GMNN: Graph Markov Neural Networks

Jian Tang

HEC Montreal

Mila-Quebec AI Institute

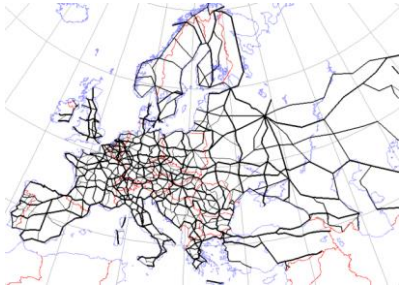
CIFAR AI Research Chair

Meng Qu, Yoshua Bengio, **Jian Tang**. “GMNN: Graph Markov Neural Networks”. To appear at ICML’19.

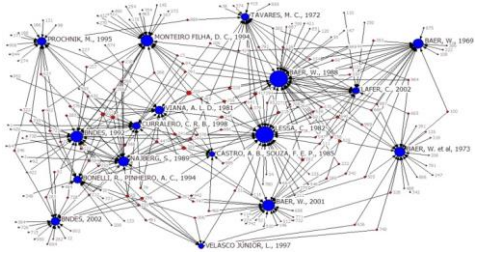
Graphs are Ubiquitous

- A general and flexible data structure to encode the relations between objects

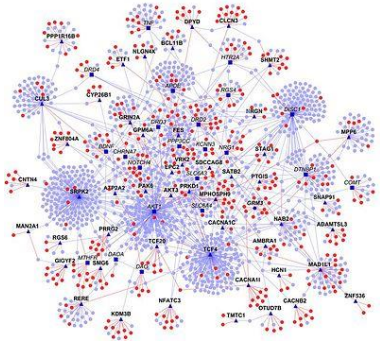
Social Network



Road Network



Citation Network

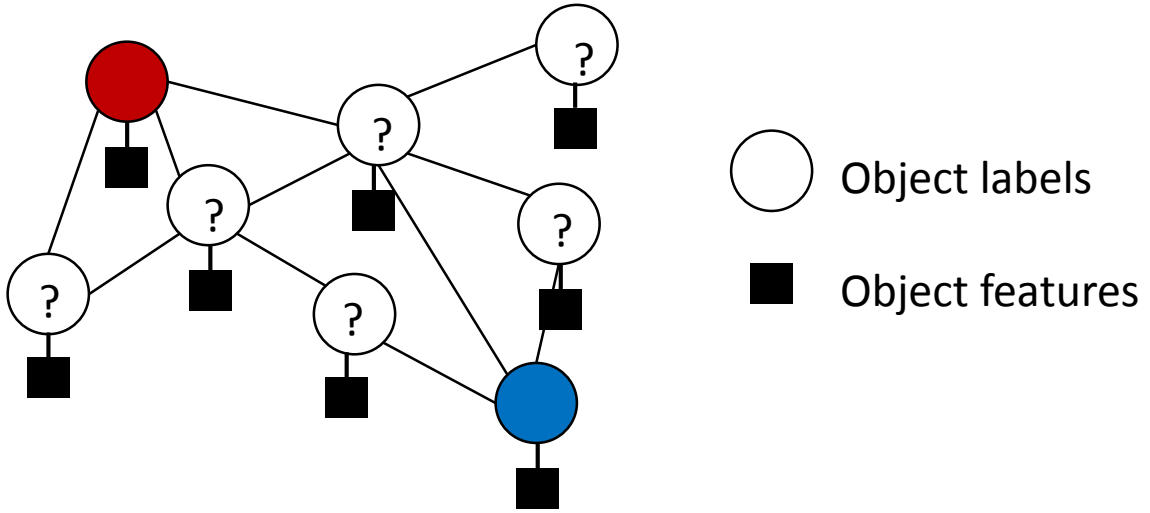


Protein-protein Interaction Network

- Cover a variety of domains and applications
  - node classification
  - Link prediction
  - Information diffusion
  - …

Semi-supervised Object Classification

- Given G= (V, E, $\mathbf{x}_V$)
  - $V = V_L \cup V_U$: objects/nodes
  - E : edges
  - $\mathbf{x}_V$: object features

