Deep Learning with Graph-Structured Data: A Mathematical Perspective

Jie Chen

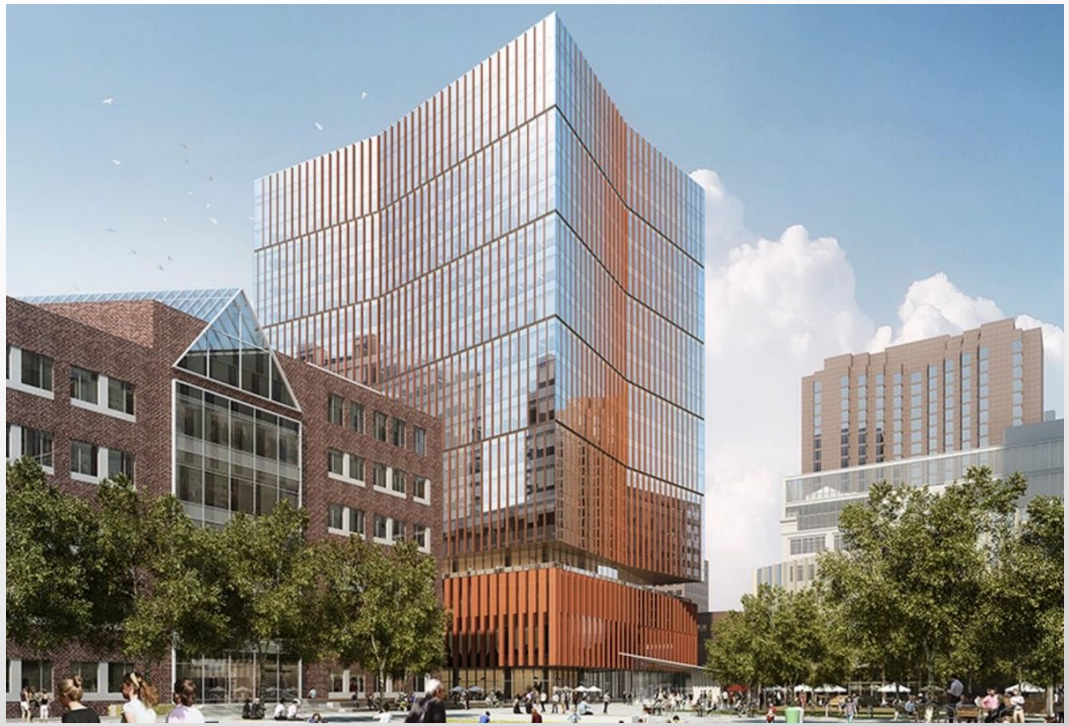
MIT-IBM Watson AI Lab, IBM Research

Presented at 47th Annual Spring Lecture Series, University of Arkansas

About me/

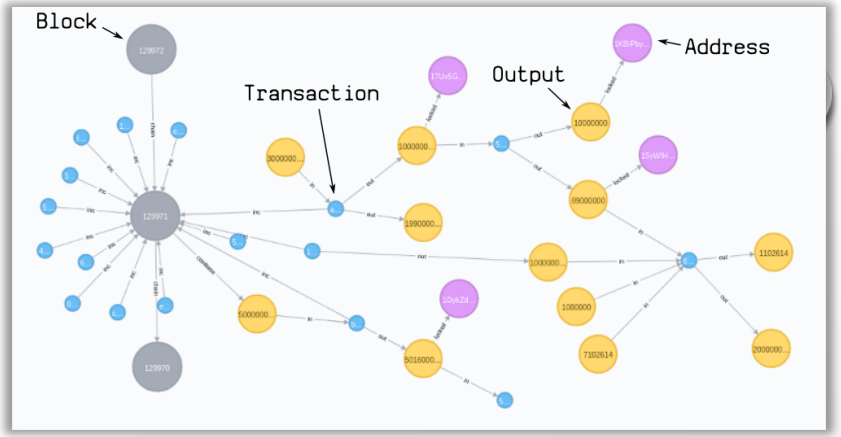
MIT-IBM Watson AI Lab

The MIT-IBM Watson AI Lab is an academic-industrial collaboration model dedicated to pushing the frontiers of artificial intelligence and translating fundamental scientific breakthroughs into business and real-world impacts.



Power grid

Social network

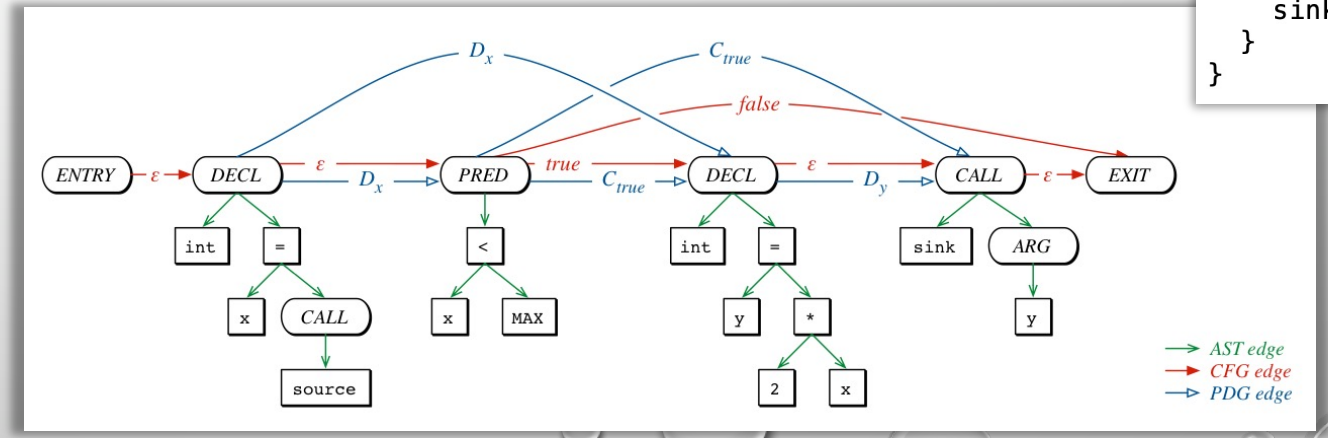


Bitcoin transactions

Graphs are ubiquitous

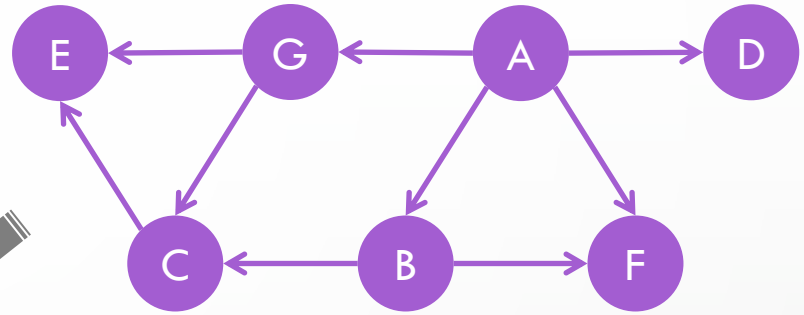
Molecules

Code graphs




```
void foo() {
  int x = source();
  if (x < MAX) {
    int y = 2 * x;
    sink(y);
  }
}
```

