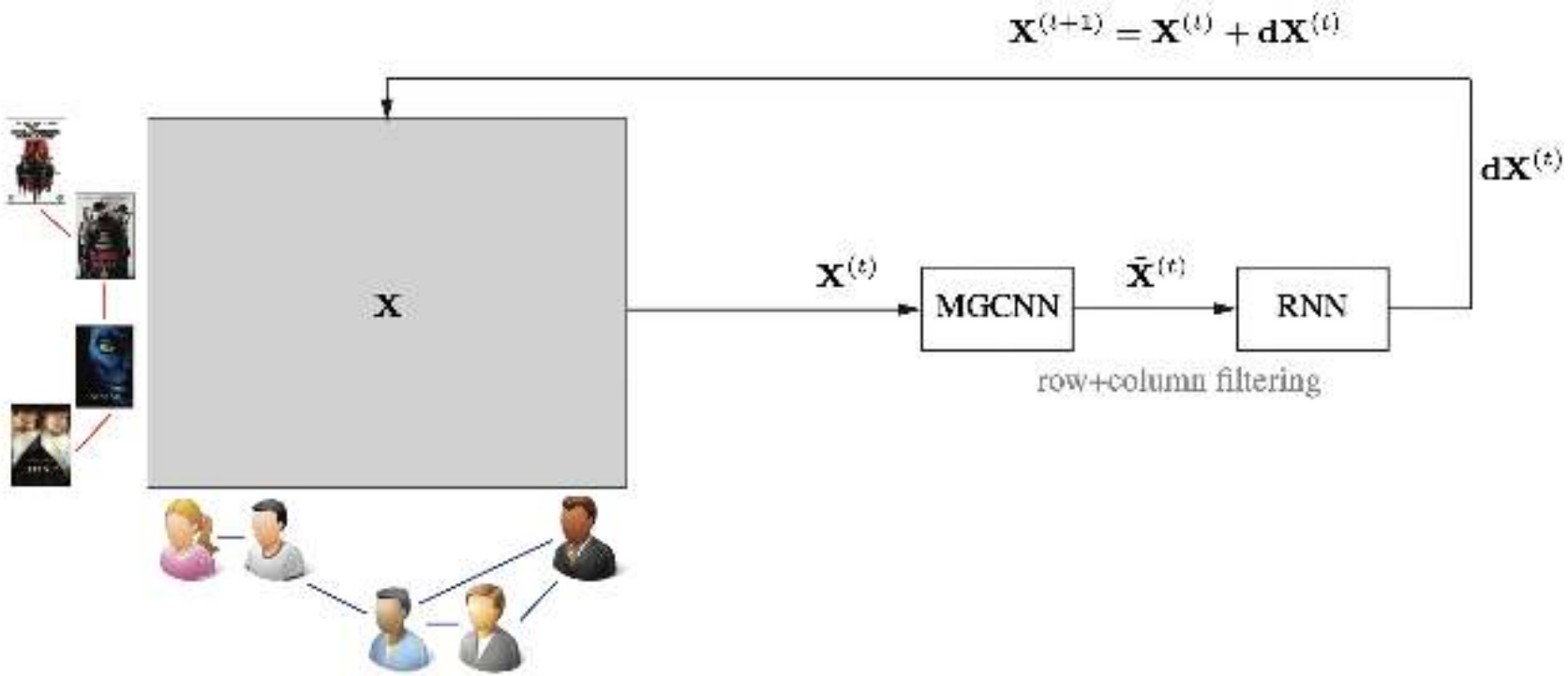
- The multi-graph convolution is obtained considering the factorized form of the matrix $\mathbf{X} = \mathbf{M}\mathbf{N}^\top$ and applying one-dimensional convolutions to the respective graph to each factor.
- We define $\mathbf{m}_l, \mathbf{n}_l$ denote the l th columns of factors $\mathbf{M}, \mathbf{N}$ and $\boldsymbol{h}^r = \left(h_0^r, \ldots, h_{K-1}^r\right)$ and $\boldsymbol{h}^c = \left(h_0^c, \ldots, h_{K'-1}^c\right)$
- We can express the filters resorting to Chebyshev polynomials:

$$\tilde{\mathbf{n}} = \sigma\left[\mathbf{z}_l^r\right] = \left[\sum_{k=0}^{K-1} h_k^r \mathbf{S}_r^k \mathbf{n}_l\right], \quad \tilde{\mathbf{m}} = \sigma\left[\mathbf{z}_l^c\right] = \left[\sum_{k'=0}^{K'-1} h_{k'}^c \mathbf{S}_c^k \mathbf{m}_l\right], \quad l = 1, \ldots, r \tag{6}$$

- Then, we use the activation function $\sigma(\cdot)$ for the output signals $\mathbf{z}_l^r$ and $\mathbf{z}_l^c$. We call such an architecture a separable MGCNN or sMGCNN [Monti et al., 2017].

Matrix diffusion with RNNs

- The next step is to feed the spatial features extracted from the matrix by the MGCNN or sMGCNN to a RNN to enable a diffusion process.
- A diffusion process that progressively reconstructs the score matrix, which is particularly suitable for extremely sparse data.
- We use the classical LSTM architecture predict accurate small changes $\mathbf{dX}$ of the matrix $\mathbf{X}$, i.e., $\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \mathbf{dX}^{(t)}$.



Algorithm Summary: MGCNN

Cornell University

- **Algorithm**:
  1. Input $m \times n$ matrix $\mathbf{X}^{(0)}$ containing initial values
  2. for $t = 0 : T$ do
     - Apply the Multi-Graph CNN (5) on $\mathbf{X}^{(t)}$ producing an $m \times n$ output $\tilde{\mathbf{X}}^{(t)}$.
     - Apply RNN to $\tilde{\mathbf{X}}^{(t)}$ producing incremental update $\mathbf{dX}^{(\mathbf{t})}$
     - Update $\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \mathbf{dX}^{(t)}$
  3. end for
- Training of the networks is performed by minimizing the loss:

$$\ell(\boldsymbol{h}, \boldsymbol{\theta}) = \left\| \mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} \right\|^2_{\mathcal{G}_r} + \left\| \mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} \right\|^2_{\mathcal{G}_c} + \frac{\mu}{2} \left\| \boldsymbol{\Omega} \circ \left(\mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} - \mathbf{Y} \right) \right\|^2_{\mathrm{F}} \qquad (7)$$

  here, T denotes the number of diffusion iterations (applications of the RNN). So, we only minimize the last $\mathbf{X}^{(t)}_{\boldsymbol{h},\boldsymbol{\theta}}$, i.e., $t = T$.
- $\mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}}$ depends on the parameters of the MGCNN (Chebyshev polynomial coefficients $\boldsymbol{h}$ and those of the LSTM (denoted by $\boldsymbol{\theta}$)

Algorithm Summary: MGCNN

Cornell University

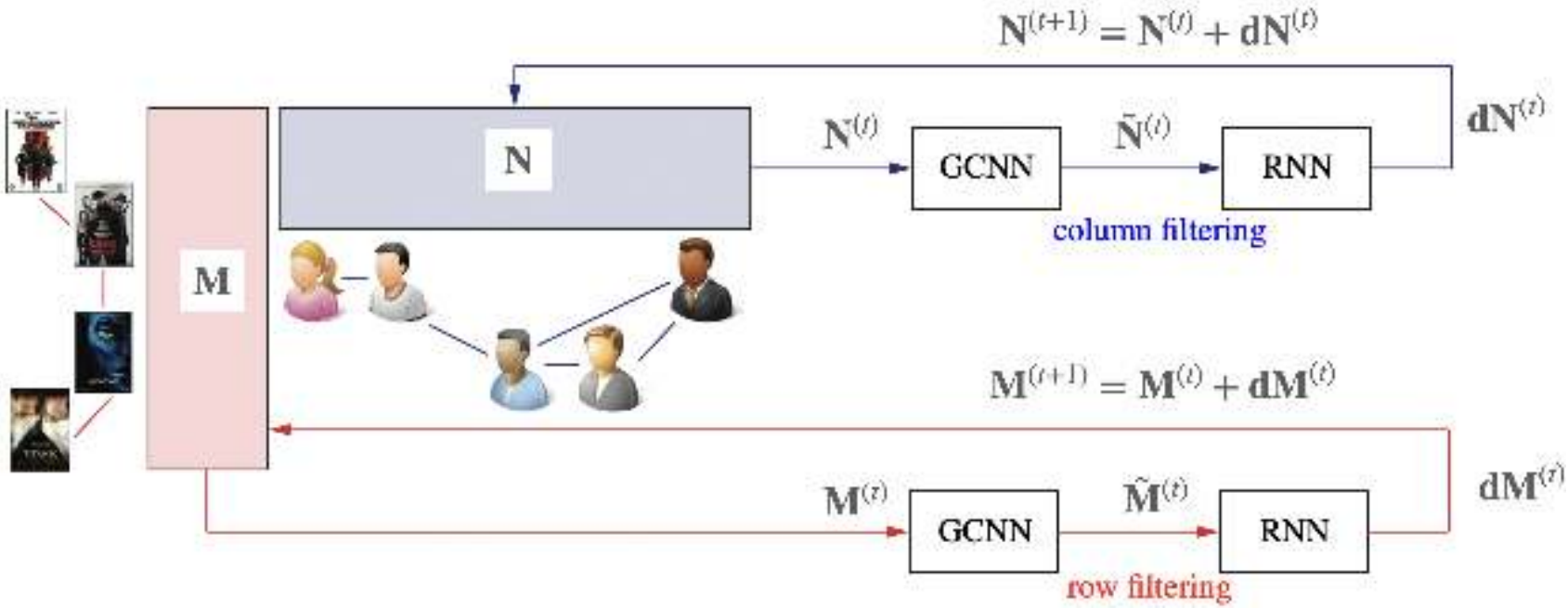
- **Algorithm**:
  1. Input $m \times n$ matrix $\mathbf{X}^{(0)}$ containing initial values
  2. for $t = 0 : T$ do
     - Apply the Multi-Graph CNN (5) on $\mathbf{X}^{(t)}$ producing an $m \times n$ output $\tilde{\mathbf{X}}^{(t)}$.
     - Apply RNN to $\tilde{\mathbf{X}}^{(t)}$ producing incremental update $\mathbf{dX}^{(\mathbf{t})}$
     - Update $\mathbf{X}^{(t+1)} = \mathbf{X}^{(t)} + \mathbf{dX}^{(t)}$
  3. end for
- Training of the networks is performed by minimizing the loss:

