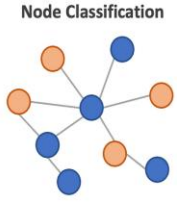
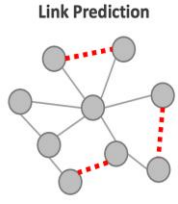
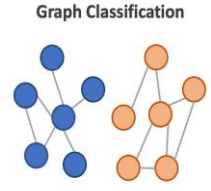
Graph 5. Graph Neural Networks (GNNs) for Node Classification

1

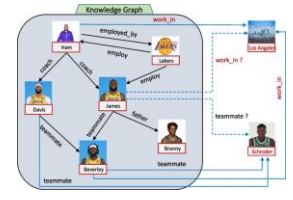
Graph-based Prediction Tasks (from Wikipedia)

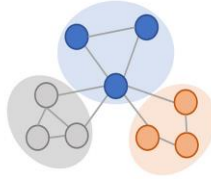


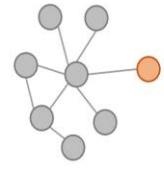




Knowledge Graph Completion





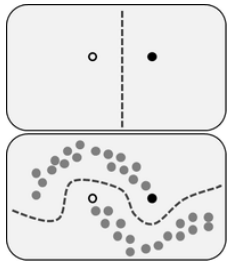


Combinatorial Optimization (e.g., TSP)

2

Node Classification with Semi-supervised Learning (SSL)

- Upper: Decision boundary based on labeled data only



- Lower: Decision boundary based on labeled + unlabeled data

How to support SSL for node classification?

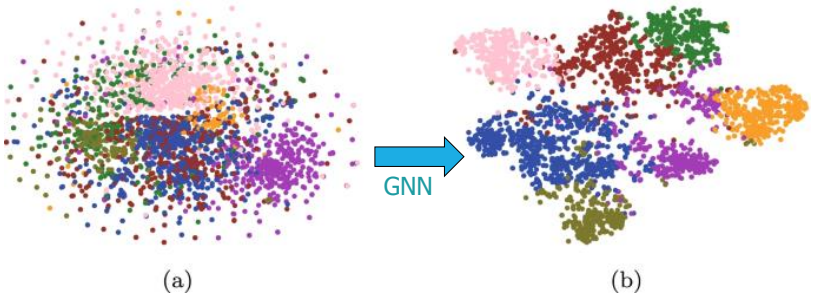
- **Approach 1**. Add a regularization term in the loss function of the classifier (e.g., Laplacian SVM);

$$\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_{reg}, \text{ with } \mathcal{L}_{reg} = \hat{\boldsymbol{y}}^T L \hat{\boldsymbol{y}} = \frac{1}{2}\sum_{i=i}^{n} A_{ij} \left\| \hat{y}_i - \hat{y}_j \right\|^2$$

  - $\mathcal{L}_0$ is the training-set loss w.r.t. the labeled nodes in the graph;
  - $\mathcal{L}_{reg}$ is the smoothness penalty of label propagation over unlabeled data;
  - $\hat{\boldsymbol{y}}$ is the system-predicted scores w.r.t. the label of yes.

- **Approach 2**. Use Graph Neural Networks (GNNs) for node embeddings (this lecture)
  - Geometrically move the node embeddings to be closer if they have a shared label.
  - In other words, making the data easily separable for the classifiers.

Task-aware node embeddings via GNN



5

Difference from the methods in previous lecture

- **Laplacian Eigenmaps and DeepWalk**
  - Both methods ignore the node-specific features and labels during embedding
- **Graph Neural Networks** (GCN, GAT, etc.)
  - Jointly leveraging node-specific features, link structures, and a training-set labels of nodes.