Sparse matrices ⇔ graphs



adjacency matrix

	A	B	C	D	E	F	G
A		1		1		1	1
B			1			1	
C					1		
D							
E							
F							
G			1		1		

reordering (topological sort)

	A	B	D	F	G	C	E
A		1	1	1	1		
B				1		1	
D							
F							
G						1	1
C							1
E							

Machine learning 101

$\xrightarrow{f}$ agaric (0.88)
jelly fungus (0.11)
fingers (0.01)

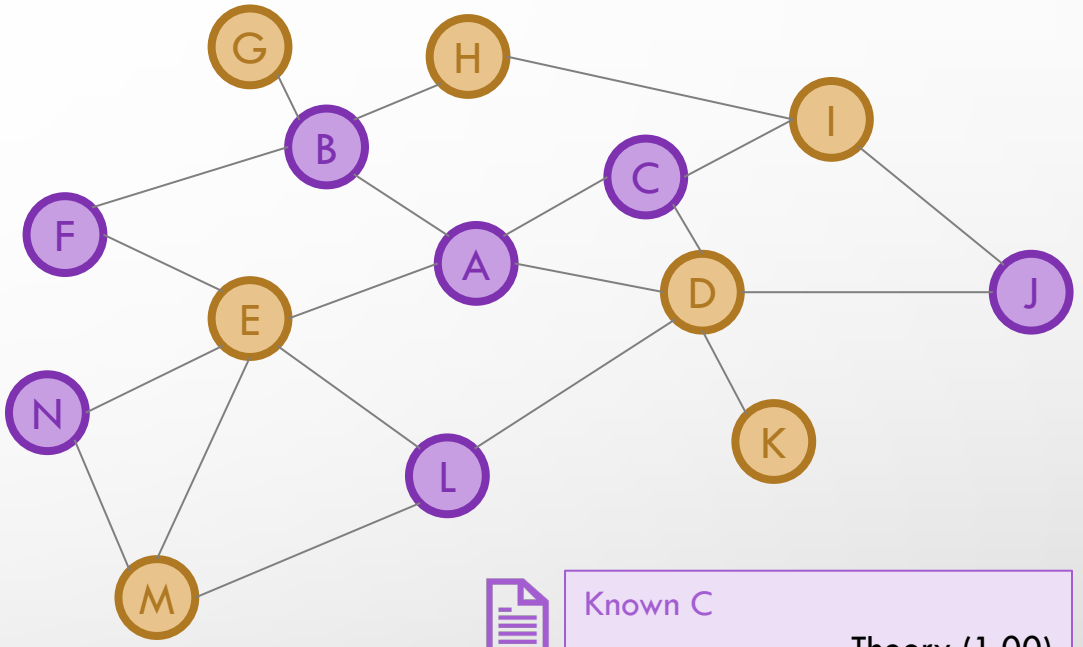
- Model $f(x;\theta)$
  - return a probability vector (length c)
  - model parameters
  - input data
- Example $f(x;\theta) = \mathrm{softmax}(W^1 \cdot \mathrm{ReLU}(W^0 x + b^0) + b^1), \quad \theta = \{W^0, W^1, b^0, b^1\}$
  - softmax(x) = (e^{x1}, e^{x2}, …, e^{xc})/normalization
  - ReLU(x) = max(x,0)
- Loss $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} L_i(\theta), \quad L_i(\theta) = -(y_i)_1 \log f(x_i;\theta)_1 - \cdots - (y_i)_c \log f(x_i;\theta)_c$
  - y_i is the labeling vector (one-hot)
- Training $\min_\theta \; L(\theta)$
  - de-facto choice: stochastic optimization

Mini-batch loss $L_B(\theta) = \frac{1}{|B|}\sum_{i\in B} L_i(\theta)$; Update rule: $\theta^{k+1} = \theta^k - \gamma^k \nabla L_B(\theta^k)$

Graph data and task

From machine learning 101:

- A data point is $x_i \in \mathbb{R}^d$ with label $y_i \in \mathbb{R}^c$
- Some labels are known, while some not



Additionally:

- Data points are connected by a (weighted) adjacency matrix A
- The task is to predict the unknown labels

Known C

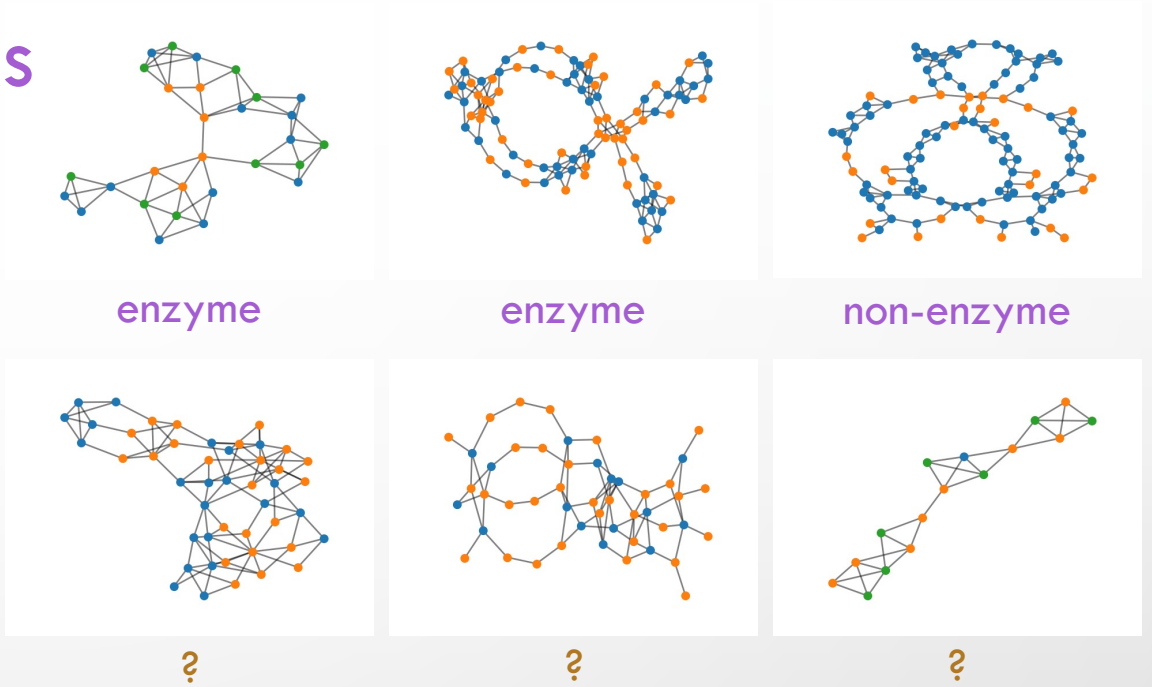
Theory (1.00)
Neural networks (0.00)
Reinforcement learning (0.00)

Predict D

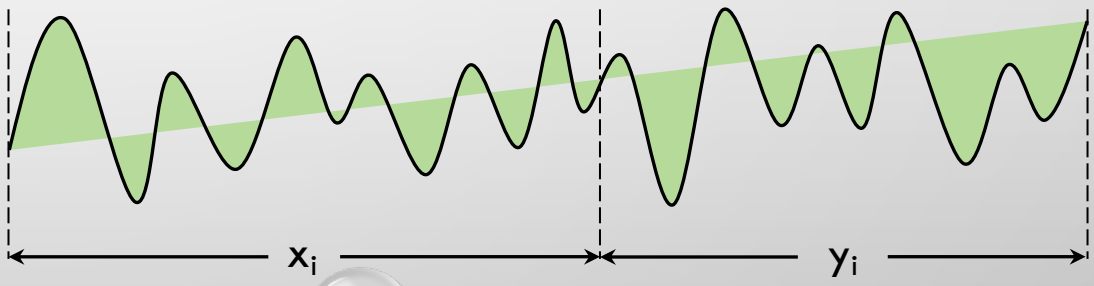
Theory (?)
Neural networks (?)
Reinforcement learning (?)