$$\ell(\boldsymbol{h}, \boldsymbol{\theta}) = \left\| \mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} \right\|^2_{\mathcal{G}_r} + \left\| \mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} \right\|^2_{\mathcal{G}_c} + \frac{\mu}{2} \left\| \boldsymbol{\Omega} \circ \left(\mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}} - \mathbf{Y} \right) \right\|^2_{\mathrm{F}} \qquad (7)$$

  here, T denotes the number of diffusion iterations (applications of the RNN). So, we only minimize the last $\mathbf{X}^{(t)}_{\boldsymbol{h},\boldsymbol{\theta}}$, i.e., $t = T$.
- $\mathbf{X}^{(T)}_{\boldsymbol{h},\boldsymbol{\theta}}$ depends on the parameters of the MGCNN (Chebyshev polynomial coefficients $\boldsymbol{h}$ and those of the LSTM (denoted by $\boldsymbol{\theta}$)

Algorithm Summary: MGCNN

- Separable Recurrent MGCNN (sRMGCNN) architecture using the factorized matrix completion model and operating separately on the rows and columns of the factors



Algorithm Summary: MGCNN

Cornell University

- **Algorithm**:
  1. Input $m \times r$ factor $\mathbf{M}^{(0)}$ and $n \times r$ factor $\mathbf{N}^{(0)}$ representing the matrix $\mathbf{X}^{(0)}$
  2. for $t = 0 : T$ do
     - Apply the Graph CNN on $\mathbf{M}^{(t)}$ producing an output $\tilde{\mathbf{M}}^{(t)}$.
     - Apply RNN to $\tilde{\mathbf{M}}^{(t)}$ producing incremental update $\mathbf{dM}$
     - Update $\mathbf{M}^{(t+1)} = \mathbf{M}^{(t)} + \mathbf{dM}^{(t)}$
  3. Repeat the above steps for $\mathbf{N}^{(t)}$
  4. end for
- In the factorized setting, we use the loss

$$\ell(\boldsymbol{h}_r, \boldsymbol{h}_c, \boldsymbol{\theta}) = \left\|\mathbf{M}^{(T)}_{\boldsymbol{h}_r,\boldsymbol{\theta}}\right\|^2_{\mathcal{G}_r} + \left\|\mathbf{N}^{(T)}_{\boldsymbol{h}_c,\boldsymbol{\theta}}\right\|^2_{\mathcal{G}_c} + \frac{\mu}{2}\left\|\boldsymbol{\Omega} \circ \left(\mathbf{M}^{(T)}_{\boldsymbol{h}_r,\boldsymbol{\theta}}(\mathbf{N}^{(T)}_{\boldsymbol{h}_c,\boldsymbol{\theta}})^\top - \mathbf{Y}\right)\right\|^2_{\mathrm{F}} \tag{8}$$

where $\boldsymbol{h}_r, \boldsymbol{h}_c$ are the parameters of the two GCNNs.

Algorithm Summary: MGCNN

- **Algorithm**:
  1. Input $m \times r$ factor $\mathbf{M}^{(0)}$ and $n \times r$ factor $\mathbf{N}^{(0)}$ representing the matrix $\mathbf{X}^{(0)}$
  2. for $t = 0 : T$ do
     - Apply the Graph CNN on $\mathbf{M}^{(t)}$ producing an output $\tilde{\mathbf{M}}^{(t)}$.
     - Apply RNN to $\tilde{\mathbf{M}}^{(t)}$ producing incremental update $\mathbf{dM}$
     - Update $\mathbf{M}^{(t+1)} = \mathbf{M}^{(t)} + \mathbf{dM}^{(t)}$
  3. Repeat the above steps for $\mathbf{N}^{(t)}$
  4. end for
- In the factorized setting, we use the loss

$$\ell(\boldsymbol{h}_r, \boldsymbol{h}_c, \boldsymbol{\theta}) = \left\| \mathbf{M}^{(T)}_{\boldsymbol{h}_r, \boldsymbol{\theta}} \right\|^2_{\mathcal{G}_r} + \left\| \mathbf{N}^{(T)}_{\boldsymbol{h}_c, \boldsymbol{\theta}} \right\|^2_{\mathcal{G}_c} + \frac{\mu}{2} \left\| \boldsymbol{\Omega} \circ \left(\mathbf{M}^{(T)}_{\boldsymbol{h}_r, \boldsymbol{\theta}} (\mathbf{N}^{(T)}_{\boldsymbol{h}_c, \boldsymbol{\theta}})^\top - \mathbf{Y} \right) \right\|^2_{\mathrm{F}} \tag{8}$$

where $\boldsymbol{h}_r, \boldsymbol{h}_c$ are the parameters of the two GCNNs.

Simulation Results

- Datasets: MovieLens, Flixster, Douban and YahooMusic datasets
- The user and item (movie) graphs are unweighted **10-nearest neighbor graphs** in the space of user and movie features

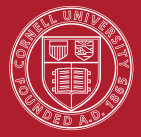
Table 4: Performance (RMS error) of different matrix completion methods on the MovieLens dataset.

METHOD	RMSE
GLOBAL MEAN	1.154
USER MEAN	1.063
MOVIE MEAN	1.033
MC [9]	0.973
IMC [17, 42]	1.653
GMC [19]	0.996
GRALS [33]	0.945
sRMGCNN	**0.929**

Table 5: Performance (RMS error) on several datasets. For Douban and YahooMusic, a single graph (of users and items respectively) was used. For Flixster, two settings are shown: users+items graphs / only users graph.

METHOD	FLIXSTER	DOUBAN	YAHOOMUSIC
GRALS	1.3126 / 1.2447	0.8326	38.0423
sRMGCNN	**1.1788 / 0.9258**	**0.8012**	**22.4149**

