Graph Convolutional Networks

lecture-7

Course on Graph Neural Networks (Winter Term 21/22)

Prof. Dr. Holger Giese (holger.giese@hpi.uni-potsdam.de)

Christian Medeiros Adriano (christian.adriano@hpi.de) - **"Chris"**

Matthias Barkowski (matthias.barkowski@hpi.de)

Quick recap

- Metrics
- Classification
- Random Walk
- Node and Graphs Embeddings
- Graph Structure Learning

Why do we need Graph Neural Networks?
We need a way to scale without having to make large random walks in the graph

Topics

Example from Industry

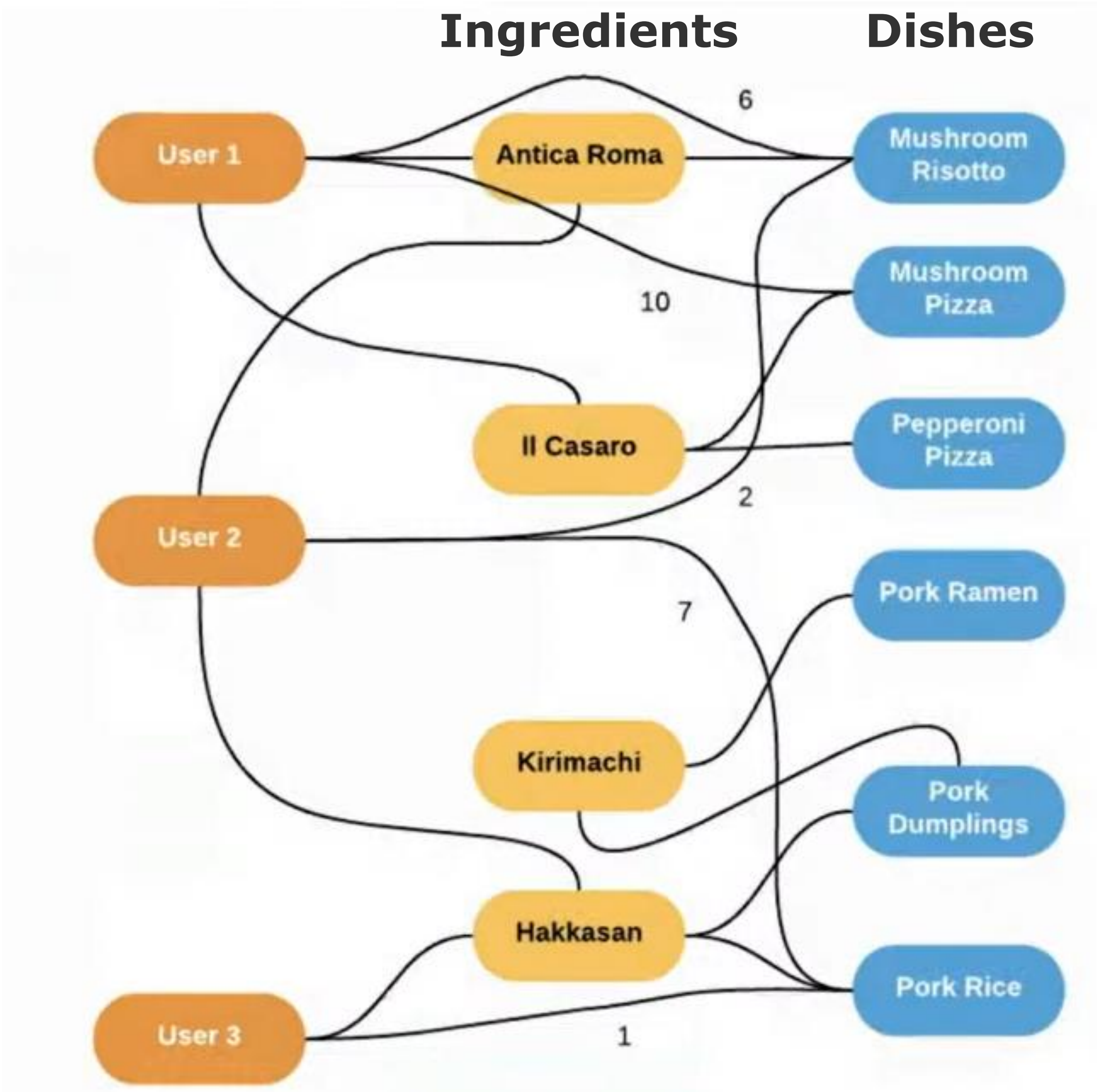
Graph Convolutional Networks (GCN)