More graph data and tasks

- Graph-level prediction: The entire graph is a data point x_i ; the task is to predict the label y_i of this graph



- Time series forecasting: Split time interval $T = T_1 + T_2$, where input $x_i \in \mathbb{R}^{T_1}$ and output $y_i \in \mathbb{R}^{T_2}$. Additionally, several time series are interconnected by a graph.



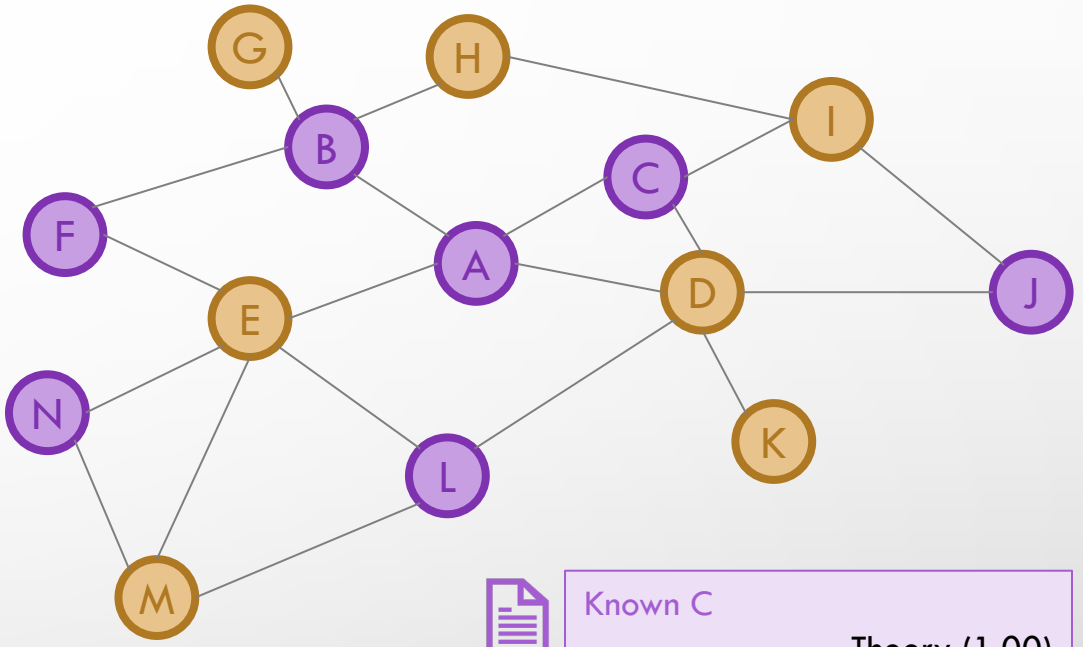
Graph neural networks (GNN)

- Graph convolution network (GCN)

$$f(X) = \mathrm{softmax}(\widehat{A} \cdot \mathrm{ReLU}(\widehat{A}XW^0) \cdot W^1)$$

$$f(X) = \quad X =$$

$$\widehat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}, \quad \widetilde{A} = A + I, \quad \widetilde{D} = \mathrm{diag}\left(\sum_j \widetilde{A}_{ij}\right)$$



Known C

Theory (1.00)
Neural networks (0.00)
Reinforcement learning (0.00)

Predict D

Theory (?)
Neural networks (?)
Reinforcement learning (?)

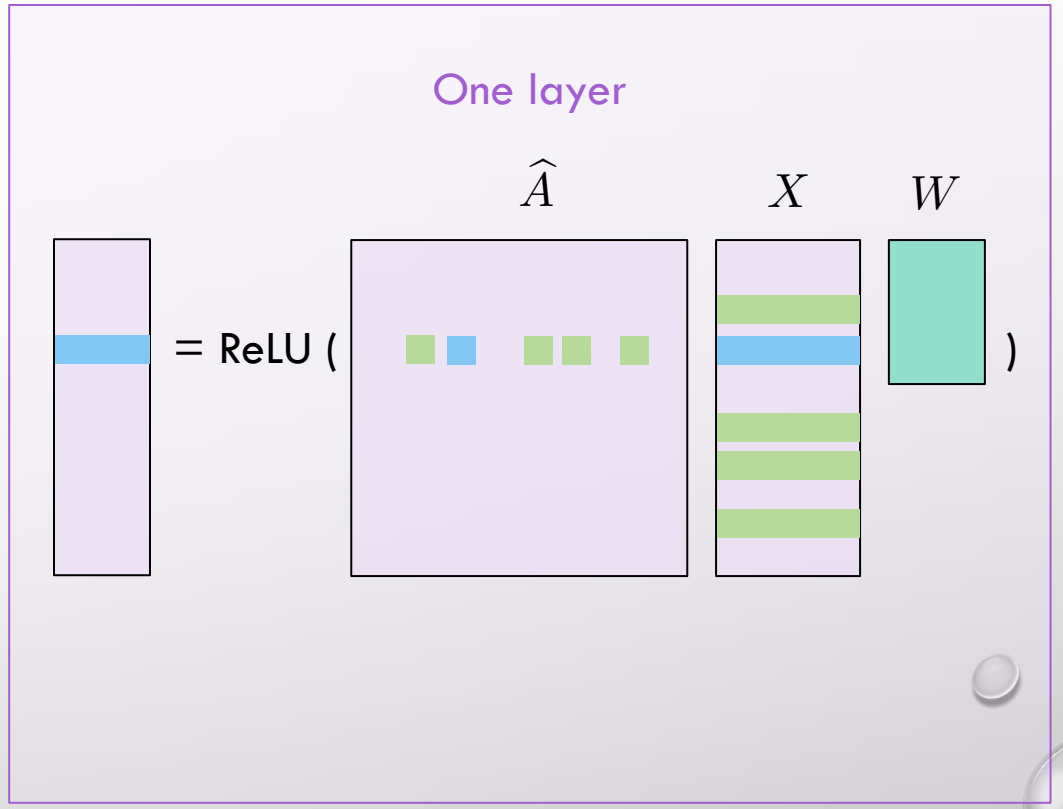
Graph neural networks (GNN) --- Matrix view

- Graph convolution network (GCN)

$$f(X) = \mathrm{softmax}(\widehat{A} \cdot \mathrm{ReLU}(\widehat{A}XW^0) \cdot W^1)$$

$$f(X) = \quad X =$$

$$\widehat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}, \quad \widetilde{A} = A + I, \quad \widetilde{D} = \mathrm{diag}\left(\sum_j \widetilde{A}_{ij}\right)$$