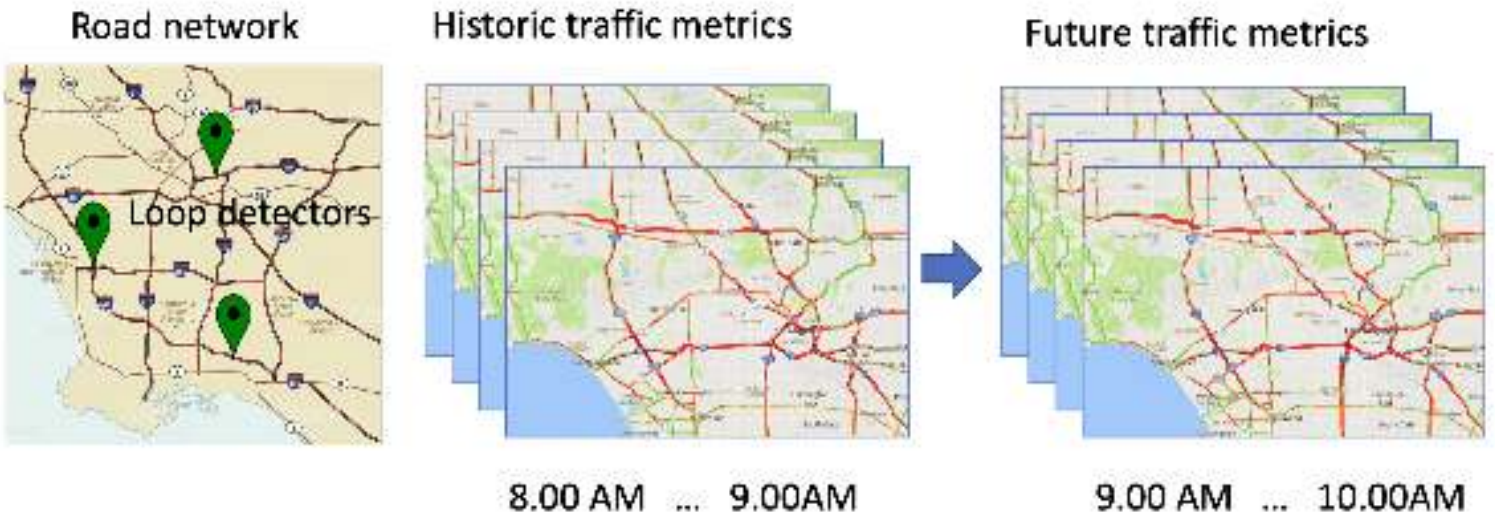
Outline



Cornell University

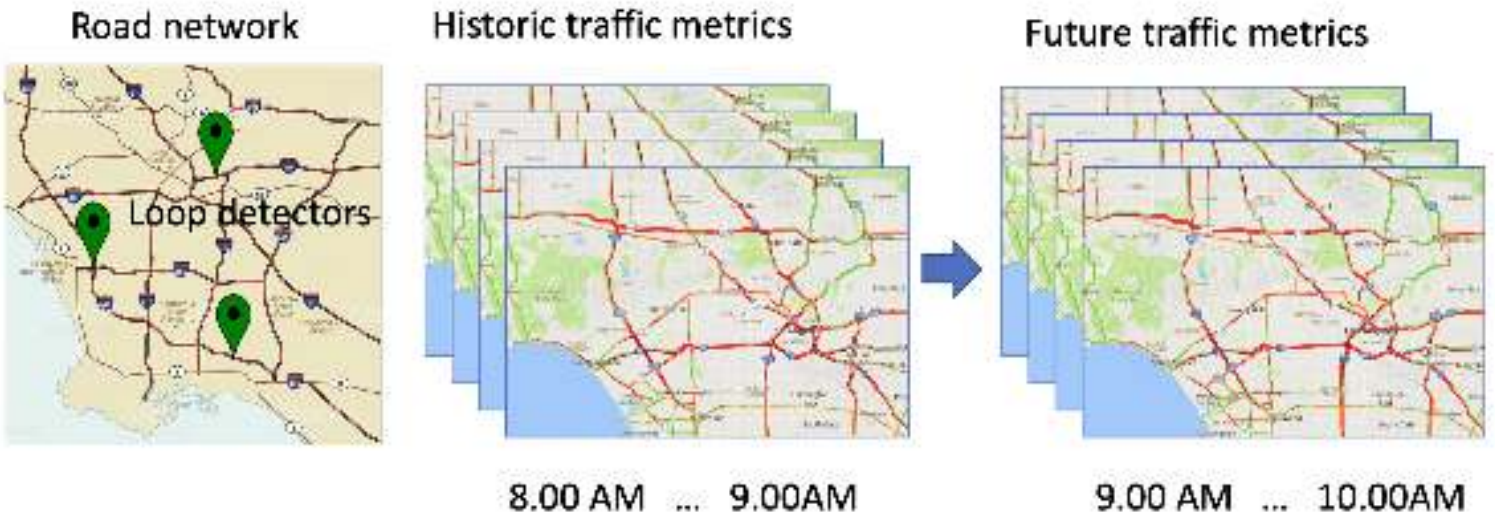
Traffic forecasting

- Why we need to investigate traffic forecasting:
  - Provide a scientific basis for traffic managers to sense traffic congestion and limit vehicles in advance
  - Provide security for urban travelers to choose appropriate travel routes and improve travel efficiency
- Traffic forecasting has always been a challenge task due to:
  - **Spatial dependence**
  - **Temporal dependence**



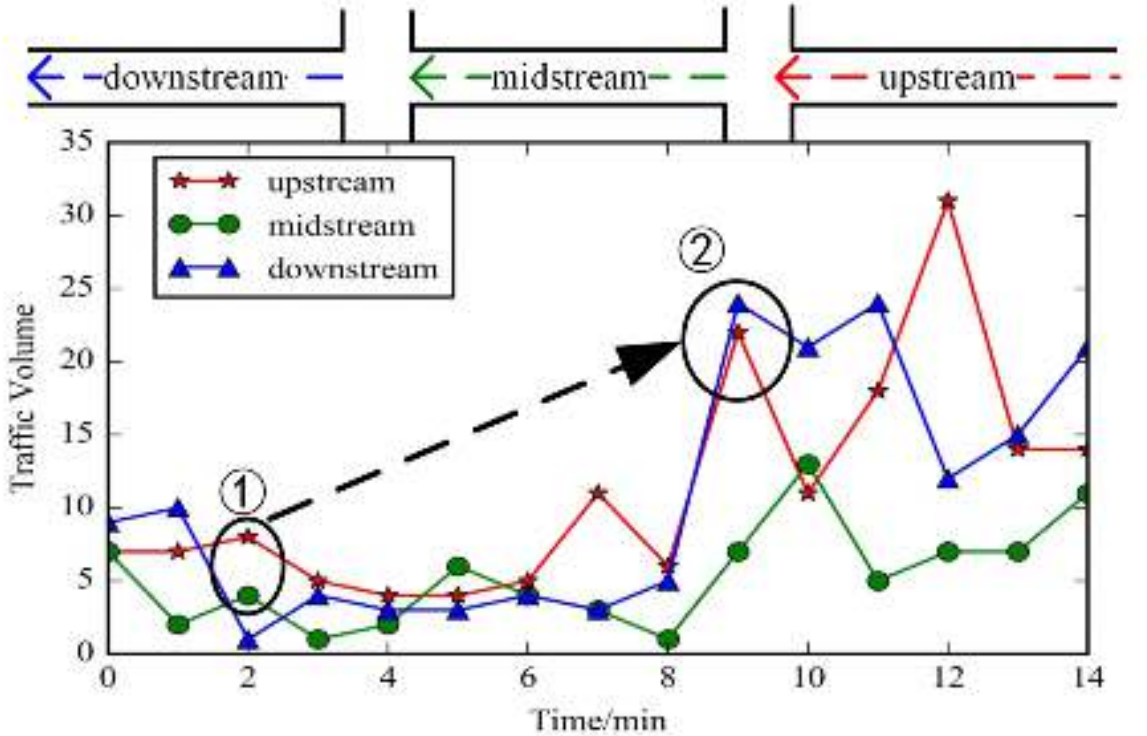
Traffic forecasting

- Why we need to investigate traffic forecasting:
  - Provide a scientific basis for traffic managers to sense traffic congestion and limit vehicles in advance
  - Provide security for urban travelers to choose appropriate travel routes and improve travel efficiency
- Traffic forecasting has always been a challenge task due to:
  - **Spatial dependence**
  - **Temporal dependence**



Spatial dependence

- The change in traffic volume is dominated by the topological structure of the urban road network:
  - the traffic status at upstream roads impact traffic status at downstream roads through the transfer effect
  - the traffic status at downstream roads impact traffic status at upstream through the feedback effect



Temporal dependence

- The traffic volume changes dynamically over time and is mainly reflected in periodicity and trend.