Relational GCN

Mathematical background

- Laplacian
- Chebshev Approximation
- Softmax

Motivation – Uber Eats



Offline evaluation

Trained the downstream Personalized Ranking Model using graph node embeddings

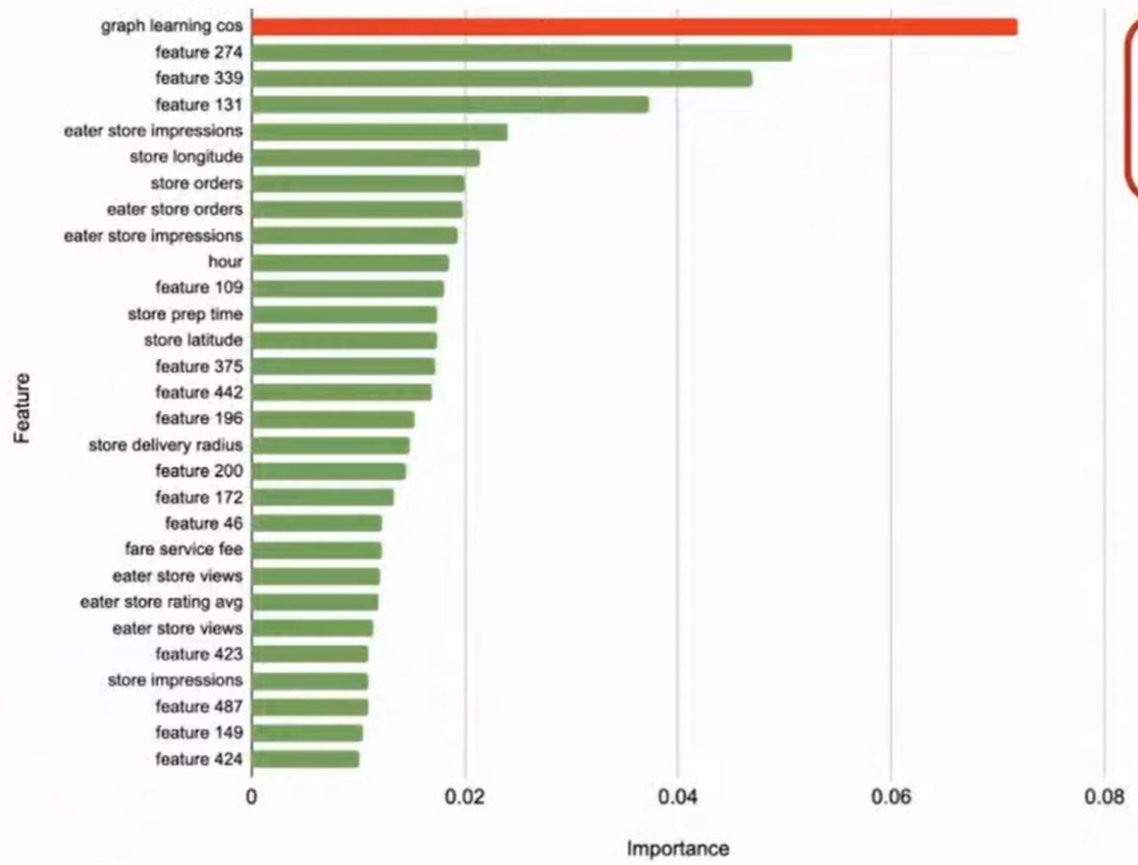
~12% improvement in test AUC over previous production model