Graph neural networks (GNN) --- Message passing

- Graph convolution network (GCN)

$$f(X) = \mathrm{softmax}(\widehat{A} \cdot \mathrm{ReLU}(\widehat{A}XW^0) \cdot W^1)$$

$$f(X) = \quad X =$$

$$\widehat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}, \quad \widetilde{A} = A + I, \quad \widetilde{D} = \mathrm{diag}\left(\sum_j \widetilde{A}_{ij}\right)$$

- Message-passing framework

$$h_i^\ell = \mathrm{COM}^\ell\left(h_i^{\ell-1}, \mathrm{AGG}^\ell\left(\{h_j^{\ell-1} \mid j \in \mathcal{N}_i\}\right)\right), \quad \ell = 1, \ldots, L$$

$$f(x_i) = h_i^L \qquad x_i = h_i^0$$

One layer

$\widehat{A}$ X W

= ReLU ( )

AGGregate(neighbors)

COMbine(self, neighbors)

More examples of message passing

For simplicity, drop superscript (layer index): $h_i' = \mathrm{COM}\Big(h_i,\ \mathrm{AGG}\big(\{h_j \mid j \in \mathcal{N}_i\}\big)\Big)$

GraphSAGE

$$h_i' = \mathrm{ReLU}\left(W_1 h_i + W_2 \cdot \underset{j \in \mathcal{N}_i}{\mathrm{mean}}\, h_j\right)$$

Graph isomorphism network (GIN)

$$h_i' = \mathrm{NeuralNetwork}\left((1+\epsilon)\cdot h_i + \sum_{j \in \mathcal{N}_i} h_j\right)$$

Graph attention network (GAT)

$$h_i' = \mathrm{ReLU}\left(\alpha_{ii} W h_i + \sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j\right)$$

$$\alpha_{ij} = \frac{\exp(\mathrm{LeakyReLU}(a_1^T W h_i + a_2^T W h_j))}{\text{normalize over } \{j \in \{i\} \cup \mathcal{N}_i\}}$$

Questions to consider

All models are wrong;
some are useful.
– George Box

1. Why do GNNs work?
2. Do GNNs scale to massive graphs?
3. What if the graph evolves over time?
4. Are GNNs robust to perturbations of the graph?
5. Are there better-performing GNNs for special types of graphs (e.g., DAG)?
6. What if the weight matrices W are infinitely large? What if the number of layers, L, is infinite?
7. If there is not a graph, can we learn one so that we can apply GNNs?
8. How can we generate new graphs?

Learning with massive graphs

References: Chen, Ma, and Xiao (2018). FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling.

Chen and Luss (2018). Stochastic Gradient Descent with Biased but Consistent Gradient Estimators.

Recall the setting

- n nodes, d input features, c classes
- Train a model f such that $f(X) \approx Y, \quad X \in \mathbb{R}^{n \times d},\ Y \in \mathbb{R}^{n \times c},\ A \in \mathbb{R}^{n \times n}$
- Focus on the GCN model $f(X) = \mathrm{softmax}(\widehat{A} \cdot \mathrm{ReLU}(\widehat{A}XW^0) \cdot W^1)$
- Question: How do we find the parameters of f (e.g., W^0 and W^1) efficiently for large n?

Challenge

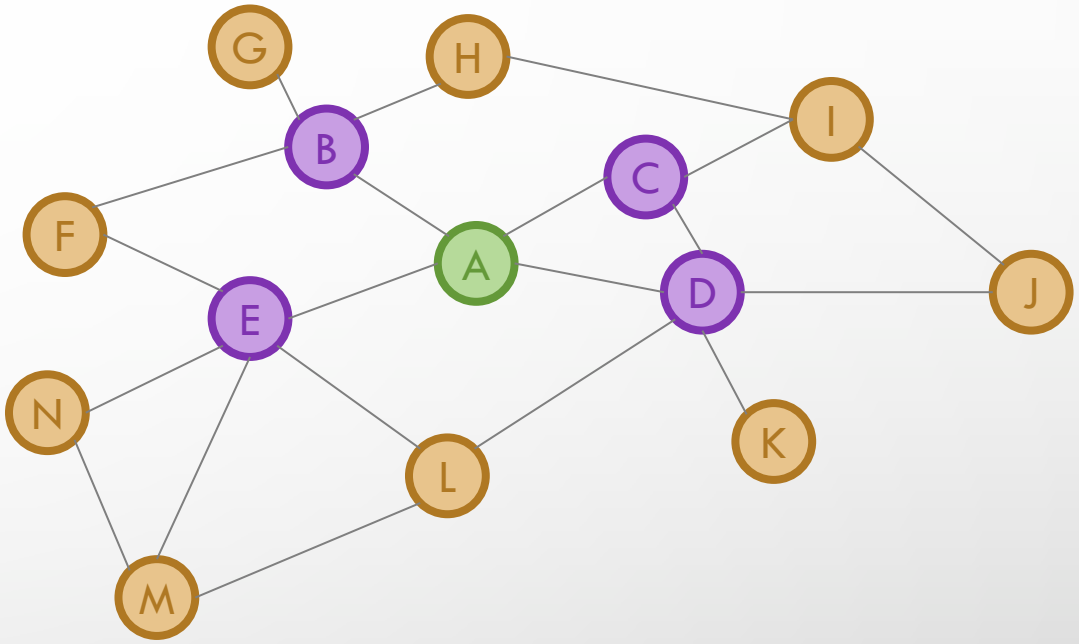
$$f(X) = \text{softmax}(\hat{A} \cdot \text{ReLU}(\hat{A}XW^0) \cdot W^1)$$

one or a small batch

need neighbors

need neighbors' neighbors

- In stochastic optimization, each step randomly picks one (or a small batch of) node(s) to compute the approximate gradient
- For GCN, the worst-case cost is O(n) per gradient step, for small-world graphs



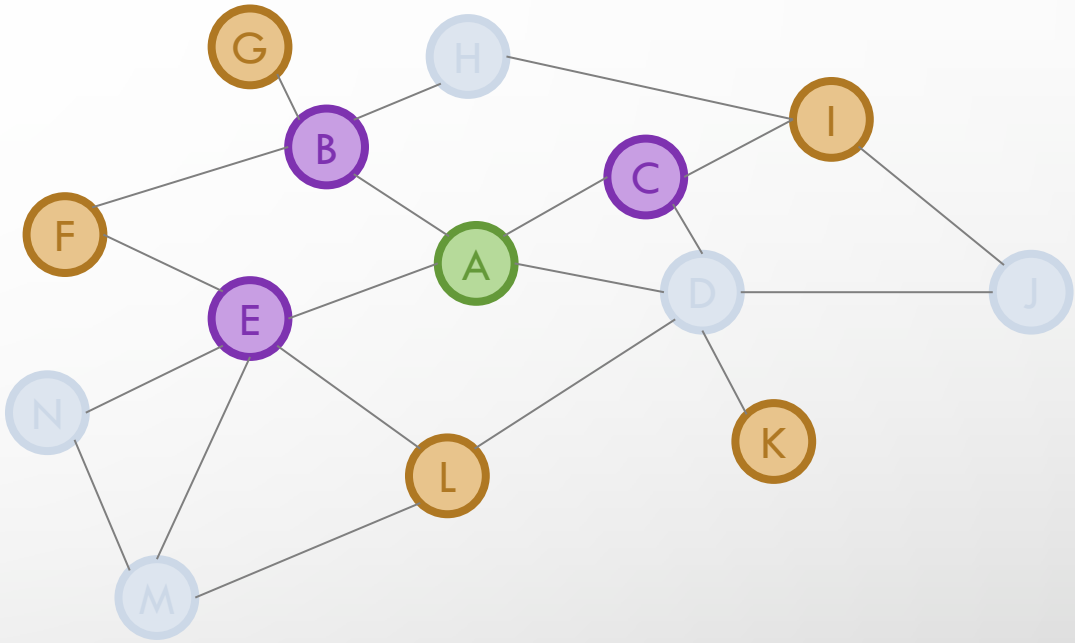
Solution: Sampling

$$f(X) = \text{softmax}(\hat{A} \cdot \text{ReLU}(\hat{A}XW^0) \cdot W^1)$$

one or a small batch

3 one-hop neighbors

5 two-hop neighbors



- Randomly pick (according to any probability distribution) a fixed number of neighbors only, for each layer
- The worst-case cost is O(1) per gradient step