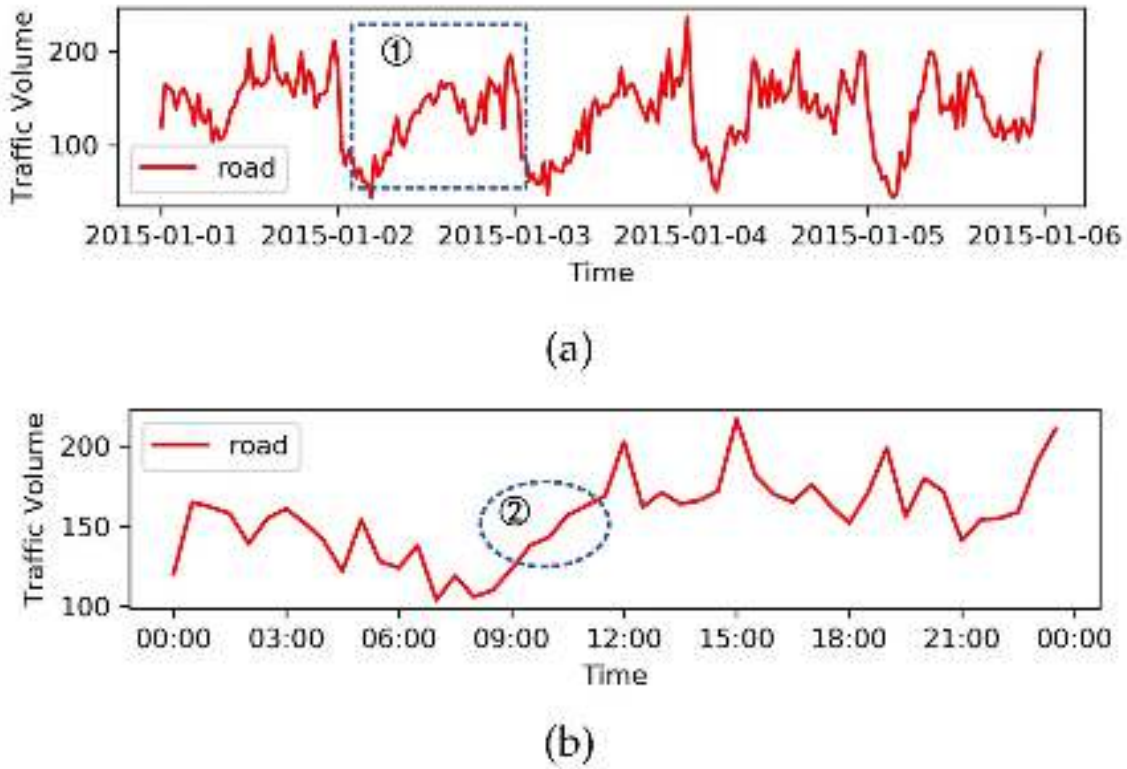


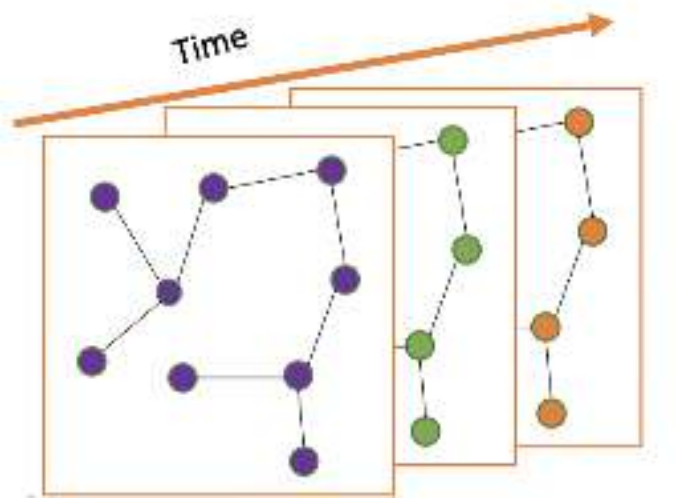
Fig. 2. (a) Periodicity. The traffic volume in the road changes periodically within one week. (b) Trend. The traffic volume in the road has tendency change within one day.

Problem Formulation

- Traffic forecast is a typical time-series prediction problem, i.e. predicting the most likely traffic measurements (e.g. traffic flow) in the next H time steps given the previous M traffic observations.

$$\boldsymbol{x}_{t+1}^*, \ldots, \boldsymbol{x}_{t+H}^* = \underset{\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+H}}{\arg\max} \log P\left(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+H} \mid \boldsymbol{x}_{t-M+1}, \ldots, \boldsymbol{x}_t\right) \tag{9}$$

where $\boldsymbol{x}_t \in \mathcal{R}^N$ is the graph signals at time step t.



- We model the traffic network as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\boldsymbol{x}_t$ is the graph signals over $\mathcal{G}$.
- The node $v \in \mathcal{V}$ represents monitor stations in a traffic network;
- $\mathcal{E}$ is a set of edges, indicating the connectedness between stations (e.g. distance);

1st Order Approximation of RF For GCN

- [Kipf and Welling, 2016] adopts a first-order filter

$$\boldsymbol{z} = h_0\boldsymbol{x} + h_1(\tilde{\mathbf{S}})\boldsymbol{x} = h_0\boldsymbol{x} + h_1(2\mathbf{S}/\lambda_{\max} - \boldsymbol{I}_N)\boldsymbol{x}$$

- [Kipf and Welling, 2016] has two approximations:
  1. The largest eigenvalue $\lambda_{\max} \approx 2$
  2. h_0 and h_1 are replaced by a single parameter h: $h = h_0 = -h_1$.
- Recall that Normalized Laplacian: : $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$, where $A_{uv} = w_{uv}$ if $(u, v) \in \mathcal{E}$ and 0 else.

$$\mathbf{S} = \boldsymbol{I}_N - \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}}$$

- We have

$$\boldsymbol{z} = h(2 * \boldsymbol{I}_N - \mathbf{S})\boldsymbol{x} = h(\boldsymbol{I}_N + \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}})\boldsymbol{x}$$

1st Order Approximation of RF For GCN

- [Kipf and Welling, 2016] adopts a first-order filter

$$\boldsymbol{z} = h_0\boldsymbol{x} + h_1(\tilde{\mathbf{S}})\boldsymbol{x} = h_0\boldsymbol{x} + h_1(2\mathbf{S}/\lambda_{\max} - \boldsymbol{I}_N)\boldsymbol{x}$$

- [Kipf and Welling, 2016] has two approximations:
  1. The largest eigenvalue $\lambda_{\max} \approx 2$
  2. h_0 and h_1 are replaced by a single parameter h: $h = h_0 = -h_1$.
- Recall that Normalized Laplacian: : $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$, where $A_{uv} = w_{uv}$ if $(u, v) \in \mathcal{E}$ and 0 else.

$$\mathbf{S} = \boldsymbol{I}_N - \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}}$$

- We have

$$\boldsymbol{z} = h(2 * \boldsymbol{I}_N - \mathbf{S})\boldsymbol{x} = h(\boldsymbol{I}_N + \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}})\boldsymbol{x}$$

1st Order Approximation of RF For GCN

- [Kipf and Welling, 2016] adopts a first-order filter

$$\boldsymbol{z} = h_0\boldsymbol{x} + h_1(\tilde{\mathbf{S}})\boldsymbol{x} = h_0\boldsymbol{x} + h_1(2\mathbf{S}/\lambda_{\max} - \boldsymbol{I}_N)\boldsymbol{x}$$

- [Kipf and Welling, 2016] has two approximations:
  1. The largest eigenvalue $\lambda_{\max} \approx 2$
  2. h_0 and h_1 are replaced by a single parameter h: $h = h_0 = -h_1$.
- Recall that Normalized Laplacian: : $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$, where $A_{uv} = w_{uv}$ if $(u, v) \in \mathcal{E}$ and 0 else.

$$\mathbf{S} = \boldsymbol{I}_N - \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}}$$

- We have

$$\boldsymbol{z} = h(2 * \boldsymbol{I}_N - \mathbf{S})\boldsymbol{x} = h(\boldsymbol{I}_N + \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}})\boldsymbol{x}$$

1st Order Approximation of RF For GCN

- [Kipf and Welling, 2016] adopts a first-order filter

$$\boldsymbol{z} = h_0\boldsymbol{x} + h_1(\tilde{\mathbf{S}})\boldsymbol{x} = h_0\boldsymbol{x} + h_1(2\mathbf{S}/\lambda_{\max} - \boldsymbol{I}_N)\boldsymbol{x}$$

- [Kipf and Welling, 2016] has two approximations:
  1. The largest eigenvalue $\lambda_{\max} \approx 2$
  2. h_0 and h_1 are replaced by a single parameter h: $h = h_0 = -h_1$.
- Recall that Normalized Laplacian: : $\boldsymbol{d} = \boldsymbol{A}\mathbf{1}$, where $A_{uv} = w_{uv}$ if $(u, v) \in \mathcal{E}$ and 0 else.