6

Two types of GNNs for node classification

- GNNs with Spatial Convolution
  - ICLR 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, **Yoshua Bengio**. “Graph attention networks”(GAT)
- GNNs with Spectral Graph Convolution
  - [ICLR 2017] Thomas N. Kipf and **Max Welling**. "Semi-supervised classification with graph convolutional networks" (GCN)

Notation

- $G = (V, A, X)$ is an input graph (e.g., a citation graph);
  - V is the set of nodes and $|V| = n$ (e.g., each node is an article in a citation graph);
  - $A_{n\times n}$ is the adjacency matrix with non-negative elements and zero at diagonal;
  - $X \in \mathbb{R}^{n\times d}$ is the input matrix of node-specific features (one row per node, denoted as X_i), e.g., the BERT-produced embedding per article;
- $Z \in \mathbb{R}^{n\times d'}$ is the matrix of the GNN-produced node embeddings;
- $Y_i \in \{0,1\}^K$ is the label vector of node i for $i = 1, \ldots, n$;
- $\hat{Y}_i \in \mathbb{R}^K$ is the vector of system−predicted scores of lables given node i;
- $\mathcal{D}_l = \{(X_i, Y_i)\}$ is a training set of labeled nodes.

Graph Attention Networks (GAT) [P. Velickovic et al., ICLR 2018]

- A multi-head transformer module at each layer
  - Input $H = \{\vec{h_1}, \vec{h_2}, \dots, \vec{h_n}\}$ $\quad \vec{h_i} \in \mathbb{R}^d$
  - Output $H' = \{\vec{h_1'}, \vec{h_2'}, \dots, \vec{h_n'}\}$ $\quad \vec{h_i'} \in \mathbb{R}^{d'}$
  - Attention between two nodes

    $e_{ij} = \theta^T[\mathrm{W}\vec{h_i} \parallel \mathrm{W}\vec{h_j}]$ " ||" is the concatenation operator;

    $\alpha_{ij} = \frac{exp(LeakReLU(e_{ij}))}{\sum_{j' \in \mathcal{N}_i} exp(LeakReLU(e_{ij'}))}$ $\mathcal{N}_i$ is the neighborhood of node i;

    $W \in R^{d' \times d}$ and $\theta \in R^{2d'}$ are the trainable model parameters for the layer;
  - Time complexity per attention assuming $|\mathcal{N}_i| = k < d'$: $O(dd') + O(d') + O(k) = O(dd')$.

Multi-head Node Embedding in GAT

- Intermediate layer

$$\vec{h_i'} := \Big\Vert_{m=1}^{M} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^m W^m \vec{h_j}\right) \qquad \left(\vec{h_i'} \in \mathbb{R}^{Md'}\right)$$

  $\sigma(.)$ is a non-linear operator (e.g., Exponentially Linear Unit or ELU).

- Final later

$$\vec{h_i'} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^m W^m \vec{h_j}\right) \qquad \left(\vec{h_i'} \in \mathbb{R}^{d'}\right)$$

  $\sigma(.)$ is SoftMax or sigmoid activation for multi-class or multi-label problems.

- Model Training (for multi-class classification)
  - Minimizing a cross-entropy loss over labeled training instances with Adam SGD.

Evaluation Settings for Node Classification

- **Transductive Learning**
  - Train the model with a partially labeled graph and predict the labels for the unlabeled nodes.
  - Citation Graphs: Cora, Citeseer and PutMed (with 2.7k-19k nodes) where the nodes are documents, the edges are citation links, and each document has one and only one label (multi-class problems).
- **Inductive Learning**
  - Train the model with a set of labeled graphs, predict the node labels on new graphs;
  - Using 20 PPI (protein-protein interaction) graphs for training, 2 PPI graphs for validation, and 2 PPI graphs for testing

Node Classification Results in Transductive Learning
(P. Velickovic et al., ICLR 2018)

Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

	Transductive		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	81.7 ± 0.5%	—	78.8 ± 0.3%
GCN-64*	81.4 ± 0.5%	70.9 ± 0.5%	**79.0** ± 0.3%
GAT (ours)	**83.0** ± 0.7%	**72.5** ± 0.7%	**79.0** ± 0.3%

MLP: ignoring graph structure

DeepWalk+LR(OVA): ignoring node features

GCN with signal passing over the given links in the graph (next)

GAT: multi-layer transformer module with graph-restricted attentions

Node Classification Results in Inductive Learning
(P. Velickovic et al., ICLR 2018)

Table 3: Summary of results in terms of micro-averaged F_1 scores, for the PPI dataset. GraphSAGE* corresponds to the best GraphSAGE result we were able to obtain by just modifying its architecture. Const-GAT corresponds to a model with the same architecture as GAT, but with a constant attention mechanism (assigning same importance to each neighbor; GCN-like inductive operator).