Theory: Sample gradient is not unbiased, but consistent

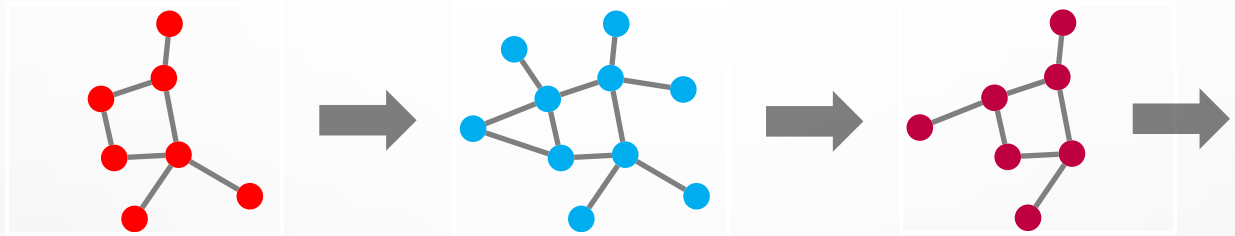
- Assume objective L(θ) is nonconvex but L-smooth; gradient is bounded with $\|\nabla L(\theta)\| \leq G$
- Use stochastic gradient update $\theta_{k+1} = \theta_k - \gamma_k g_k$ where g_k is consistent
- Specifically, $\Pr\left(\|g_k - \nabla L(\theta_k)\| \geq \delta \|\nabla L(\theta_k)\| \,\middle|\, g_1, \ldots, g_{k-1}\right) \leq C_k e^{-N_k \tau(\delta)}$ where δ is in (0,1) and τ is an increasing and positive function
- Use constant (oracle) step size $\gamma_k = D_f/[(1+\delta)G\sqrt{T}]$ where $D_f = [2(L(\theta_1) - L(\theta^*))/L]^{\frac{1}{2}}$
- For any ϵ in (0,1), using sample size $N_k \geq \tau(\delta)^{-1} \log(TC_k/\epsilon)$, with probability ≥ 1 - ϵ,

$$\min_{k=1,\ldots,T} \|\nabla L(\theta_k)\|^2 \leq \frac{(1+\delta)LGD_f}{(1-\delta)\sqrt{T}}$$

Learning with dynamic graphs

References: Pareja, et al (2020). EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs.

In most real-life applications, graphs are dynamic



- n nodes, d input features, c classes, T time steps
- Train a model f such that $f(X_t) \approx Y_t, \quad X_t \in \mathbb{R}^{n \times d},\ Y_t \in \mathbb{R}^{n \times c},\ A_t \in \mathbb{R}^{n \times n},\ t = 1, \ldots, T$
- The number of nodes, n, can change over time (but for simplicity we call it n)
- Question: How do we build such a model based on one for static graphs?

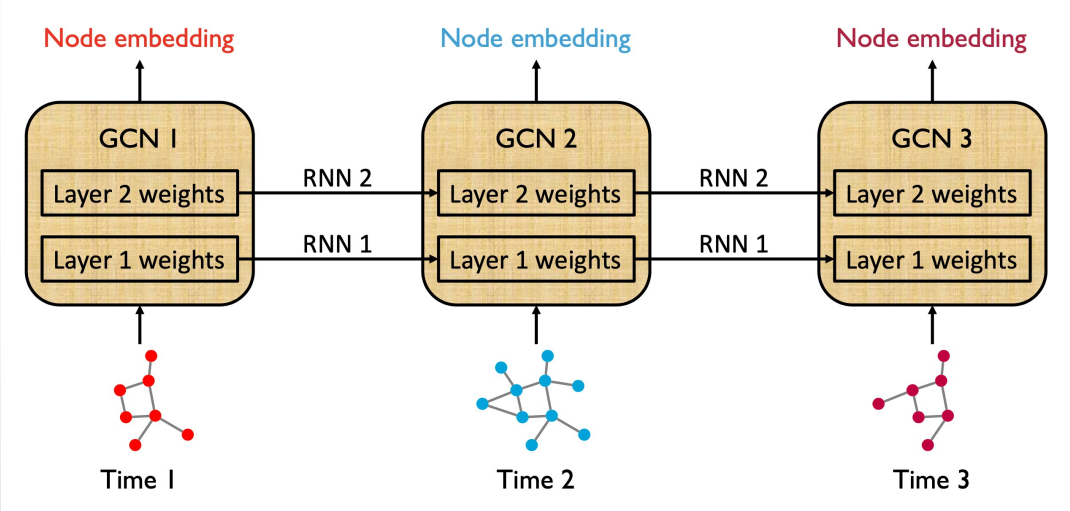
A closer look at GCN

$$f(X) = \mathrm{softmax}(\hat{A} \cdot \mathrm{ReLU}(\hat{A}XW^0) \cdot W^1)$$

$$\text{layer notation: } H^{\ell+1} = \sigma(\hat{A}H^{\ell}W^{\ell})$$

When A evolves over time, we have two choices

- Model the evolution of layer embeddings H^{ℓ} by using a recurrent neural network (RNN)
  - This is a typical approach, but it cannot handle the dynamic appearance/disappearance of nodes
- Model the evolution of layer parameters W^{ℓ} by using a recurrent neural network (RNN)



Two evolution mechanisms

- H version: GCN parameters are GRU hidden states

$$W_t^\ell = \mathrm{GRU}(H_t^\ell, W_{t-1}^\ell)$$
$$H_t^{\ell+1} = \sigma(\widehat{A}_t H_t^\ell W_t^\ell)$$

- O version: GCN parameters are LSTM input/ outputs

$$W_t^\ell = \mathrm{LSTM}(W_{t-1}^\ell)$$
$$H_t^{\ell+1} = \sigma(\widehat{A}_t H_t^\ell W_t^\ell)$$

function $H_t = \mathrm{GRU}(X_t, H_{t-1})$

// Input X_t, hidden states H_{t-1} and H_t

$Z_t = \mathrm{sigmoid}(W_Z X_t + U_Z H_{t-1} + B_Z)$

$R_t = \mathrm{sigmoid}(W_R X_t + U_R H_{t-1} + B_R)$

$\widetilde{H}_t = \tanh(W_H X_t + U_H(R_t \circ H_{t-1}) + B_H)$

$H_t = (1 - Z_t) \circ H_{t-1} + Z_t \circ \widetilde{H}_t$

end function

function $H_t = \mathrm{LSTM}(X_t)$

// Current input X_t = past output H_{t-1}

$F_t = \mathrm{sigmoid}(W_F X_t + U_F H_{t-1} + B_F)$

$I_t = \mathrm{sigmoid}(W_I X_t + U_I H_{t-1} + B_I)$

$O_t = \mathrm{sigmoid}(W_O X_t + U_O H_{t-1} + B_O)$

$\widetilde{C}_t = \tanh(W_C X_t + U_C H_{t-1} + B_C)$

$C_t = F_t \circ C_{t-1} + I_t \circ \widetilde{C}_t$

$H_t = O_t \circ \tanh(C_t)$

end function

Parameter evolution compared with other static/dynamic models

Table 2: Performance of link prediction. Each column is one data set.