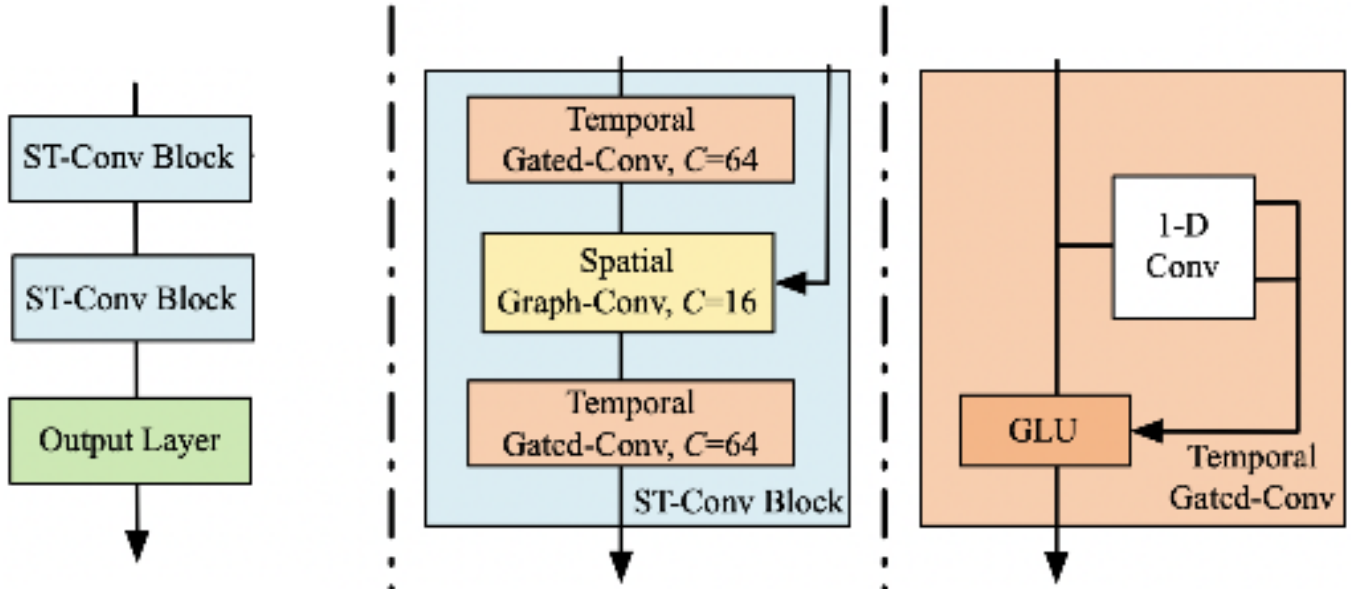
$$\mathbf{S} = \boldsymbol{I}_N - \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}}$$

- We have

$$\boldsymbol{z} = h(2 * \boldsymbol{I}_N - \mathbf{S})\boldsymbol{x} = h(\boldsymbol{I}_N + \text{diag}(\boldsymbol{d})^{-\frac{1}{2}}\boldsymbol{A}\text{diag}(\boldsymbol{d})^{-\frac{1}{2}})\boldsymbol{x}$$

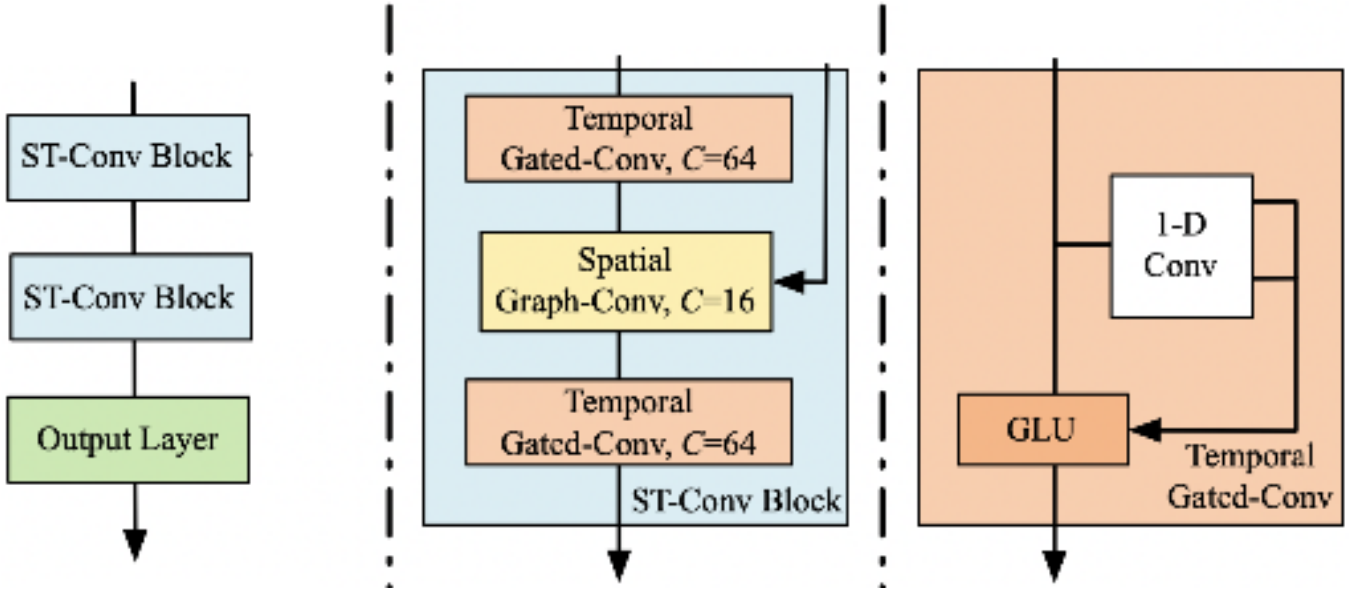
The STGCN algorithm for Traffic Forecasting

- [Yu et al., 2018] proposed the following STGCN algorithm for traffic forecasting:



- The framework STGCN consists of two spatio-temporal convolutional blocks (ST-Conv blocks) and a fully-connected output layer in the end.
- Each ST-Conv block contains two temporal gated convolution layers and one spatial graph convolution layer in the middle (C represents the output channels).

The STGCN algorithm for Traffic Forecasting



- The spatial graph convolution layer has been introduced as
  1. Either the Chebyshev Polynomials or the 1st-order approximation of Chebyshev Polynomials shown:

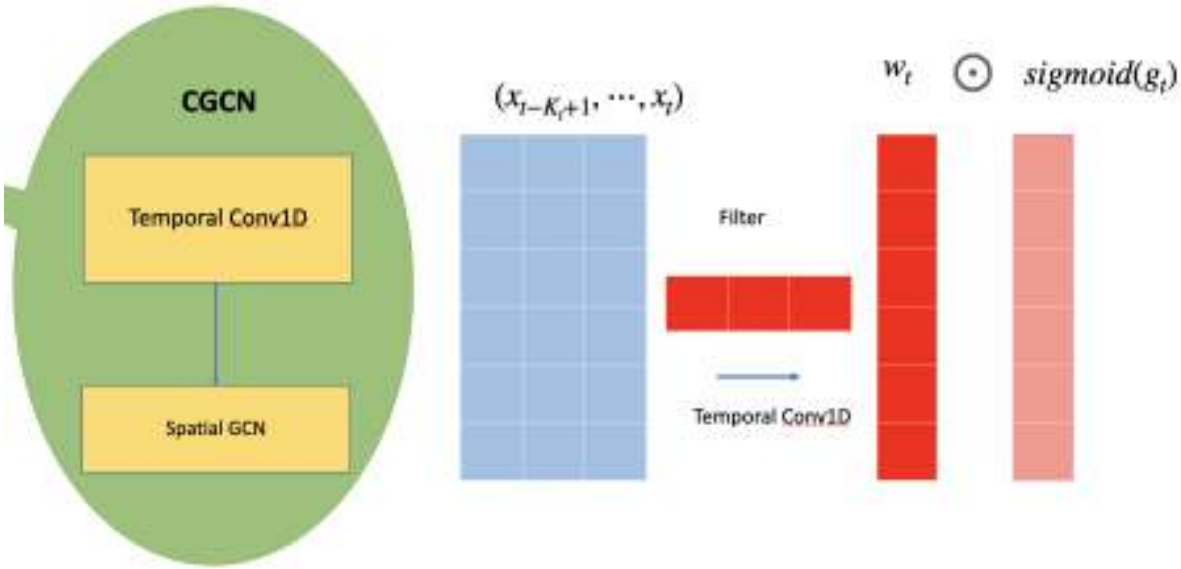
$$\boldsymbol{z} = h(\boldsymbol{I}_N + \text{diag}(\boldsymbol{d})^{-\frac{1}{2}} \boldsymbol{A} \text{diag}(\boldsymbol{d})^{-\frac{1}{2}})\boldsymbol{x}$$

The STGCN algorithm for Traffic Forecasting

- How to design the **temporal gated convolutional neural networks**?
- Recall that the graph signal z_t of STGCN in Lecture 25 as follows:

$$z_t = \mathrm{ReLU}\left[\sum_{k=0}^{K-1}\sum_{\tau=0}^{K_t-1} h_{k,\tau}\mathbf{S}^k x_{t-\tau}\right],$$

- We have one channel for $\boldsymbol{w}_t = \left(\sum_{\tau=0}^{K_t} h_{k,\tau}\boldsymbol{x}_{t-\tau}\right)$ and another channel for gating $\boldsymbol{g}_t = \left(\sum_{\tau=0}^{K_t} h'_{k,\tau}\boldsymbol{x}_{t-\tau}\right)$, i.e., $\boldsymbol{z}^t = \sum_{k=0}^{K}\Theta_k\mathbf{S}^k(\boldsymbol{w}_t \odot \mathrm{sigmoid}(\boldsymbol{g}_t))$.