	Inductive
Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

Semi-supervised Classification with Graph Convolution Network (GCN) [Kipf and Welling, ICLR 2017]

Key Ideas & Contributions

- Using a (two-layer) neural network with graph convolution for node classification
- Propagating node signals via (a modified version of) the adjacent matrix
- Producing task-aware node embeddings at the hidden layers with learnable parameters
- Motivated by the Spectral Graph Convolution (based on the eigen-decomposition of the graph Laplacian) and the 1st-order Chebyshev approximation to avoid explicit eigendecompostion
- Making the 1st connection of GCN with Weisfeiler-Lehman (WL) test on graph isomorphism (more discussions with GIN later)

Multi-layer GCN [Kipf and Welling, ICLR 2017]

- **Input layer**

$$H^{(0)} := X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n) \qquad (\boldsymbol{x}_i \text{ is the input feature vector of node } i.)$$

- **Intermediate layers**

$$H^{(l+1)} := \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \quad \text{for } l = 0, 1, \cdots, L-1$$

where $H^{\{l\}} \in R^{n \times d_l}$ is the matrix of node embeddings at layer l;

$W^{(l)} \in R^{d_l \times d_{l+1}}$ are the layer-specific matrix of learnable parameters;

$\tilde{A} = A + I_n$ is the adjacency matrix of the graph with added self-connections (I_n);

$\tilde{D} = \text{diag}\left(\tilde{D}_{ii}\right)_{i=1,\dots,n}$ with $\tilde{D}_{ii} = \sum_{j=1}^{n} \tilde{A}_{ij}$, aggregating the degree information of nodes.

GCN vs. GAT at an Intermediate Layer

- **GCN** $\quad H^{(l+1)} := \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$
  - The red part does not have learnable model parameters (task-agnostic).
- **GAT** $\quad \vec{h}_i^{(l+1)} := \Big\Vert_{m=1}^{M} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(m,l)} W^{(m,l)} \vec{h}_j^{(l)}\right)$ (slide #9)

$$H^{(l+1)} := \Big\Vert_{m=1}^{M} \sigma\left(\mathcal{A}_G^{(m,l)} H^{(l)} W^{(m,l)}\right) \qquad \text{(compact formula)}$$

  - $\mathcal{A}_G^{(m,l)}$ is the graph-masked attention matrix (based on learnable node embeddings).

Recap of Attention in GAT (slide #8)

- Attention between two nodes

$e_{ij} = \theta^T[W\vec{h}_i \parallel W\vec{h}_j]$ " ||" is the concatenation operator;

$\alpha_{ij} = \frac{exp(LeakReLU(e_{ij}))}{\sum_{j' \in \mathcal{N}_i} exp(LeakReLU(e_{ij'}))}$ $\mathcal{N}_i$ is the neighborhood of node i;

$W \in R^{d \times d'}$ and $\theta \in R^{2d'}$ are trainable model parameters.

Design Choices of Signal Passing Mechanism

- Multi-layer Perceptron (MLP) $H^{(l+1)} := \sigma(H^{(l)}W^{(l)})$ (1)
- Signal passing via $A_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ $H^{(l+1)} := \sigma(A_{sym}H^{(l)}W^{(l)})$ (2)
- Signal passing via $\tilde{A} = I_n + A_{sym}$ $H^{(l+1)} := \sigma(\tilde{A}H^{(l)}W^{(l)})$ (3)
- "Renormalization trick" $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ $H^{(l+1)} := \sigma(\hat{A}H^{(l)}W^{(l)})$ (4) Empirical Winner
- Signal passing via graph Laplacian $L_{sym} = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^T$