	mean average precision					mean reciprocal rank				
	SBM	BC-OTC	BC-Alpha	UCI	AS	SBM	BC-OTC	BC-Alpha	UCI	AS
GCN	0.1987	0.0003	0.0003	0.0251	0.0003	0.0138	0.0025	0.0031	0.1141	0.0555
GCN-GRU	0.1898	0.0001	0.0001	0.0114	0.0713	0.0119	0.0003	0.0004	0.0985	0.3388
DynGEM	0.1680	0.0134	0.0525	0.0209	0.0529	0.0139	0.0921	0.1287	0.1055	0.1028
dyngraph2vecAE	0.0983	0.0090	0.0507	0.0044	0.0331	0.0079	0.0916	0.1478	0.0540	0.0698
dyngraph2vecAERNN	0.1593	**0.0220**	**0.1100**	0.0205	0.0711	0.0120	**0.1268**	**0.1945**	0.0713	0.0493
EvolveGCN-H	0.1947	0.0026	0.0049	0.0126	**0.1534**	**0.0141**	0.0690	0.1104	0.0899	**0.3632**
EvolveGCN-O	**0.1989**	0.0028	0.0036	**0.0270**	0.1139	0.0138	0.0968	0.1185	**0.1379**	0.2746

Our method (parameter evolution)

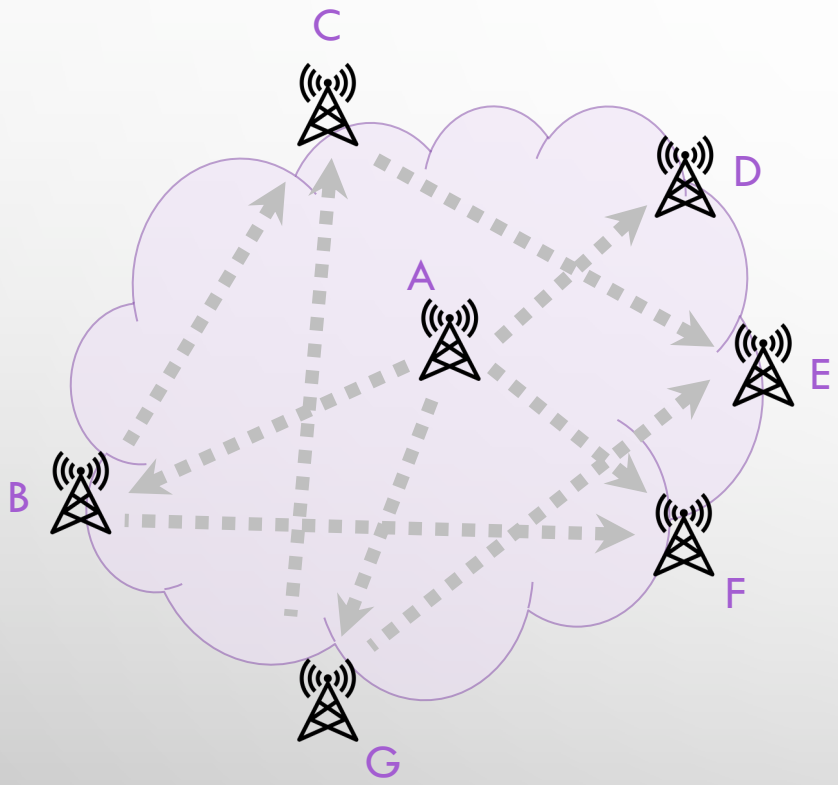
Learning to construct a graph

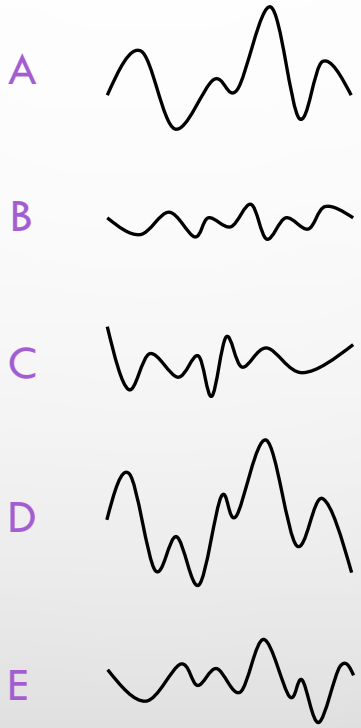
References: Shang, Chen, and Bi (2021). Discrete Graph Structure Learning for Forecasting Multiple Time Series.

Dai and Chen (2022). Graph-Augmented Normalizing Flows for Anomaly Detection of Multiple Time Series.

Yu, et al (2019). DAG-GNN: DAG Structure Learning with Graph Neural Networks.

Multiple time series





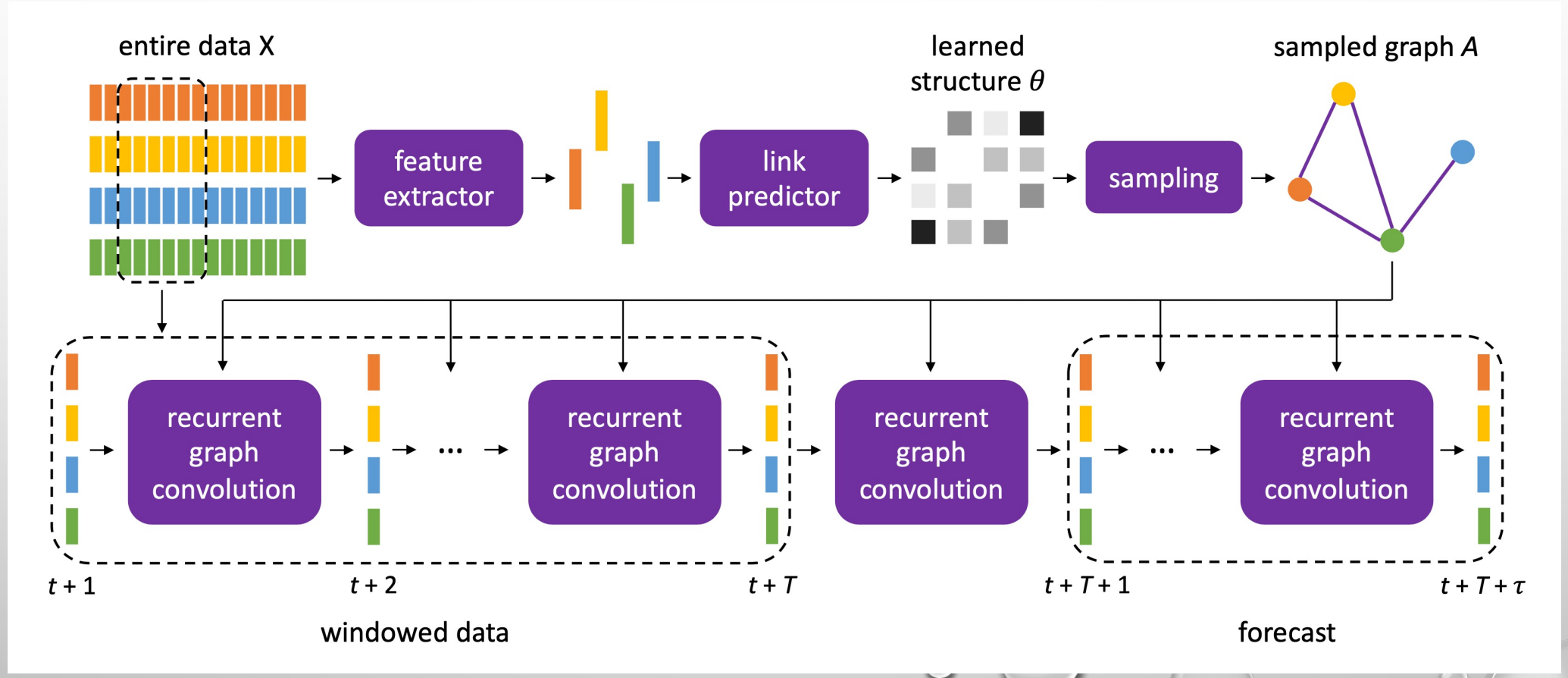
Forecasting task: Predicting values for the next few time steps