Model	Test AUC
Previous production model	0.784
With graph embeddings	**0.877**

https://eng.uber.com/uber-eats-graph-learning/

Outcomes

Feature Importance



Graph learning cosine similarity is the top feature in the model

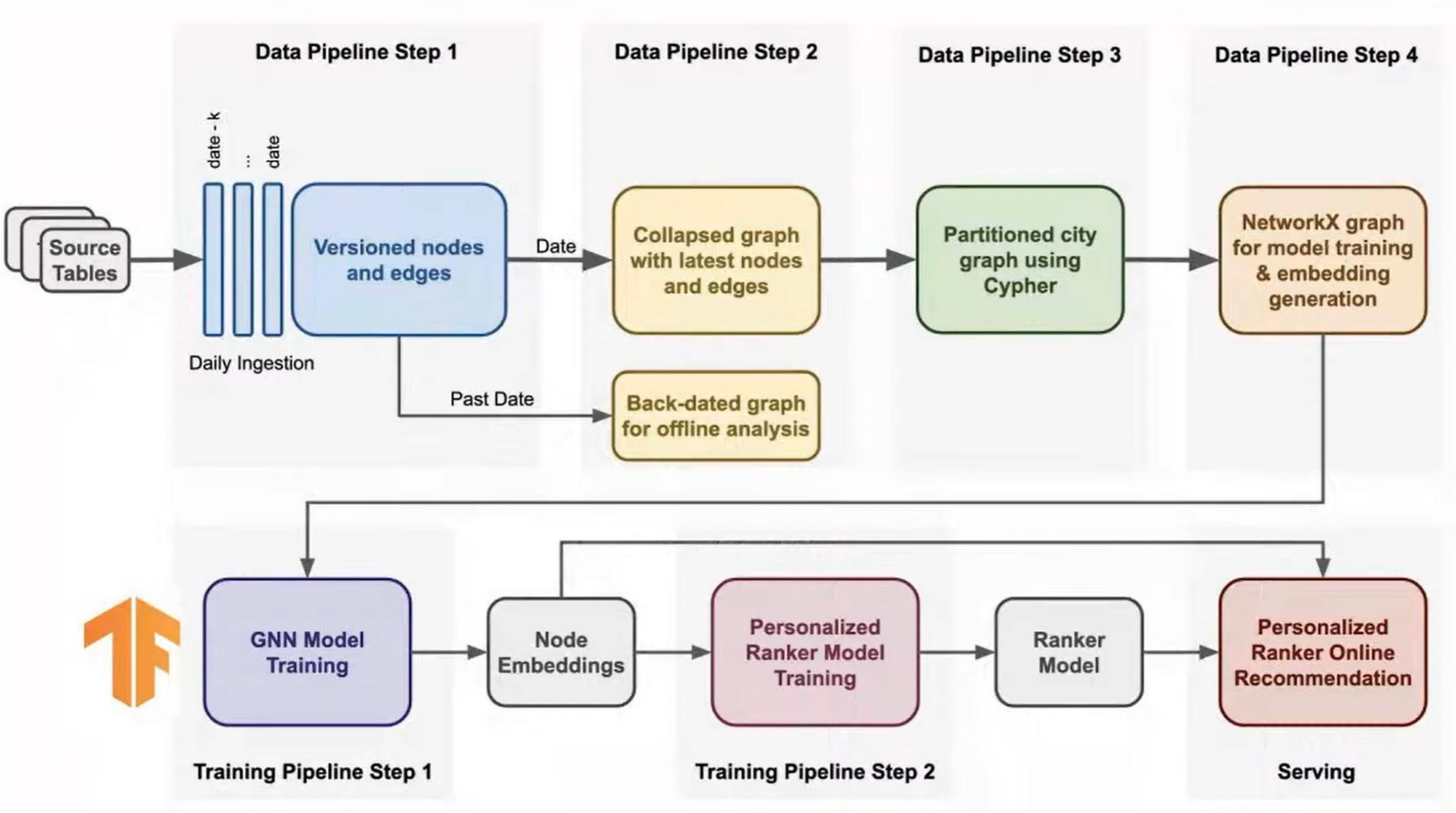
Online evaluation

Ran a A/B test of the Recommended Dishes Carousel in San Francisco

Significant uplift in Click-Through Rate with respect to the previous production model

Conclusion: Dish Recommendations with graph learning features are live in San Francisco, soon everywhere else

Pipeline



Data Pipeline Step 1
Data Pipeline Step 2
Data Pipeline Step 3
Data Pipeline Step 4
date - k
...
date
Source Tables
Versioned nodes and edges
Daily Ingestion
Date
Past Date
Collapsed graph with latest nodes and edges
Back-dated graph for offline analysis
Partitioned city graph using Cypher
NetworkX graph for model training & embedding generation
GNN Model Training
Node Embeddings
Personalized Ranker Model Training
Ranker Model
Personalized Ranker Online Recommendation
Training Pipeline Step 1
Training Pipeline Step 2
Serving

Bipartite Graph for Dish Recommendation

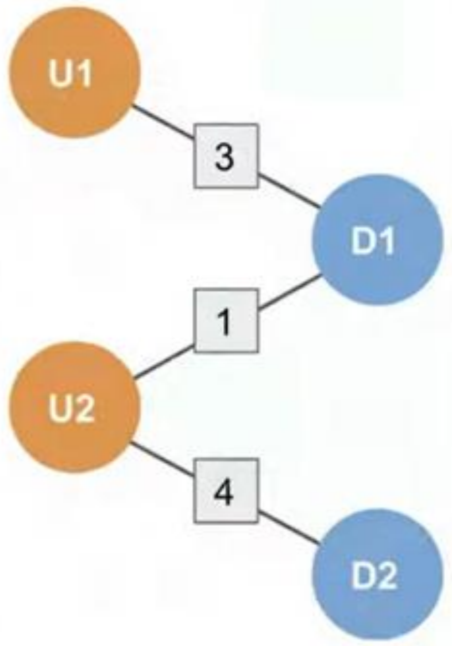
U_i users

D_i Dishes

Users connect to dishes by number of times they order a dish D_i

New users and new dishes are added every day

Each node has a feature (e.g., word2vec)