$H^{(l+1)} := \sigma(Ug_\theta(\Lambda)U^T H^{(l)})$ (5)

- Chebyshev filter (K-th order polynomial approximation of formula 5)

$H^{(l+1)} := \sigma\left(\sum_{k=0}^{K} T_k(\tilde{L})H^{(l)}W_k^{(l)}\right)$ for $K = 0,1,2,3$ (6)

Theoretically & Technically Motivated (for "spectral" analysis)

GCN Model Training for Node Classification

(on Multi-class Datasets Cora, Citeseer, PubMed: one label per document)

- Labeled training data $\mathcal{D}_l = \{(X_i, Y_i)\}$
  - $X_i \in \mathbb{R}^d$ is the feature vector of node i (a document);
  - $Y_i \in \mathbb{R}^K$ is a 0-1 valued vector for a corresponding category label.
- Output $\hat{Y} \in \mathbb{R}^{n \times K}$ (n is the number of nodes in the graph)

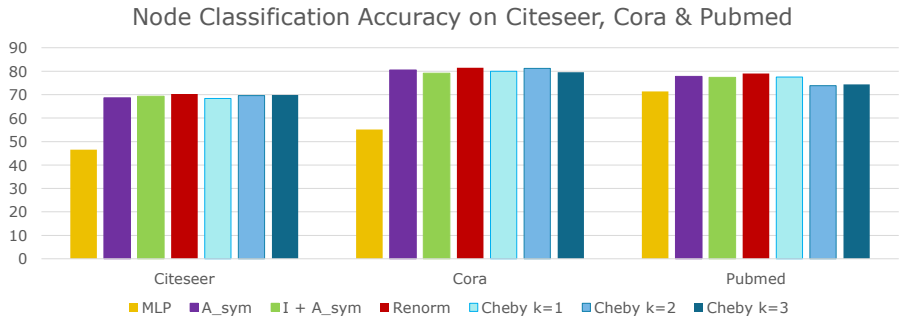
$$\hat{Y} = f(X, A) = softmax\big(\hat{A}\, ReLU\big(\hat{A}\, \mathrm{X} W^{(0)}\big) W^{(1)}\big)$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A}\, \tilde{D}^{-\frac{1}{2}}$ and $\tilde{A} = A_{sym} + I_n$ (adjacency matrix with a re-normalization trick)

- Loss function (cross-entropy) for multi-class problem

$$\mathcal{L} = -\sum_{Y_i \in \mathcal{D}_l} \sum_{j=1}^{K} Y_{ij}\, ln \hat{Y}_{ij}$$

Ablation Tests [Kipf and Welling, ICLR 2017]



- Graph-based signal passing is much better than MLP (no signal passing).
- Using adjacency matrix (or modified versions) is at least as good as using Chebyshev filters.

Comparative Eval Results [Kipf and Welling, ICLR 2017]

Results are summarized in Table 2. Reported numbers denote classification accuracy in percent. For ICA, we report the mean accuracy of 100 runs with random node orderings. Results for all other baseline methods are taken from the Planetoid paper (Yang et al., 2016). Planetoid* denotes the best model for the respective dataset out of the variants presented in their paper.

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

DeepWalk + LR(One vs. All)

Relative performance in node classification

- $GNN\ (GAT\ or\ GCN) \gg DeepWalk \gg MLP$ (Why?)
- $GAT > GCN$ (Why?)

	Using node-specific features?	Using graph structure?	Link weights learnable from labeled data?
DeepWalk+LR	×	✓	
MLP	✓	×	
GCN	✓	✓	
GAT	✓	✓	

Appendix p1. Graph Laplacian and its Spectrum

- Denoting signal $x \in \mathbb{R}^n$ and filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^n$ in the Fourier domain, the spectral convolutions on a graph is defined as

$$g_\theta * x = U g_\theta U^T x$$

where U is the matrix of eigenvectors of graph Laplacian $L_{sym} = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$;

Λ is the diagonal matrix of the eigenvalues (the spectrum) sorted in a descending order;

$U^T x$ is considered as being the Fourier transform of x (projecting it onto the eigenvectors);

$g_\theta = g_\theta(\Lambda)$ is a function of the eigenvalues of L, as the convolution filter in the eigen space;

$y = g_\theta(U^T x)$ is the feature map obtained by applying filter g_θ to $U^T x$ in the Fourier domain;

$U g_\theta U^T x = Uy$ is the inverse Fourier transform of y back to the original space of x;

$\theta \in \mathbb{R}^n$ are the trainable model parameters by a Graph Convolutional Network (GCN).