Anomaly detection task: Find out time series that behave abnormally

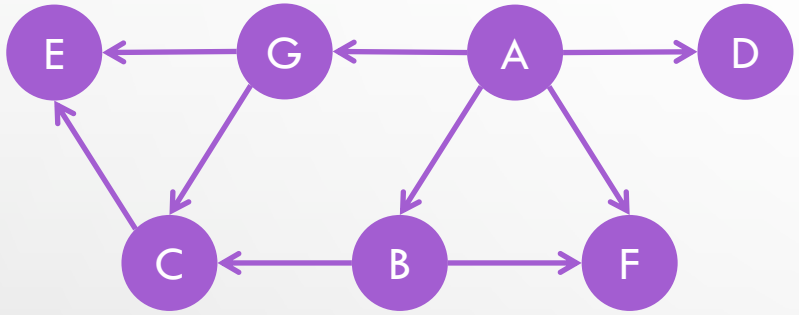
Sensors are connected, leading to interactions/interdependency

- Can we learn the interactions/interdependency?
- Do they help the task?

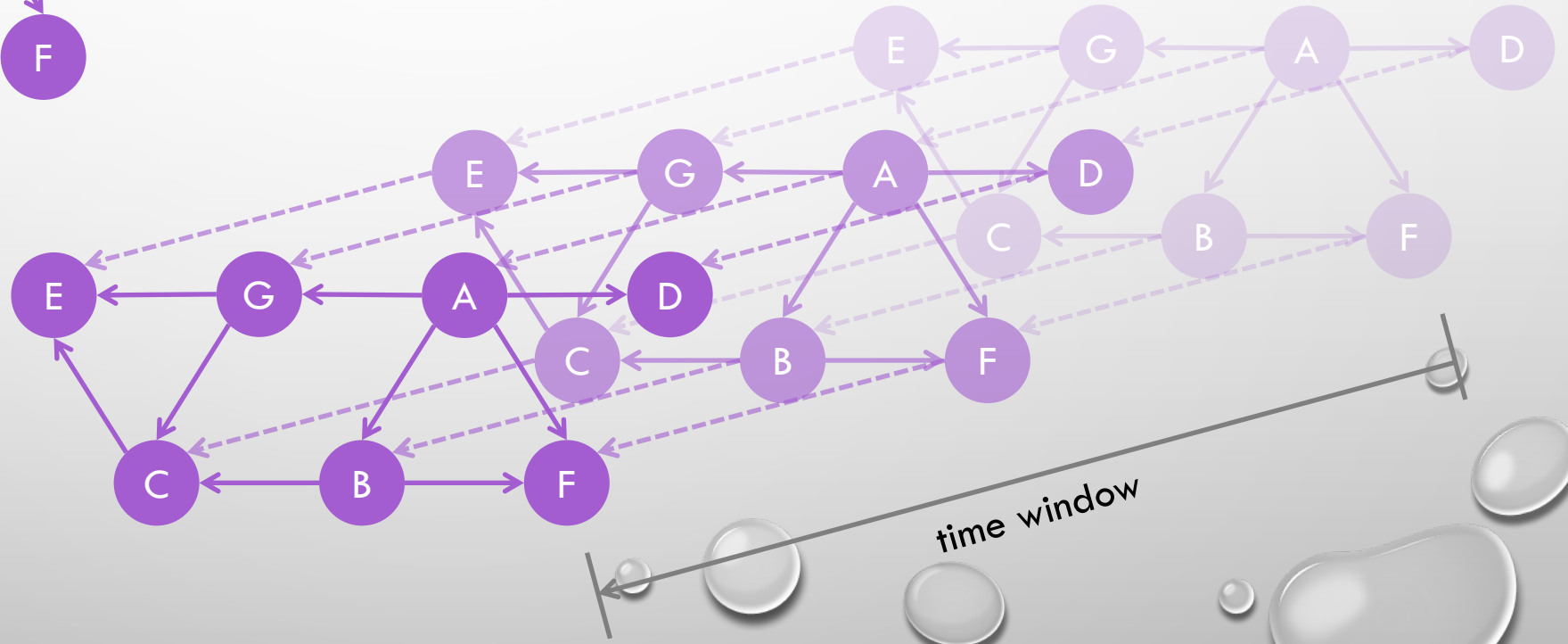
A link prediction approach



A Bayesian network approach (directed acyclic)



$$P(A,B,C,D,E,F,G) = P(E|C,G) \cdot P(G|A) \cdot P(A) \cdot P(D|A) \cdot P(C|B,G) \cdot P(B|A) \cdot P(F|A,B)$$



A Bayesian network approach

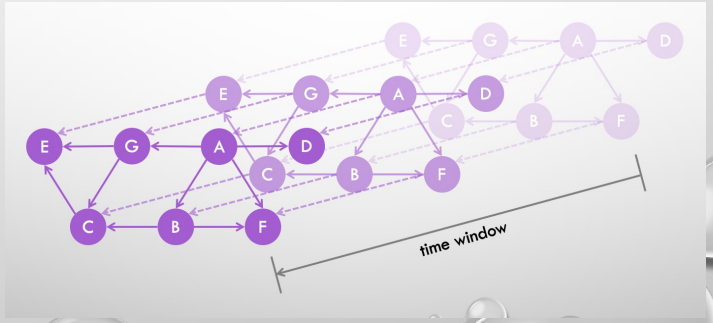
```
P(A1, B1, C1, D1, E1, F1, G1,
  A2, B2, C2, D2, E2, F2, G2,
  A3, B3, C3, D3, E3, F3, G3)

=   P(A1)         * P(A2|         A1) * P(A3|         A2)
  * P(B1|A1)      * P(B2|A2,      B1) * P(B3|A3,      B2)
  * P(C1|B1,G1)   * P(C2|B2,G2,   C1) * P(C3|B3,G3,   C2)
  * P(D1|A1)      * P(D2|A2,      D1) * P(D3|A3,      D2)
  * P(E1|C1,G1)   * P(E2|C2,G2,   E1) * P(E3|C3,G3,   E2)
  * P(F1|A1,B1)   * P(F2|A2,B2,   F1) * P(F3|A3,B3,   F2)
  * P(G1|A1)      * P(G2|A2,      G1) * P(G3|A3,      G2)
```

normalizing flow

P(B3|A3, B2)

dependency encoder; graph convolution



Yes, learning a graph indeed helps

Table 2: Forecasting error (PMU).