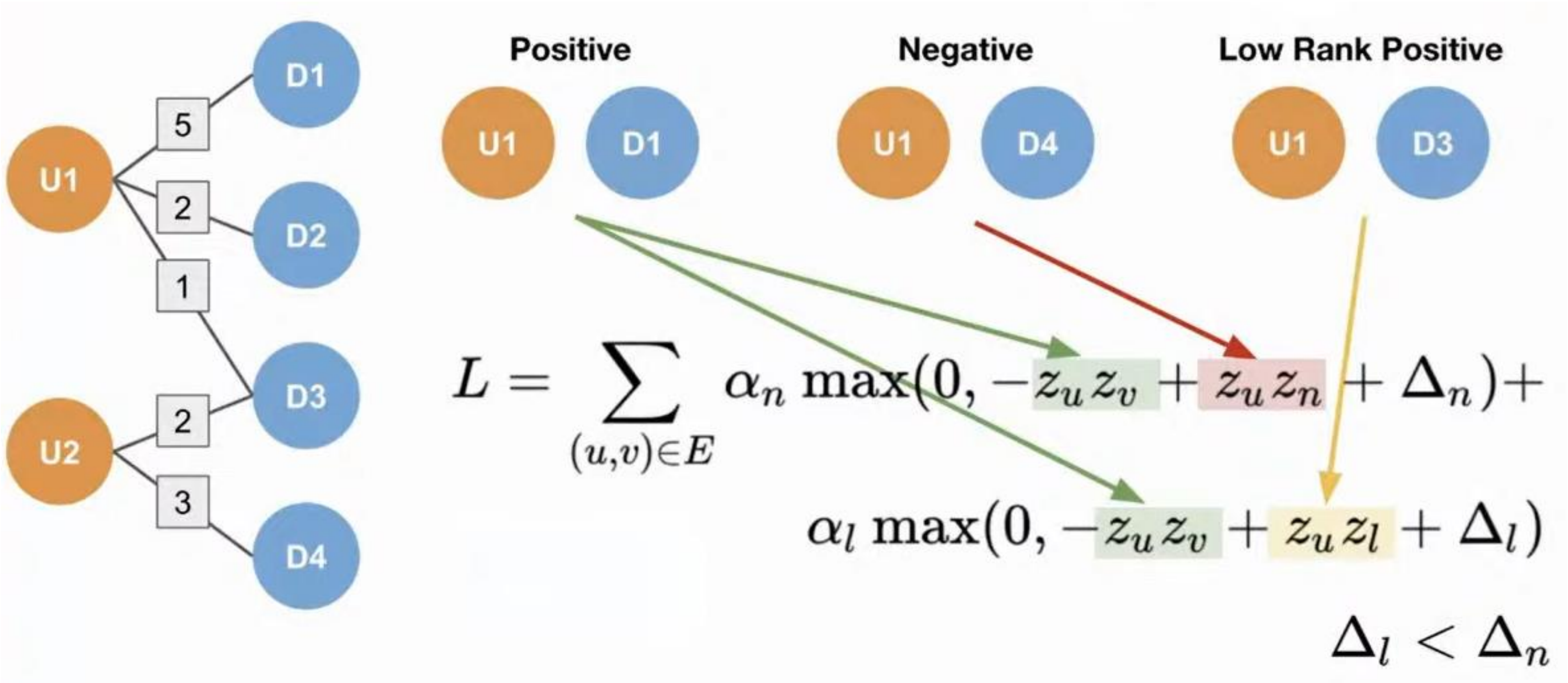
Margin Loss

For dish recommendation, the goal is to rank dishes for each user.

$$L = \sum_{(u,v)\in E} \max(0, -\underbrace{z_u z_v}_{\text{positive pair}} + \underbrace{z_u z_n}_{\text{negative sample}} + \underbrace{\Delta}_{\text{margin}})$$

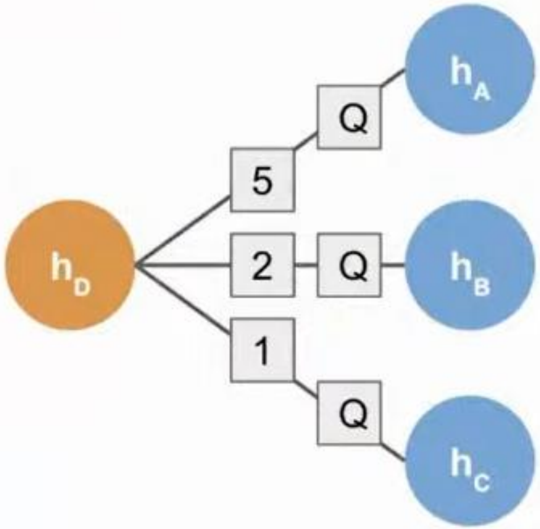
Goal is for the positive pair to surpass the negative pair

Loss with Low Rank Positives



Weighted pool aggregation

Aggregate neighborhood based on the edge weights

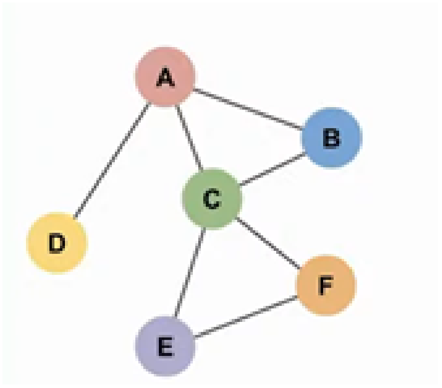


$$\mathbf{AGG} = \sum_{u \in N(v)} w(u,v) Q h_u^{k-1}$$

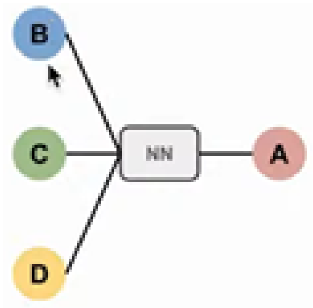
GNN key idea

Obtain a node representation by using a neural network to aggregate the information from the neighbors.

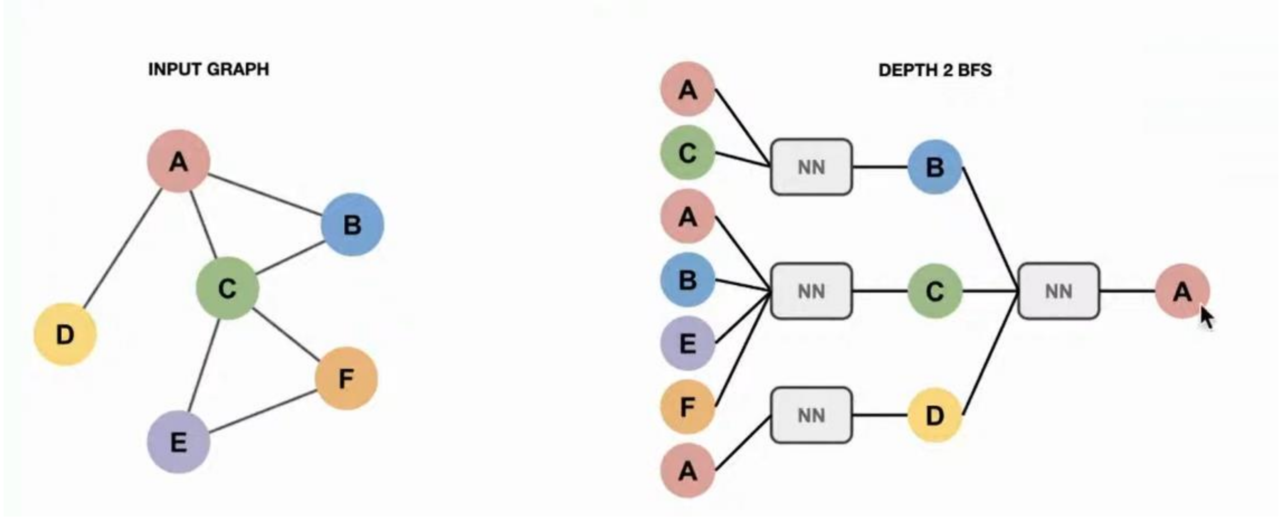
Input graph



Output graph
(depth =2 BFS)