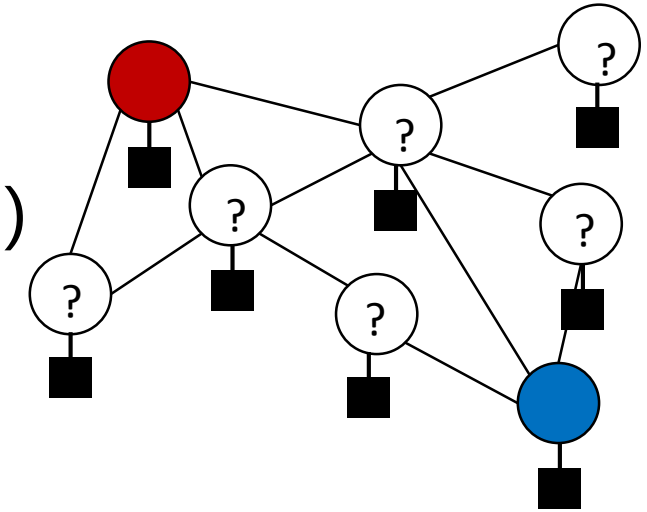


- Give some labeled objects V_L, we want to infer the labels of the rest of objects V_U
- Many other tasks on graphs can be formulated as object classification
  - E.g., link classification

Related Work: Statistical Relational Learning

- Models the joint distribution of the object labels given the object features, i.e., $p(\mathbf{y}_V|\mathbf{x}_V)$ with conditional random fields

$$p(\mathbf{y}_V|\mathbf{x}_V) = \frac{1}{Z(\mathbf{x}_V)} \prod_{(i,j)\in E} \psi_{i,j}(\mathbf{y}_i, \mathbf{y}_j, \mathbf{x}_V)$$



$$\psi_{i,j}(y_i, y_j, \mathbf{x}_V) = \exp\left(\sum_{k=1}^{K} \lambda_k f_k(y_i, y_j, \mathbf{x}_i, \mathbf{x}_j) + \mu_k g_k(y_i, \mathbf{x}_i)\right)$$

Edge potential functions

node potential functions

Optimization of Statistical Relational Learning

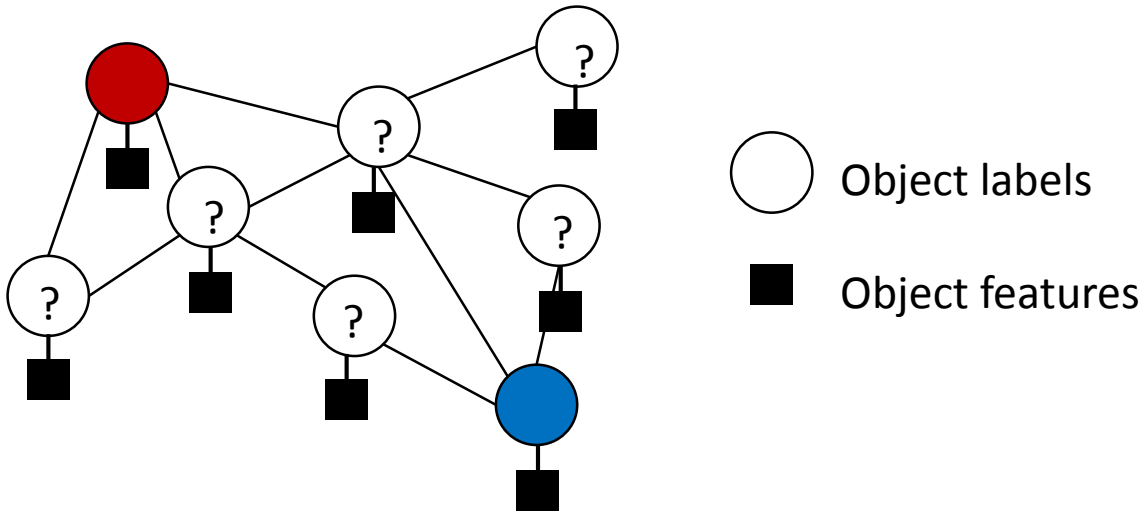
- Learning by maximizing the likelihood of the observed labels $\log p(\mathbf{y}_{\mathrm{L}}|\mathbf{x}_{\mathrm{V}})$
- Inferring the posterior distributions $p(\mathbf{y}_{\mathrm{U}}|\mathbf{y}_{\mathrm{L}},\mathbf{x}_{\mathrm{V}})$ with approximate methods such as loop belief propagation is used