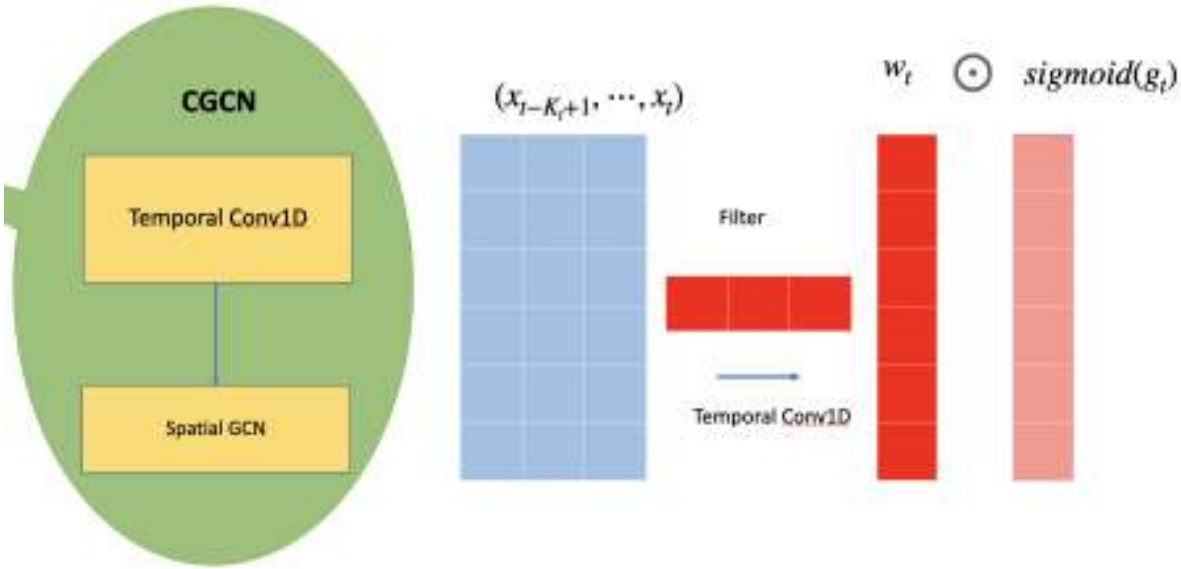


The STGCN algorithm for Traffic Forecasting

- How to design the **temporal gated convolutional neural networks**?
- Recall that the graph signal $\boldsymbol{z}_t$ of STGCN in Lecture 25 as follows:

$$\boldsymbol{z}_t = \text{ReLU}\left[\sum_{k=0}^{K-1}\sum_{\tau=0}^{K_t-1} h_{k,\tau}\mathbf{S}^k\boldsymbol{x}_{t-\tau}\right],$$

- We have one channel for $\boldsymbol{w}_t = \left(\sum_{\tau=0}^{K_t} h_{k,\tau}\boldsymbol{x}_{t-\tau}\right)$ and another channel for gating $\boldsymbol{g}_t = \left(\sum_{\tau=0}^{K_t} h'_{k,\tau}\boldsymbol{x}_{t-\tau}\right)$, i.e., $\boldsymbol{z}^t = \sum_{k=0}^{K}\Theta_k\mathbf{S}^k(\boldsymbol{w}_t \odot \text{sigmoid}(\boldsymbol{g}_t))$.



Objective and Graph Construction

Cornell University

- The loss function of STGCN for traffic prediction can be written as,

$$\min_{\mathcal{H}} \sum_{t} \ell(\Phi(\boldsymbol{x}_t, \cdots \boldsymbol{x}_{t-M+1}; \mathbf{S}, \mathcal{H}), \boldsymbol{x}_{t+H}) =$$

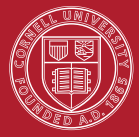
$$\min_{\mathcal{H}} \sum_{t} \|\Phi(\boldsymbol{x}_t, \cdots \boldsymbol{x}_{t-M+1}; \mathbf{S}, \mathcal{H}) - \boldsymbol{x}_{t+H}\|^2 \tag{10}$$

- The sensor graphs in the urban road network are constructed by

$$w_{uv} = \begin{cases} \exp\left(-\frac{dist_{uv}^2}{\sigma^2}\right), & u \neq v \text{ and } \exp\left(-\frac{dist_{uv}^2}{\sigma^2}\right) \geq \epsilon \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

where w_{uv} is the weight of edge which is decided by $dist_{uv}$ (the distance between station u and v). σ^2 and ϵ are thresholds to control the distribution and sparsity of the weighted adjacent matrix $\boldsymbol{A}$, assigned to 10 and 0.5, respectively.

Objective and Graph Construction



- The loss function of STGCN for traffic prediction can be written as,

$$\min_{\mathcal{H}} \sum_{t} \ell(\Phi(\boldsymbol{x}_t, \cdots \boldsymbol{x}_{t-M+1}; \mathbf{S}, \mathcal{H}), \boldsymbol{x}_{t+H}) =$$

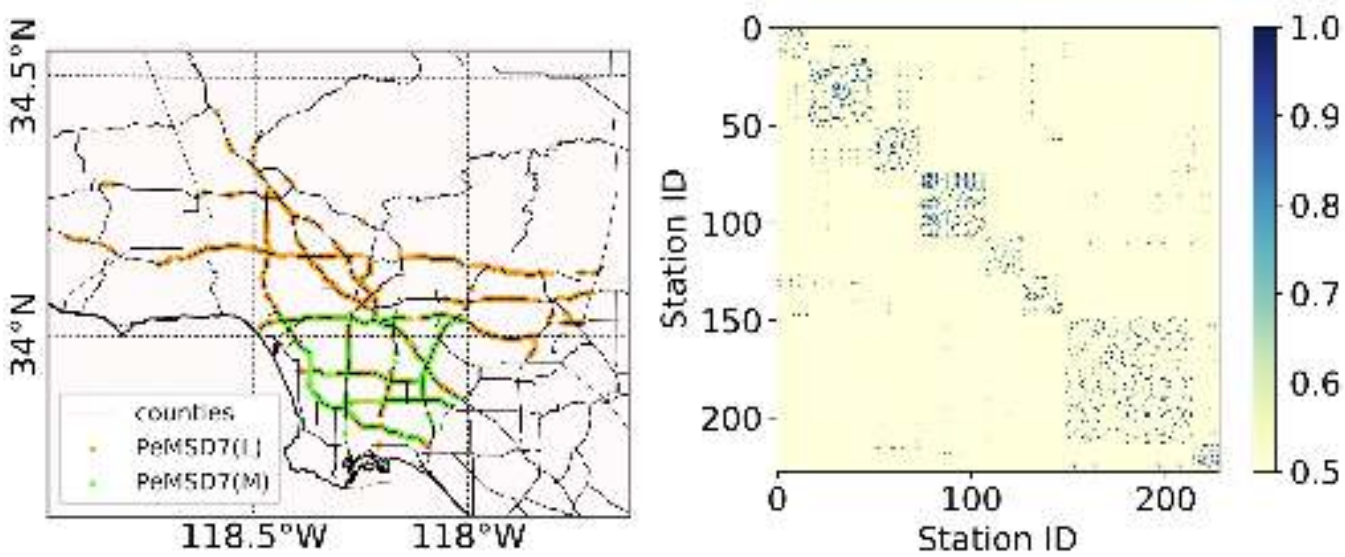
$$\min_{\mathcal{H}} \sum_{t} \|\Phi(\boldsymbol{x}_t, \cdots \boldsymbol{x}_{t-M+1}; \mathbf{S}, \mathcal{H}) - \boldsymbol{x}_{t+H}\|^2 \tag{10}$$

- The sensor graphs in the urban road network are constructed by

$$w_{uv} = \begin{cases} \exp\left(-\frac{dist_{uv}^2}{\sigma^2}\right), & u \neq v \text{ and } \exp\left(-\frac{dist_{uv}^2}{\sigma^2}\right) \geq \epsilon \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