Do it recursively



Solution for Recursion = Message Passing

Every node has its own embedding at every layer

Each layer is how deep we go in the network

Input feature x_u has layer K embeddings h_u^k

Layer-0 has the features of each node

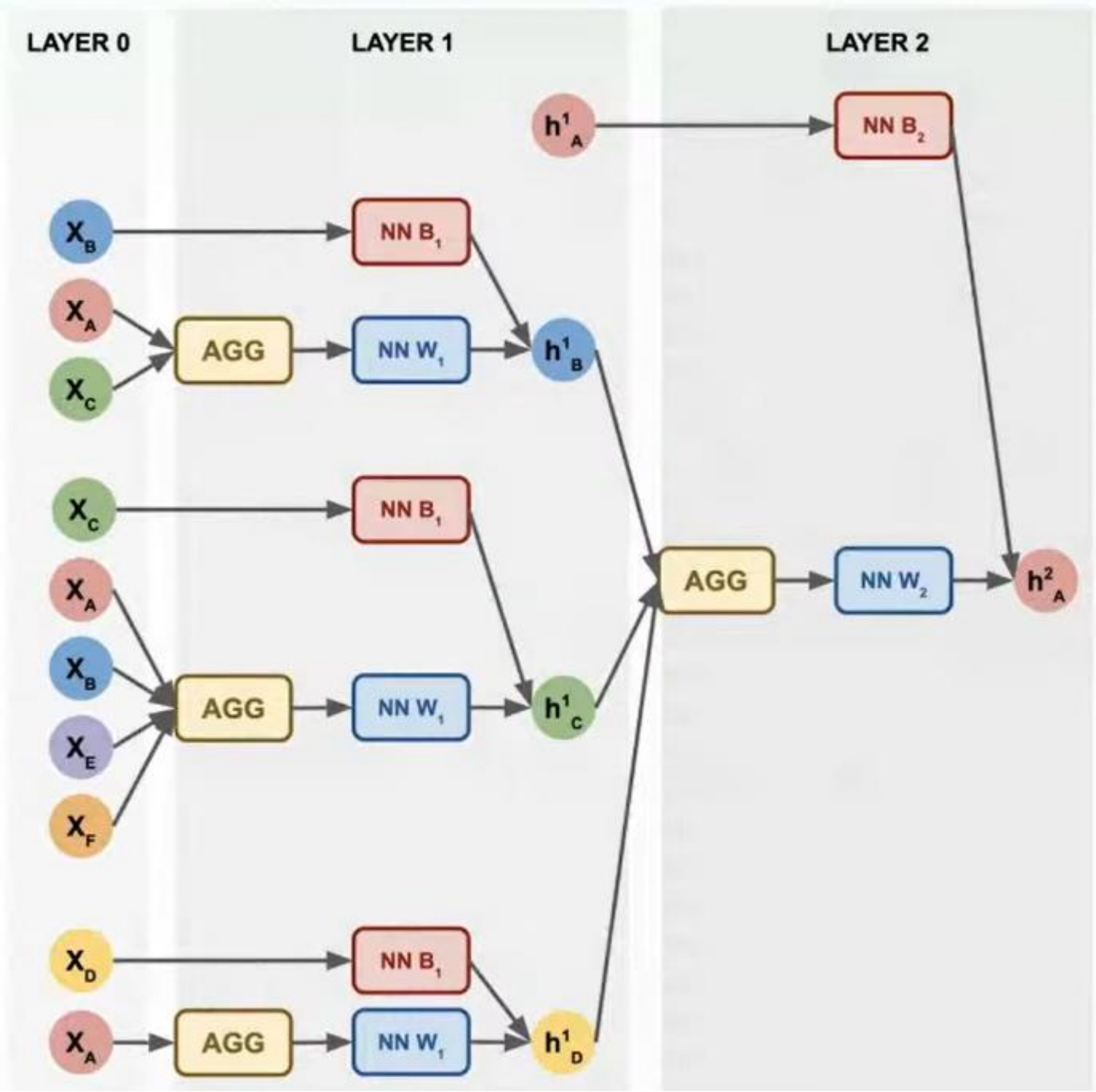
AGG = aggregation function

$NN\ B_k$ = network self-embedding embedding

$NN\ W_1$ = network 1

$NN\ W_2$ = network 2

h_u^k = computed representations



Scalability

Message Passing in GNN allows to scale because:

1. We do not depend on the number of nodes in graph, but the number of neurons that we decide to use to compute de embeddings at each layer.