Pros and Cons of Statistical Relation Learning

- Pros
  - Capable of modeling the dependency between the object labels
- Cons
  - Some manually defined potential functions
  - Limited model capacity
  - Inference is difficult due to the complicated graph structures

Related Work: Graph Neural Networks

- Learning effective node representations and then predicting the node labels independently
  - Graph convolutional Networks (Kipf et al. 2016)
  - Graph attention networks (Veličković et al. 2017)
  - Neural message passing (Gilmer et al. 2017)
- Predicting the node labels independently with the node representations



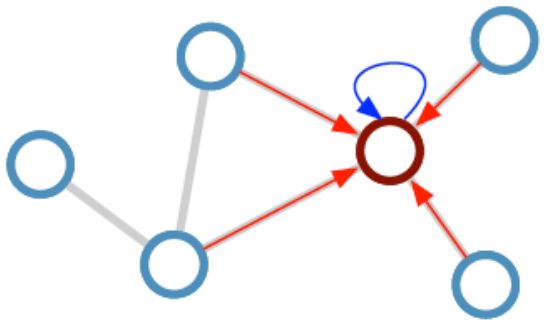
Graph Convolutional Networks (Kipf et al. 2016)

- Iteratively update the node representations by aggregating the node representations of neighbors and its own node representations
  - Starting from the initial node features $H_0 = \mathbf{x}_V$

$$H_{(l+1)} = f(H_l, A)$$

$$\hat{A} = A + I$$

$$f(H_l, A) = \sigma(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H_l \, W_l)$$



Pros and Cos of Graph Neural Networks

- Pros
  - Learns effective node representations by feature propagations
  - High model compacity by using multiple non-linear graph convolutional layers
- Cons
  - Ignore the dependency between node labels

Can we combine the advantages of both worlds?

GMNN: Graph Markov Neural Networks (Qu, Bengio, and Tang, ICML'19)

- Towards combining statistical relational learning and graph neural networks
- Learning effective node representations for predicting the node labels
- Modeling the label dependencies of nodes
- State-of-the-art performance
  - semi-supervised node classification
  - unsupervised node representation
  - link classification

GMNN: Graph Markov Neural Networks

- Model the joint distribution of object labels $\mathbf{y}_V$ conditioned on object attributes $\mathbf{x}_V$, i.e., $\mathrm{p}_\phi(\mathbf{y}_V|\mathbf{x}_V)$
- Learning the model parameters ϕ by maximizing the lower-bound of log-likelihood of the observed data, $\log \mathrm{p}_\phi(\mathbf{y}_L|\mathbf{x}_V)$

$$\log p_\phi(\mathbf{y}_L|\mathbf{x}_V) \geq$$
$$\mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_L, \mathbf{y}_U|\mathbf{x}_V) - \log q_\theta(\mathbf{y}_U|\mathbf{x}_V)]$$

Optimization with Pseudolikelihood Variational-EM

- E-step: fix p_{ϕ} and update the variational distribution $\mathrm{q}_{\theta}(\mathbf{y}_U|\mathbf{x}_V)$ to approximate the true posterior distribution $\mathrm{p}_{\phi}(\mathbf{y}_U|\mathbf{y}_L, \mathbf{x}_V)$.
- M-step: fix q_{θ} and update p_{ϕ} to maximize the lower bound

$$\ell(\phi) = \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_L, \mathbf{y}_U|\mathbf{x}_V)]$$

- Directly optimize the joint likelihood is difficult due to the partition function in p_{ϕ}, instead we optimize the pseudolikelihood function

$$\ell_{PL}(\phi) \triangleq \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n \in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{V\setminus n}, \mathbf{x}_V)]$$

$$= \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n \in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB(n)}}, \mathbf{x}_V)]$$

Inference/E-step: approximate $\mathrm{p}_{\phi}(\mathbf{y}_U|\mathbf{y}_L, \mathbf{x}_V)$

- Approximate it with variational distribution $\mathrm{q}_{\theta}(\mathbf{y}_U|\mathbf{x}_V)$. Specifically we use mean-field method:

$$q_{\theta}(\mathbf{y}_U|\mathbf{x}_V) = \prod_{n \in U} q_{\theta}(\mathbf{y}_n|\mathbf{x}_V)$$

- We parametrize each variational distribution with a Graph Neural Network

$$q_{\theta}(\mathbf{y}_n|\mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W_{\theta}\mathbf{h}_{\theta,n}))$$