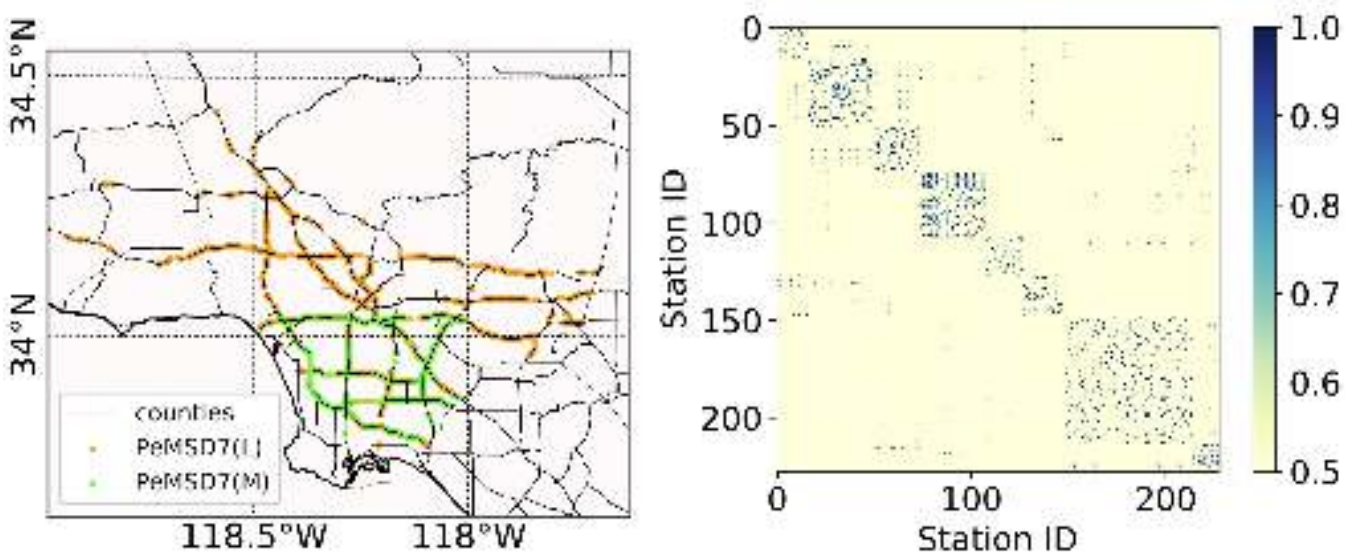
where w_{uv} is the weight of edge which is decided by $dist_{uv}$ (the distance between station u and v). σ^2 and ϵ are thresholds to control the distribution and sparsity of the weighted adjacent matrix $\boldsymbol{A}$, assigned to 10 and 0.5, respectively.

Testing Urban



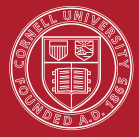
- PeMS sensor network in District 7 of California (left), each dot denotes a sensor station
- Heat map of weighted adjacency matrix in PeMSD7(M) (right).

Testing Urban



- PeMS sensor network in District 7 of California (left), each dot denotes a sensor station
- Heat map of weighted adjacency matrix in PeMSD7(M) (right).

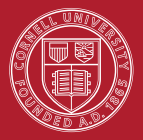
Testing Urban



Model	PeMSD7(M) (15/ 30/ 45 min)		
	MAE	MAPE (%)	RMSE
HA	4.01	10.61	7.20
LSVR	2.50/ 3.63/ 4.54	5.81/ 8.88/ 11.50	4.55/ 6.67/ 8.28
ARIMA	5.55/ 5.86/ 6.27	12.92/ 13.94/ 15.20	9.00/ 9.13/ 9.38
FNN	2.74/ 4.02/ 5.04	6.38/ 9.72/ 12.38	4.75/ 6.98/ 8.58
FC-LSTM	3.57/ 3.94/ 4.16	8.60/ 9.55/ 10.10	6.20/ 7.03/ 7.51
GCGRU	2.37/ 3.31/ 4.01	5.54/ 8.06/ 9.99	4.21/ 5.96/ 7.13
STGCN(Cheb)	**2.25/ 3.03/ 3.57**	5.26/ **7.33/ 8.69**	**4.04/ 5.70/ 6.77**
STGCN(1^{st})	2.26/ 3.09/ 3.79	**5.24**/ 7.39/ 9.12	4.07/ 5.77/ 7.03

- The sensors sample traffic volumes very 5 mins. Therefore, we predict traffic volumes in the future of 15 mins, 30 mins, and 45 mins.
- STGCN, including STGCN(Cheb) and STGCN(1^{st}), have smaller MAE (Mean absolute error), MAPE (Mean absolute percentage error) and RMSE (Root-mean-square error) compared to other algorithms.

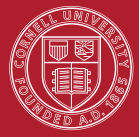
Testing Urban



Model	PeMSD7(M) (15/ 30/ 45 min)		
	MAE	MAPE (%)	RMSE
HA	4.01	10.61	7.20
LSVR	2.50/ 3.63/ 4.54	5.81/ 8.88/ 11.50	4.55/ 6.67/ 8.28
ARIMA	5.55/ 5.86/ 6.27	12.92/ 13.94/ 15.20	9.00/ 9.13/ 9.38
FNN	2.74/ 4.02/ 5.04	6.38/ 9.72/ 12.38	4.75/ 6.98/ 8.58
FC-LSTM	3.57/ 3.94/ 4.16	8.60/ 9.55/ 10.10	6.20/ 7.03/ 7.51
GCGRU	2.37/ 3.31/ 4.01	5.54/ 8.06/ 9.99	4.21/ 5.96/ 7.13
STGCN(Cheb)	**2.25/ 3.03/ 3.57**	5.26/ **7.33/ 8.69**	**4.04/ 5.70/ 6.77**
STGCN(1^{st})	2.26/ 3.09/ 3.79	**5.24**/ 7.39/ 9.12	4.07/ 5.77/ 7.03

- The sensors sample traffic volumes very 5 mins. Therefore, we predict traffic volumes in the future of 15 mins, 30 mins, and 45 mins.
- STGCN, including STGCN(Cheb) and STGCN(1^{st}), have smaller MAE (Mean absolute error), MAPE (Mean absolute percentage error) and RMSE (Root-mean-square error) compared to other algorithms.

Outline



Cornell University

Benchmarks and databases per application

- **Recommendation / matrix completion**: MovieLens-100K/1M/10M, Netflix Prize, Amazon-Reviews, Yelp, Pinterest (Pin-Sage). Douban, Flixster, YahooMusic are the small testbeds used by MGCNN and sMGCNN. Ciao and Epinions add a social graph.
- **Traffic forecasting**: METR-LA (207 loop sensors, Los Angeles), PeMS-BAY (325 sensors, Bay Area), PeMSD7(M/L) – the one in today's STGCN results. NYC-Taxi and TaxiBJ for OD flows.
- **Molecules and chemistry**: QM9 (134K small molecules), ZINC, MoleculeNet (Tox21, ESOL, Lipophilicity, FreeSolv, BBBP), PCQM4Mv2 (OGB-LSC, 3.7M molecules), Open Catalyst 2020 / OC22 for catalysis.
- **Citation / academic graphs**: Cora, CiteSeer, Pubmed (Planetoid splits); OGBN-Arxiv (170K papers), OGBN-Products (2.4M Amazon items), OGBN-Papers100M.
- **Knowledge graphs**: FB15k-237, WN18RR, NELL-995; OGBL-Biokg, Hetionet (biomedical).
- **Umbrella hubs**: OGB (Open Graph Benchmark) for standardized splits and leaderboards; TUDataset for graph-level classification; PyG and DGL ship most of these as one-liners.

How to actually train a GCN

Cornell University

- **Pick S.** For Kipf–Welling: $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}(\mathbf{A}+\mathbf{I})\tilde{\mathbf{D}}^{-1/2}$. For Chebyshev: rescaled Laplacian $\tilde{\mathbf{L}} = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}$. Always add self-loops; always normalize.
- **Architecture knobs.**
  - filter order K: 2–3 is plenty for most node tasks;
  - depth L: start with 2–4 – deeper GCNs over-smooth;
  - width F: 64–256 hidden features;
  - dropout on features *and* on edges (DropEdge), weight decay 10^{-4}.
- **Loss.** Cross-entropy for node/graph classification; MSE for regression (e.g. STGCN's $\|\Phi(\cdot) - \boldsymbol{x}_{t+H}\|^2$); BPR or binary cross-entropy with *negative sampling* for link prediction and recommendation.
- **Optimizer.** Adam or AdamW, learning rate 10^{-3} to $5 \cdot 10^{-3}$, cosine decay or plateau scheduling, early stopping on validation.
- **Batching for big graphs.** Full-batch works up to $\sim 10^5$ nodes. Above that: *neighbor sampling* (GraphSAGE), *Cluster-GCN*, or subgraph sampling. Never recompute $\mathbf{S}$ from scratch per batch.

Scaling I: GraphSAGE

- The layer $\mathbf{X}_\ell = \sigma[\sum_k \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}]$ touches every node and every edge at each step: memory and compute scale with $|\mathcal{E}|$, so it becomes infeasible beyond $\sim 10^5$ nodes.
- **Idea** (Hamilton, Ying, Leskovec, NeurIPS 2017). At each layer and for each target node v in the batch, sample a fixed-size subset $\mathcal{N}_s(v) \subset \mathcal{N}(v)$, $|\mathcal{N}_s(v)| = S_\ell$.
- Per-node update:

$$\boldsymbol{x}_v^{(\ell)} = \sigma\Big[\, \mathbf{W}^{(\ell)} \,\big[\, \boldsymbol{x}_v^{(\ell-1)},\ \mathrm{AGG}\big(\{\boldsymbol{x}_u^{(\ell-1)} : u \in \mathcal{N}_s(v)\}\big)\,\big]\,\Big],$$

  with $\mathrm{AGG} =$ mean, max-pool, or LSTM over the sampled neighbours.
- With fan-outs (S_1, S_2) and depth $L = 2$, each target expands into $1 + S_1 + S_1 S_2$ nodes – **independent of graph size**.
- **Inductive by construction**: the model is a function of local structure, so it generalizes to *unseen* nodes at inference (new Pinterest pins, new Twitter users, new papers).
- Trade-off: sampling variance (need enough fan-out) vs. memory (fan-out grows exponentially with L). Typical: $(S_1, S_2) = (25, 10)$.