2. We can limit the number of neurons because all these nodes share the same network at each layer

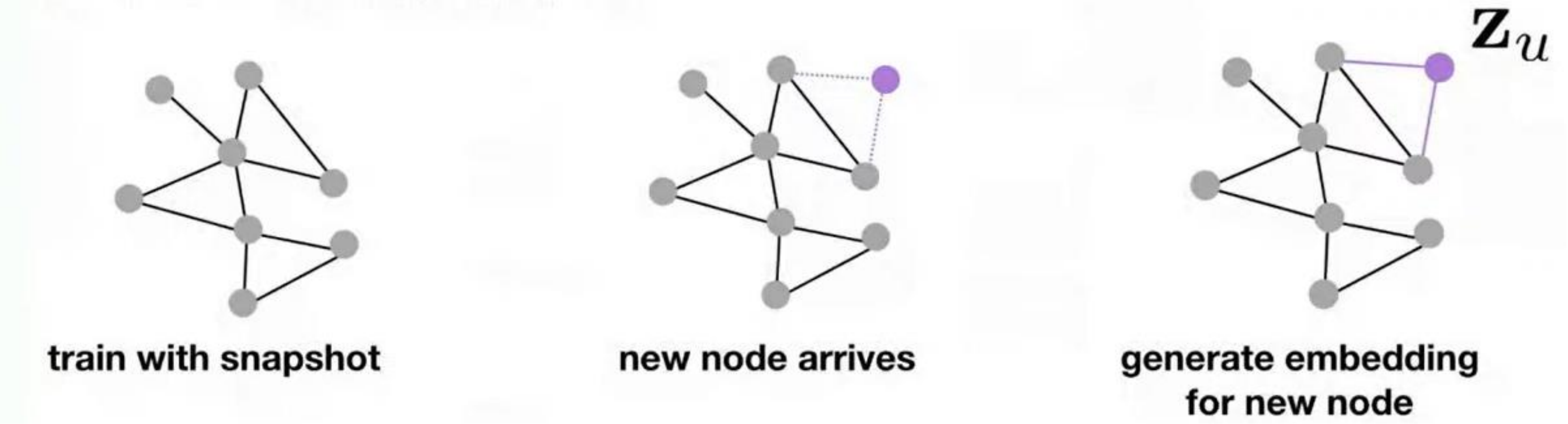
3. This is the scalability that we do not have in the standard methods like Node2Vec, for instance.

Inductive capability

In many applications new nodes are constantly added to the graph

This would require constant retraining to update the embeddings

While this is difficult to do with traditional methods, GNN allows to retrain on the part affected



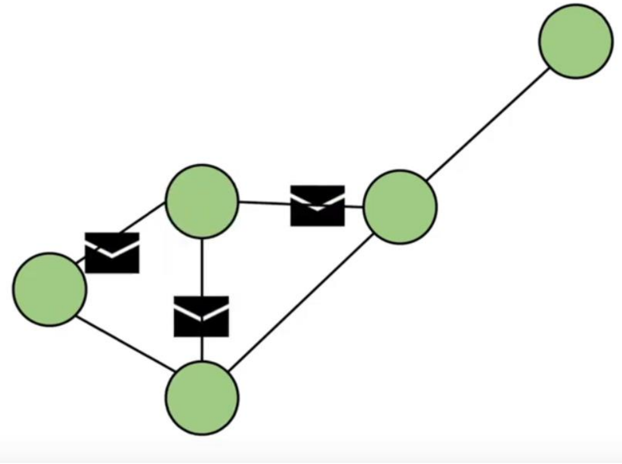
Number of Layers in a GCN

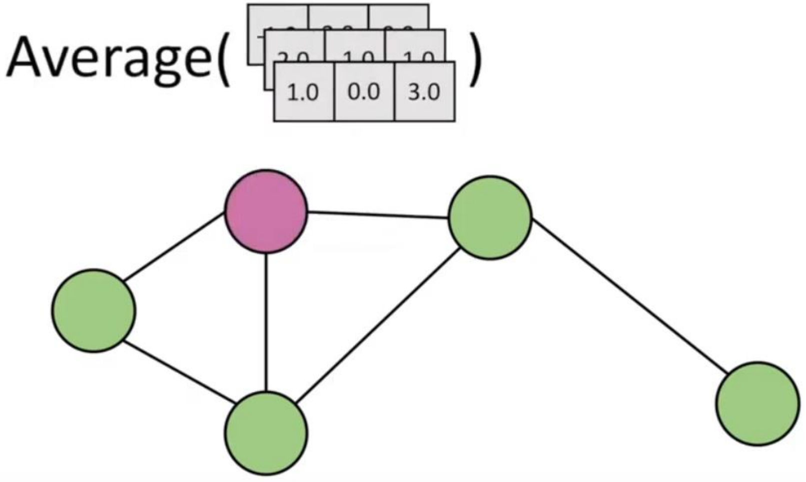
- Each node can have one or multiple dimensions (depends on the features that we choose)
- In a 2-layer GCN, each node pull the information from their neighbors two times.
- The number of GCN layer tells how far the signal could travel.
  - In a 2-layer GCN, the signal travels to a maximum to a two hops from the node.

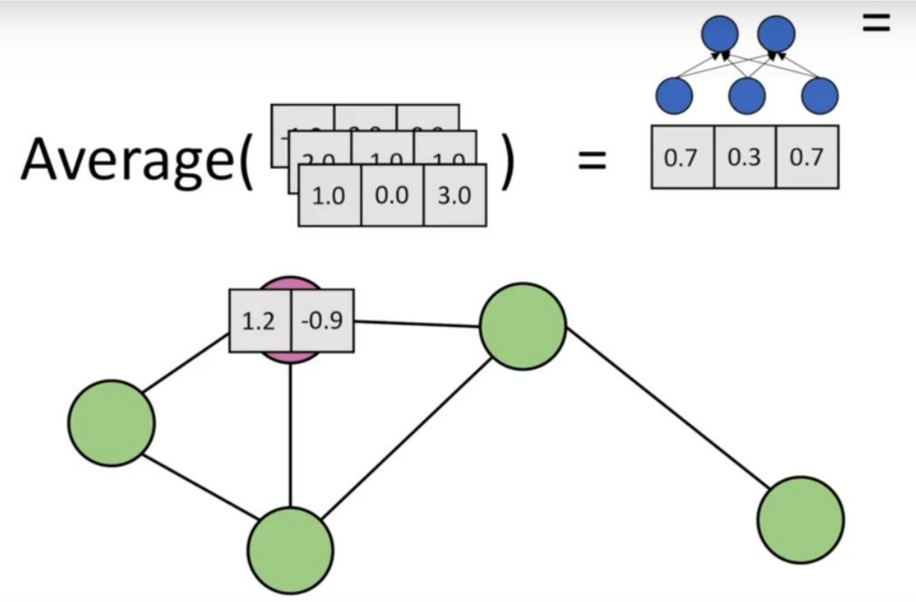
Steps

1. Message propagation
2. Feature aggregation
3. Enconding

Label Propagation	GCN
= Label	= Input Features
= (fraud)	= US VN IR
label smoothing	*feature smoothing*







source: Zack Jost, Short Lectures -[Intro to Relational - Graph Convolutional Networks]