Object representations learned by GNN

Inference/E-step: approximate $\mathrm{p}_{\phi}(\mathbf{y}_U|\mathbf{y}_L, \mathbf{x}_V)$

- The optimal variational distribution satisfies:

$$\log q_\theta(\mathbf{y}_n|\mathbf{x}_V) =$$
$$\mathbb{E}_{q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)}, \mathbf{x}_V)] + \text{const.}$$

- Estimate the right term by sampling from $\mathrm{q}_\theta\left(\mathbf{y}_{NB(n)\cap U}|\mathbf{x}_V\right)$, and then we have

$$q_\theta(\mathbf{y}_n|\mathbf{x}_V) \approx p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)}, \mathbf{x}_V)$$

Label distribution of object n by the learning module

Inference/E-step: approximate $\mathrm{p}_{\phi}(\mathbf{y}_U|\mathbf{y}_L, \mathbf{x}_V)$

- Minimize the KL-divergence between the two distributions
  - The supervision from the learning module is used as pseudo label to train the variational distribution

$$O_{\theta,U} = \sum_{n \in U} \mathbb{E}_{p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)}, \mathbf{x}_V)}[\log q_\theta(\mathbf{y}_n|\mathbf{x}_V)]$$

- The variational distribution can also by trained on the labeled data

$$O_{\theta,L} = \sum_{n \in L} \log q_\theta(\mathbf{y}_n|\mathbf{x}_V).$$

- Final objective:

$$O_\theta = O_{\theta,U} + O_{\theta,L}$$

Learning/M-step:

- The log-pseudo likelihood:

$$\ell_{PL}(\phi) \triangleq \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n\in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{V\setminus n}, \mathbf{x}_V)]$$

$$= \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n\in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB(n)}}, \mathbf{x}_V)]$$

- According to the inference, only the $p_\phi(\mathbf{y_n}|\mathbf{y}_{\mathrm{NB(n)}}, \mathbf{x_V})$ is required
- Parametrize $p_\phi(\mathbf{y_n}|\mathbf{y}_{\mathrm{NB(n)}}, \mathbf{x_V})$ with another GCN

$$p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)}, \mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W_\phi \mathbf{h}_{\phi,n}))$$

Learning/M-step:

- Estimate the expectation by drawing a sample from $\mathrm{q}_\theta(\mathbf{y}_U|\mathbf{x}_V)$
- Final objective:

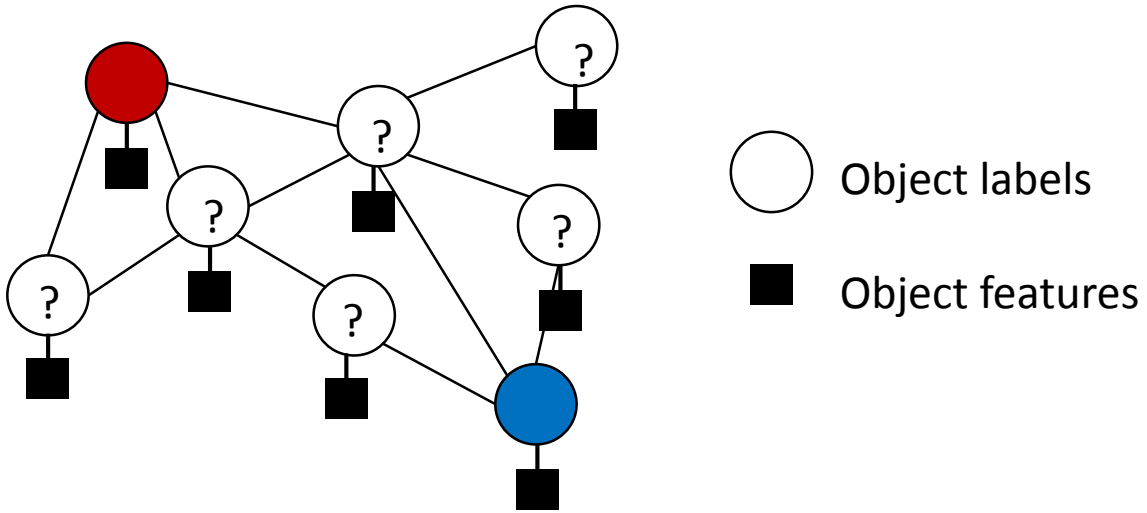
$$O_\phi = \sum_{n \in V} \log p_\phi(\hat{\mathbf{y}}_n | \hat{\mathbf{y}}_{\mathrm{NB(n)}}, \mathbf{x}_V)$$