Scaling II: Cluster-GCN

- Alternative to neighbour sampling: *partition the graph first*, then batch on clusters (Chiang *et al.*, KDD 2019).
- Use METIS (or similar) to split $\mathcal{V}$ into P roughly balanced clusters $\{\mathcal{V}_1, \ldots, \mathcal{V}_P\}$ that minimize edge cut. Write $\mathbf{S}$ in block form:

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{11} & \cdots & \mathbf{S}_{1P} \\ \vdots & \ddots & \vdots \\ \mathbf{S}_{P1} & \cdots & \mathbf{S}_{PP} \end{bmatrix}, \qquad \mathbf{S}_{pq} \triangleq \mathbf{S}[\mathcal{V}_p, \mathcal{V}_q].$$

- One Cluster-GCN mini-batch: pick a cluster p (or the union of q random clusters), run the full GCN on the induced subgraph using only $\mathbf{S}_{pp}$, and backprop on loss restricted to nodes of $\mathcal{V}_p$:

$$\mathbf{X}_{\ell}^{(p)} = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}_{pp}^{k}\, \mathbf{X}_{\ell-1}^{(p)}\, \mathbf{H}_{\ell k}\right].$$

- **No exponential fan-out**. Memory per batch: $\mathcal{O}(|\mathcal{V}_p|\, F\, L)$. Enables OGBN-Products ($\sim$2.4M nodes) on a single GPU.
- Cost: inter-cluster edges $\mathbf{S}_{pq}$ $(p \neq q)$ are dropped per batch. METIS keeps the cut small; the *stochastic* variant (union of q random clusters per batch) mixes clusters and reduces this bias.

Scaling III: graph sampling in general

- Every scalable GCN trainer is some stochastic approximation of the full-batch update; the choice is *where* you sample.

Family	**What is sampled**	**Examples**
per-node	neighbours of each target, per layer	GraphSAGE, PinSage
per-layer	a shared node set per layer, across targets	FastGCN, LADIES
per-subgraph	a subgraph, then full MP on it	Cluster-GCN, GraphSAINT

- **GraphSAINT** (Zeng *et al.*, ICLR 2020) samples a subgraph $\mathcal{G}_s$ (node-, edge-, or random-walk-based) and *reweights* aggregations to remain unbiased:

$$\hat{\boldsymbol{z}}_v = \sum_{u \in \mathcal{N}(v) \cap \mathcal{G}_s} \frac{1}{p_{uv}} [\mathbf{S}]_{vu} \, \boldsymbol{x}_u, \qquad p_{uv} = \Pr[(u, v) \in \mathcal{G}_s].$$

- **Rules of thumb**
  - sparse graphs (roads, small molecules): *full-batch* is fine;
  - dense with hub nodes (social, e-commerce): *Cluster-GCN* or *GraphSAINT* – neighbour sampling suffers from hub blow-up;
  - deployment with new nodes at inference: *GraphSAGE* (inductive).
- All three samplers are one-liners in PyG (`NeighborLoader`, `ClusterLoader`, `GraphSAINTRandomWalkSampler`).

Evaluation metrics by task

- **Node classification** (Cora, OGBN-Arxiv): accuracy, macro-F1. Use the *public* split if one exists – random splits inflate numbers.
- **Graph classification/regression** (TUDataset, QM9, MoleculeNet): accuracy, ROC-AUC, or MAE. QM9 targets are usually reported per-property MAE in meV.
- **Link prediction / KG completion**: ROC-AUC, Hits@k, Mean Reciprocal Rank

$$\text{MRR} = \frac{1}{|\mathcal{T}_{\text{test}}|} \sum_{(h,r,t)\in\mathcal{T}_{\text{test}}} \frac{1}{\text{rank}(t|h,r)}.$$

  Use *filtered* ranks (exclude other known true tails) and degree-matched negatives – uniform negatives are too easy.
- **Recommendation** (MovieLens, Amazon): Recall@k, NDCG@k, MAP. Split by user-time (leave-one-out or last interaction), not random, to avoid future leakage.
- **Traffic** (as in today's STGCN slide): MAE, MAPE, RMSE, usually at 15/30/45 min horizons. Report all three horizons – models that win short often lose long.

Common pitfalls (and how to avoid them)

- Over-smoothing. Stacking many GCN layers drives embeddings to the leading eigenspace of $\mathbf{S}$. Fixes: residual connections $\mathbf{X}_\ell \leftarrow \mathbf{X}_\ell + \mathbf{X}_{\ell-1}$, PairNorm/GraphNorm, DropEdge, or just stay shallow ($L \leq 4$).
- Data leakage in relational splits. When you delete edges for link prediction, delete them *also* from the message-passing graph $\mathbf{S}$ – otherwise the model sees the answer.
- Temporal leakage. In STGCN or any forecasting setting, never train on time indices that overlap with test. Build $\mathbf{S}$ from *training-window* statistics only.
- Normalization choice matters. Raw adjacency, row-normalization $\mathbf{D}^{-1}\mathbf{A}$, and symmetric $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ give different spectral properties and can change accuracy by several points. Ablate.
- Cold-start and missing patterns. MC methods that condition on row/column graphs still need a minimal number of observed entries per row/column – don't evaluate on users with zero training interactions.
- Imbalanced classes. Node classification with rare classes (fraud, anomaly) needs class-weighted loss or focal loss; plain accuracy is misleading.

Tooling and a minimal code template

Cornell University

- **Libraries.** PyTorch Geometric (PyG) – easiest entry; DGL – scales to very large graphs; TF-GNN for TensorFlow users; Spektral for Keras.
- **Benchmark hub.** Open Graph Benchmark (OGB) gives canonical splits, metrics, and leaderboards for node/link/graph tasks; use it for reproducible numbers.

PyG: a 2-layer GCN and one training step

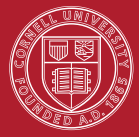
```
import torch, torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GCN(torch.nn.Module):
    def __init__(self, d_in, hid, d_out):
        super().__init__()
        self.conv1 = GCNConv(d_in, hid)
        self.conv2 = GCNConv(hid, d_out)
    def forward(self, x, edge_index):     # edge_index encodes S
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, p=0.5, training=self.training)
        return self.conv2(x, edge_index)

model = GCN(d_in, 64, n_cls)
opt   = torch.optim.Adam(model.parameters(), lr=1e-2, weight_decay=5e-4)
for epoch in range(200):
    out  = model(data.x, data.edge_index)
    loss = F.cross_entropy(out[data.train_mask], data.y[data.train_mask])
    opt.zero_grad(); loss.backward(); opt.step()
```

- Swap `GCNConv`→`ChebConv` for Chebyshev, `RGCNConv` for relational, `SAGEConv` for sampled neighbourhoods.

Outline



Cornell University

Conclusions

- Across these applications, the architecture has stayed the same: a stack of layers

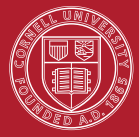
$$\mathbf{X}_{\ell} = \sigma\left[\sum_{k=0}^{K-1} \mathbf{S}^{k}\mathbf{X}_{\ell-1}\mathbf{H}_{\ell k}\right],$$

  with the graph shift operator $\mathbf{S}$ as prior information and the filter tensor $\mathcal{H}$ as the only trainable parameter.

- What changed from one application to the next was how we built $\mathbf{S}$ (row/column graph for MC, sensor graph for STGCN) and how we wired the filters in time (STGCN, GRNN).

- **Next lecture.** We will relax the assumption that $\mathbf{S}$ is fixed: we let the architecture learn a data-dependent graph shift via *attention*. This is the Graph Transformer, and it is the bridge between GSP/GNNs and the foundation-model era.

- After that, Module 3 closes with Graphical Models.



References

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016).
Semi-supervised classification with graph convolutional networks.
arXiv preprint arXiv:1609.02907.

[Monti et al., 2017] Monti, F., Bronstein, M., and Bresson, X. (2017).
Geometric matrix completion with recurrent multi-graph neural networks.
Advances in neural information processing systems, 30.

[Yu et al., 2018] Yu, B., Yin, H., and Zhu, Z. (2018).
Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting.
In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640.