Relational GCN

$$h_i^{l+1} = \sigma\left(\sum_{j \in N_i} \frac{1}{c_{ij}} h_j^l W^l\right)$$

, where:

l is a layer

σ is the sigmoid function (adds the non-linearity)

N_i are the neighbors of node i

c_{ij} is normalizing constant

h^l is the value of the node in the previous layer

W^l is a linear projection matrix for layer l

Quick Math background based on Message Passing

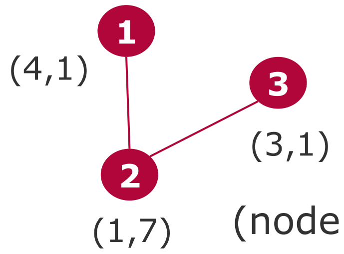
Sum Operation

$$aggregate(A_i, X)_i = A_i\, X = \sum_{j \in N} A_{i,j} X_j$$

This allow to compute the aggregate feature representation of the i_{th} node as vector-matrix product. Computationally, this weighted sum is obtained by using the Adjacency matrix to adjust or weight the features of a node by its he connectivity (edges or neighbors).

The contribution of the j_{th} node in the feature aggregation depends on value of the j_{th} column of the i_{th} row of A. Because A is the adjacency matrix, if this value is 1, then the j_{th} node is a neighbor of the i_{th} node.

$$\underset{\boldsymbol{A}}{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}} \underset{\boldsymbol{X}}{\begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}} = \begin{bmatrix} (0)(3) + (\mathbf{1})(\mathbf{1}) + (0)(4) \\ (1)(3) + (0)(1) + (\mathbf{1})(\mathbf{4}) \\ (0)(3) + (\mathbf{1})(\mathbf{1}) + (0)(4) \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 1 \end{bmatrix}$$



(node feature, aggregated feature)

Hence, the contribution of each neighbor depends only on the connectivity specified in the adjacency matrix ***A***

Average Operation

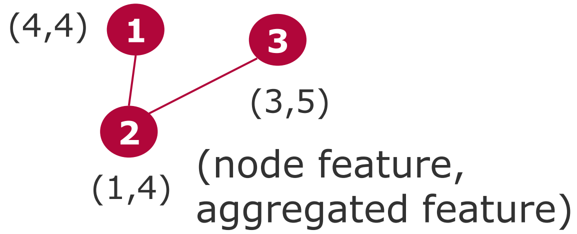
$$aggregate(A,X)_i = D^{-1}A_i\, X = \sum_{k\in N} D^{-1}_{i,k} = \sum_{j\in N} A_{i,j}X_j = \sum_{j\in N} \frac{A_{i,j}}{D_{i,i}} X_j$$

Where D is a diagonal matrix called the degree matrix, which has in its diagonal the degree of each node and zero in all other positions outside the diagonal. The inverse of D is still a diagonal matrix. This allow to remove first summation over $D_{i,k}$ and use $D_{i,i}$.

When we divide the Adjacency matrix A by the degree matrix D, we obtain an average of the connections for each node, for example, if a node *i* has two connections, then it will get 0.5 in for each of the edges (or position in the A matrix). Hence, now we are multiplying the features by this averaged connection. **The consequence** is that neighbor nodes contribute individually (larger weight) to the aggregated feature of less connected nodes than of higher connect ones.

If one wants to include the features of the node itself, one can simply add 1 to the diagonal of the adjacency matrix A. Computationally this can be achieved by adding the identity matrix to the adjacency matrix, i.e., $\tilde{A} = A + I$

$$\overset{\boldsymbol{D^{-1}}}{\begin{bmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{bmatrix}}\left(\overset{\boldsymbol{A}}{\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}} + \overset{\boldsymbol{I}}{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}}\right)\overset{\boldsymbol{X}}{\begin{bmatrix}3\\ 1\\ 4\end{bmatrix}} = \begin{bmatrix}(1)(3)+(1)(1)+(0)(4)\\ (1/2)(3)+(1/2)(1)+(1/2)(4)\\ (0)(3)+(1)(1)+(1)(4)\end{bmatrix} = \begin{bmatrix}4\\ 4\\ 5\end{bmatrix}$$



Spectral Operation

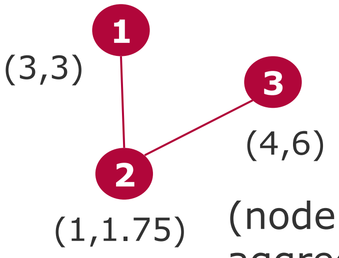
$$aggregate(A,X)_i = D^{-0.5}\tilde{A}\,D^{-0.5}X = \sum_{j\in N}\frac{1}{D_{i,i}^{0.5}}\tilde{A}_{i,j}\ \frac{1}{D_{i,i}^{0.5}}X_j =$$