$\hat{\mathbf{y}}_n \sim \mathrm{q}_\theta(\mathbf{y}_U|\mathbf{x}_V)$, if n is unlabeled

$\hat{\mathbf{y}}_n$ is the ground truth label, if n is labeled

Overall Optimization Procedure

- Two Graph Neural Networks Collaborate with each other
  - p_ϕ: learning network, modeling the label dependency
  - q_θ: inference network, learning the object representations
- q_θ infer the labels of unlabeled objects trained with supervision from p_ϕ and labeled objects
- p_ϕ is trained with a fully labeled graph, where the unlabeled objects are labeled by q_θ



Applications: Object/Node Classification

- Train, validation, and test are standard split
- State-of-the-art performance

Category	Algorithm	Cora	Citeseer	Pubmed
SSL	LP	74.2	56.3	71.6
SRL	PRM	77.0	63.4	68.3
	RMN	71.3	68.0	70.7
	MLN	74.6	68.0	75.3
GNN	Planetoid *	75.7	64.7	77.2
	GCN *	81.5	70.3	79.0
	GAT *	83.0	72.5	79.0
GMNN	W/o Attr. in p_ϕ	83.4	**73.1**	81.4
	With Attr. in p_ϕ	**83.7**	72.9	**81.8**

Results of Node Classification

- Random data splits
- State-of-the-art performance

Algorithm	Cora	Citeseer	Pubmed
GCN	81.5	71.3	80.3
GAT	82.1	71.5	80.1
GMNN	**83.1**	**73.0**	**81.9**

Few-shot Learning Settings

- 5 labeled objects for each class
- The performance gain are even larger

Algorithm	Cora	Citeseer	Pubmed
GCN	74.9	69.0	76.9
GAT	77.0	68.9	75.4
GMNN	**78.6**	**72.7**	**79.1**

Ablation Study on the Learning Network $p_\phi(\mathbf{y_n}|\mathbf{y}_{\mathrm{NB(n)}})$

- 1 mean pooling layer: just take the average distribution of labels of neighbors,
  - label propagation!
  - Model the label dependency in a linear way

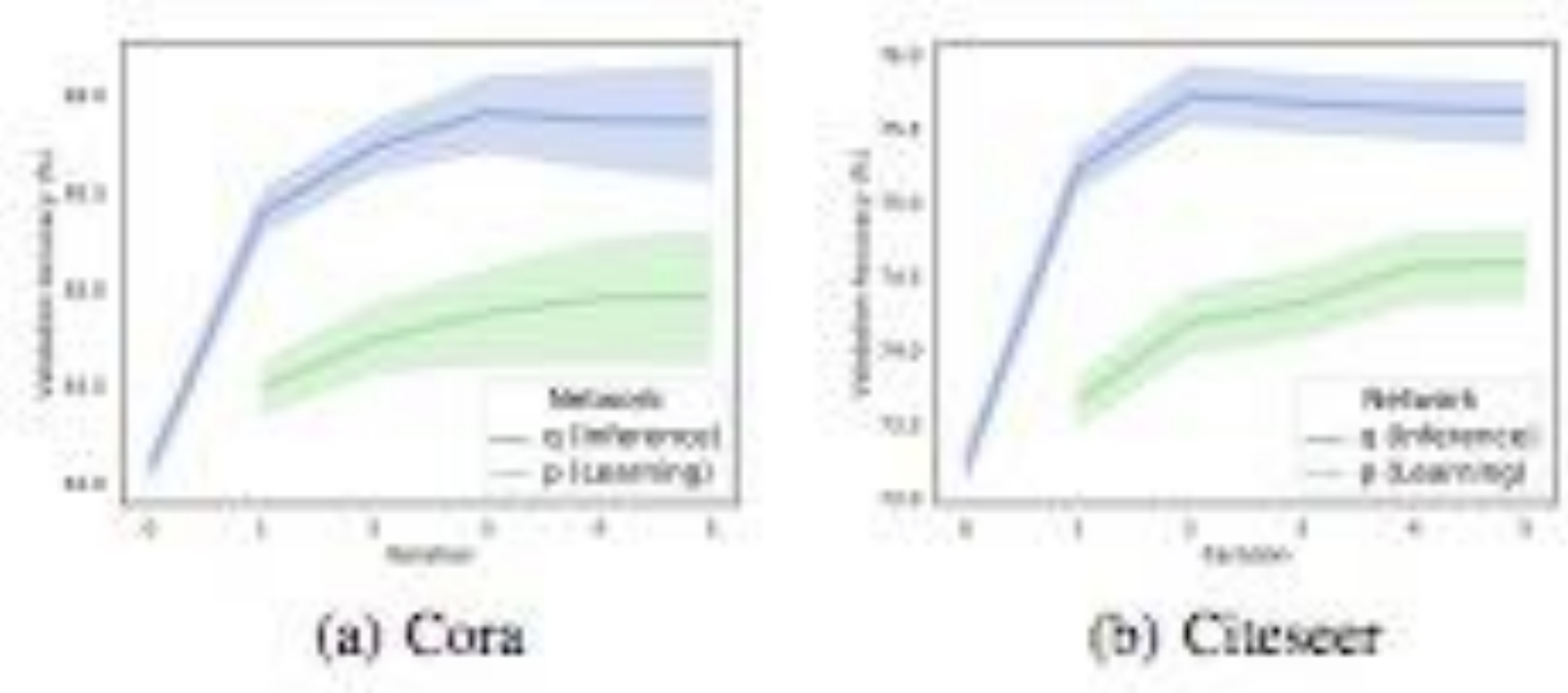
Architecture	Cora	Citeseer	Pubmed
1 Mean Pooling Layer	82.4	71.9	80.7
1 GC Layer	83.1	73.1	80.9
2 GC Layers	83.4	73.1	81.4
3 GC Layers	83.6	73.0	81.5