Appendix p2. Computational Cost of Graph Convolution

- Directly computing spectural convolution takes $O(n^2)$ time

$$g_\theta * x = U g_\theta U^T x$$

- Furthermore, solving the eigendecompostion of L is prohibitively expensive for large graphs.
- Remedy: approximating $g_\theta(\Lambda)$ with Chebyshev filter of the K'th order as

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k \, \mathrm{T}_k(\tilde{\Lambda}) \quad \text{with} \quad \tilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I_n \qquad (1)$$

- With $K = 1, 2 \; and \; 3,$ the Chebyshev filters are much more efficient to compute (slide #28).

Appendix p3. Chebyshev Polynomials with 1-D input

- Recursive Definition

$$T_k(v) = 2vT_{k-1}(v) - T_{k-2}(v) \text{ for } v \in R$$

with $T_0(v) = 1;\ T_1(v) = v;$

- Calculation by induction

$$T_2(v) = 2v^2 - 1;$$

$$T_3(v) = 4v^3 - 2v - v = 4v^3 - 3v$$

$$\vdots$$

  - Resulted $\sum_{k=0}^{K} \theta_k T_k(v)$ is a weighted combination of v^k for $k = 0,1,\cdots,K$.
  - Similarly, $Z_\theta(\Lambda) := \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$ is a weighted combination of $\tilde{\Lambda}^k$s (diagonal matrices).

Appendix p4. Chebyshev Polynomials with $\tilde{\Lambda}$ input

- Chebyshev polynomials of the K^{th}order ($K < n$)

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k \, \mathrm{T_k}(\tilde{\Lambda}) \text{ with } \tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_n$$

with $T_0(\tilde{\Lambda}) = I_n;$

$$T_1(\tilde{\Lambda}) = \tilde{\Lambda};$$

$$\vdots$$

$$T_k(\tilde{\Lambda}) = 2\tilde{\Lambda}T_{k-1}(\tilde{\Lambda}) - T_{k-2}(\tilde{\Lambda})$$

where $\theta' = (\theta'_0, \theta'_1, \ldots, \theta'_K)$ are the trainable Chebyshev coefficients.

Appendix p5. Approximated Graph Convolution with the Chebyshev Filter

$$g_\theta * x = U g_\theta U^T x \quad (2)$$

$$\approx U \sum_{k=0}^{K} \theta'_k \, \mathrm{T_k}(\tilde{\Lambda}) \, U^T x = \sum_{k=0}^{K} \theta'_k \, U \mathrm{T_k}(\tilde{\Lambda}) \, U^T x \quad (3)$$

$$= \sum_{k=0}^{K} \theta'_k \, \mathrm{T_k}(\underbrace{U \tilde{\Lambda} U^T}_{\tilde{L}}) \, x = \sum_{k=0}^{K} \theta'_k \, \mathrm{T_k}(\tilde{L}) \, x \quad (4)$$

$$\tilde{L} = U \tilde{\Lambda} U^T = U \left(\frac{2}{\lambda_{max}} \Lambda - I_n \right) U^T = \frac{2}{\lambda_{max}} L - I_n$$

- Notice that the eigendecompostion of L is no longer needed (as U and Λ disappeared in 4).
- $\tilde{L}$ can be highly sparse: the number of non-zero elements equals to $|\mathcal{E}|$ (the number of edges in the graph).
- Time complexity in formula 4 is $O(K|\mathcal{E}|)$ with sparse $\tilde{L}$, compared to $O(n^2)$ in formula 2 with dense U.