Our method

	Metric	FNN	LSTM	DCRNN	LDS	NRI	GTSv	GTS
15 min	MAE ($\times 10^{-3}$)	1.23	1.02	0.71	0.49	0.66	0.26	**0.24**
	RMSE ($\times 10^{-2}$)	1.28	1.63	1.42	1.26	0.27	0.20	**0.19**
	MAPE	0.20%	0.21%	0.09%	0.07%	0.14%	0.05%	**0.04%**
30 min	MAE ($\times 10^{-3}$)	1.42	1.11	1.08	0.81	0.71	0.31	**0.30**
	RMSE ($\times 10^{-2}$)	1.81	2.06	1.91	1.79	0.30	0.23	**0.22**
	MAPE	0.23%	0.20%	0.15%	0.12%	0.15%	**0.05%**	**0.05%**
60 min	MAE ($\times 10^{-3}$)	1.88	1.79	1.78	1.45	0.83	**0.39**	0.41
	RMSE ($\times 10^{-2}$)	2.58	2.75	2.65	2.54	0.46	0.32	**0.30**
	MAPE	0.29%	0.27%	0.24%	0.22%	0.17%	**0.07%**	**0.07%**

Table 1: AUC-ROC (%) of anomaly detection.

Our method

Dataset	EncDecAD	DeepSVDD	ALOCC	DROCC	DeepSAD	GANF
PMU-B	55.6±1.8	55.6±3.3	62.9±2.2	58.6±3.0	63.7±0.9	**67.5**±0.8
PMU-C	53.7±0.5	56.9±0.9	60.9±1.3	61.9±2.7	60.1±1.4	**70.6**±3.3
SWaT	76.5±0.7	68.8±2.0	75.4±2.3	73.3±1.6	75.4±1.2	**79.6**±0.9

Epilogue

Further readings

All papers can be found in https://jiechenjiechen.github.io

Efficient training

Dynamic graphs

Structure learning

DAGs

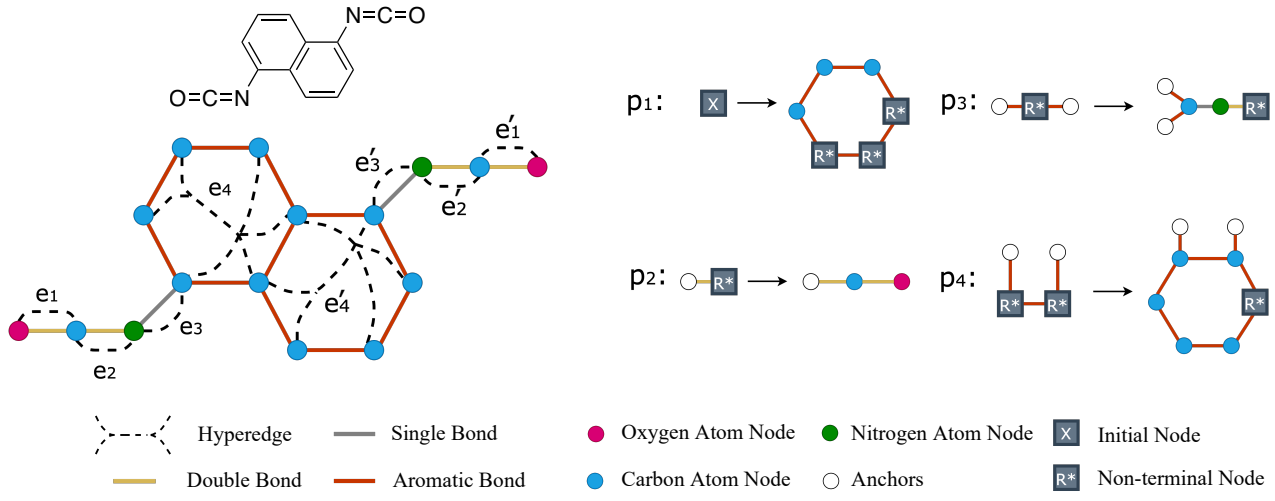
Graph generation

Coarsening

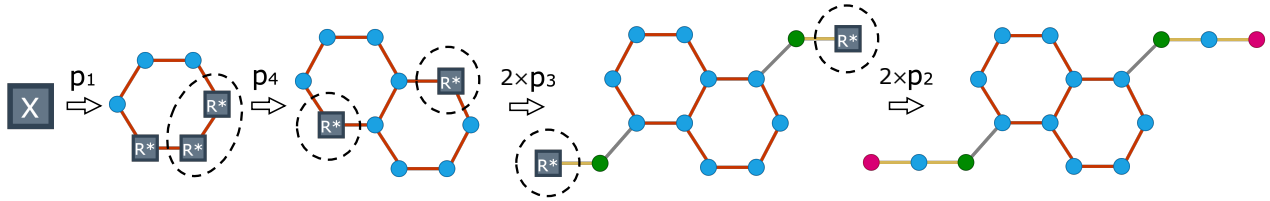
Applications

Some graph projects at MIT-IBM

Learning a graph grammar for molecule generation and optimization

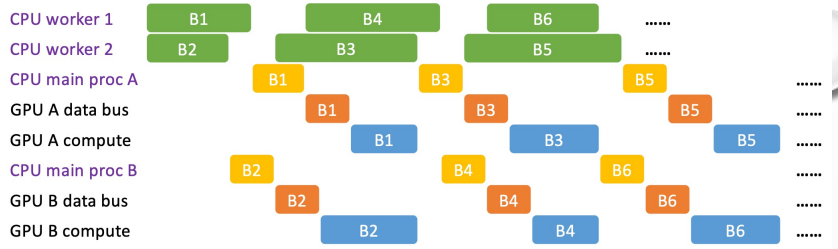


(a) Hypergraph representation of naphthalene diisocyanate (b) Production rules of a learned graph grammar

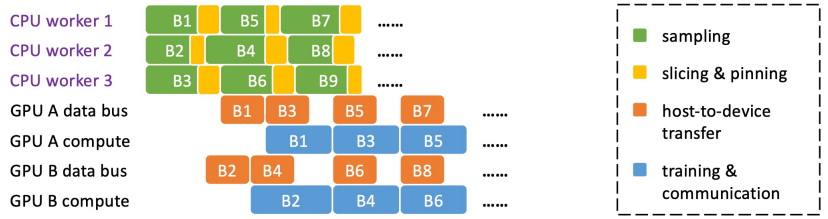


(c) Generation process of naphthalene diisocyanate using the graph grammar in (b)

Training and inference system (in-memory/out-of-core)



(a) Standard PyTorch workflow.



(b) Our system SALIENT.

Program translation through encoding-decoding parse trees