Ablation Study on the Inference Network $\mathrm{q}_\theta(\mathbf{y}_n|\mathbf{x}_V)$

- Non-amortized: treat $\mathrm{q}_\theta(\mathbf{y}_n|\mathbf{x}_V)$ as parameter, independent of $\mathbf{x}_V$
- 1 Linear Layer: only use the features $\mathbf{x}_n$

Architecture	Cora	Citeseer	Pubmed
Non-amortized	45.3	28.1	42.2
1 Linear Layer	55.8	57.5	69.8
1 GC Layer	72.9	67.6	71.8
2 GC Layers	83.4	73.1	81.4
3 GC Layers	82.0	70.6	80.7

Convergence Analysis of Optimization



(a) Cora

(b) Citeseer

Applications: Unsupervised Node Representation Learning

- There are no labeled nodes!!
- Instead, we introduce a pseudo task. For each node n, we aims to predict the neighbors , i.e., $p(y \mid \mathrm{n}, \mathbf{x}_V)$
- E-step: infer the neighbor distribution for each node with q_θ
- M-step: update the p_ϕ to model the local dependency of the inferred neighbor distributions

Node/Object classification

- DGI: Deep Graph Infomax, Veličković et al. 2019

Category	Algorithm	Cora	Citeseer	Pubmed
GNN	DeepWalk *	67.2	43.2	65.3
	DGI *	82.3	**71.8**	76.8
GMNN	With only q_θ .	78.1	68.0	79.3
	With q_θ and p_ϕ	**82.8**	71.5	**81.6**

Applications: Link Classification

- Construct a dual graph $\tilde{G}$ from the original graph G
  - Each edge in G -> a node in $\tilde{G}$
  - Two nodes in $\tilde{G}$ are connected if the corresponding edges in G share a node

Category	Algorithm	Bitcoin Alpha	Bitcoin OTC
SSL	LP	59.68	65.58
SRL	PRM	58.59	64.37
	RMN	59.56	65.59
	MLN	60.87	65.62
GNN	DeepWalk	62.71	63.20
	GCN	64.00	65.69
GMNN	W/o Attr. in p_φ	65.59	66.62
	With Attr. in p_φ	65.86	66.83

Summary

- A fundamental problem on graphs: semi-supervised node classification
- GMNN: towards combining statistical relational learning and graph neural networks
  - Model the label dependency with one graph neural network
  - Learn effective node representations with another graph neural network
- State-of-the-art results on semi-supervised node classification, unsupervised node representation, and link classification
- Code available at: https://github.com/DeepGraphLearning/GMNN

Questions?