This normalizes aggregated features so they remain roughly on the same scale as the input features. Another way to write is like the following:

$$aggregate(A,X)_i = \frac{1}{\sqrt[2]{d_i d_j}}\tilde{A}_{i,j}$$

However, the spectral operation weighs neighbor in the weighted sum higher if they have a low-degree and vice versa. This is useful when low-degree neighbors carry more information than their high-degree counterparts.

$$\begin{array}{cccc} \boldsymbol{D^{-0.5}} & \boldsymbol{\tilde{A}} & \boldsymbol{D^{-0.5}} & \boldsymbol{X} \\ \begin{bmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{bmatrix} & \begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 1\end{bmatrix} & \begin{bmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{bmatrix} & \begin{bmatrix}3\\ 1\\ 4\end{bmatrix} \end{array} = \begin{bmatrix}(1)(3)+(0)(1)+(0)(4)\\ (1/2)(3)+(1/4)(1)+(0)(4)\\ (1/2)(3)+(1/2)(1)+(1)(4)\end{bmatrix} = \begin{bmatrix}3\\ 1.75\\ 6\end{bmatrix}$$



(node feature, aggregated feature)

Hence, the contribution of each neighbor depends only on the connectivity specified in the adjacency matrix ***A***

Message Passing with a Transition Matrix

Summing the weights of my neighbors

$\vec{X}$ is the feature vector, where each element *i* is a feature of a node *i*

A is the adjacency matrix

$H = A * \vec{X}$

Average over the weights of my neighbors

D is the node degree matrix, where $D_{ii} = \sum_j A_{ij}$,

$\tilde{A} = D^{-1} * A$, D is the degree matrix

Kipf, Thomas N., and Max Welling. (2017) "Semi-supervised classification with graph convolutional networks." ICRL, *arXiv preprint arXiv:1609.02907*

Message Passing – Scale by Target Node [Kipf & Welling 2017]

Scaling by the target node

$\tilde{A} = A * I$, I is the identify matrix, $\tilde{A}$ adds the self-loops to the A

$$\hat{A} = \widetilde{D}^{-1/2} * \tilde{A} * \widetilde{D}^{-1/2}$$

$\hat{A}_{i,j} = \frac{1}{\sqrt[2]{\widetilde{d_i}\widetilde{d_j}}} * \widetilde{A_{ij}}$ scaling by the neighborhood sizes of source and root nodes

This allows to have an adjacency matrix that if multiplied various times the computations will be stable, i.e., it will converge to a stationary value.

Kipf, Thomas N., and Max Welling. (2017) "Semi-supervised classification with graph convolutional networks." ICRL, *arXiv preprint arXiv:1609.02907*

Message Passing – Simulation

https://github.com/christianadriano/blog_code/blob/master/gcn_numpy/message_passing.ipynb

Fast Approximate Convolutions on Graphs [Kipf & Welling 2017]

$$H^{l+1} = \sigma(\widetilde{D}^{-\frac{1}{2}} * \tilde{A} * \widetilde{D}^{-\frac{1}{2}} * H^{l} * W^{l})$$

, where:

W^{l} is a layer-specific trainable weight matrix.

σ is an activation function, such as the ReLU() = max(0;) or Sigmoid.

$H^{l} \in \mathbb{R}^{N \times D}$ is the matrix of activations in layer l

$H^{0} = \vec{X}$, where X is the feature vector

Kipf, Thomas N., and Max Welling. (2017) "Semi-supervised classification with graph convolutional networks." ICRL, *arXiv preprint arXiv:1609.02907*

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Spectral Graph Convolution as the multiplication of a signal x by a filter g_θ

$g_\theta x = U g_\theta U^T x$, where:

U is the matrix of eigenvectors of the normalized graph Laplacian L

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

g_θ is a function of the eigenvalues of the Laplacian, so $g_\theta(\Lambda)$

<u>However, computing this does not scale</u>:

- Multiplication of U is $O(N^2)$ and
- Eigen-decomposition of L is too expensive for large graphs

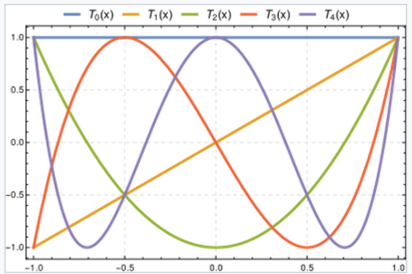
Solution is to approximate $g_\theta(\Lambda)$ with the Chebyshev polynomials

Chebyshev series is related to a Fourier cosine series (period $2L$)

$$f(x) = \frac{c_0}{2} + \sum_{n=1}^{\infty} c_n \cos \frac{n\pi x}{L} \qquad c_n = \frac{2}{L} \int_0^L f(x) \cos \frac{n\pi x}{L}\, dx, n \in \mathbb{N}_0.$$

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x)\,.$$

$$\sum_{n=0}^{\infty} T_n(x) t^n = \frac{1 - tx}{1 - 2tx + t^2}\,.$$



Plot of the first five T_n Chebyshev polynomials of the first kind

Wikipedia

<u>source:</u> Kipf, T. N. & Welling, M., "Semi-supervised classification with graph convolutional networks." ICRL, 2017

Approximation with Chebyshev Polynomials [Hammond 2011]

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda})$$

$$\tilde{\Lambda} = \frac{2}{\lambda_{\max}} \Lambda - I_N$$

, where:

λ_{max} is the largest eigenvalue of the Laplacian

$\theta'$is the Chebyshev coefficients

Chebyshev polynomial is recursively defined by

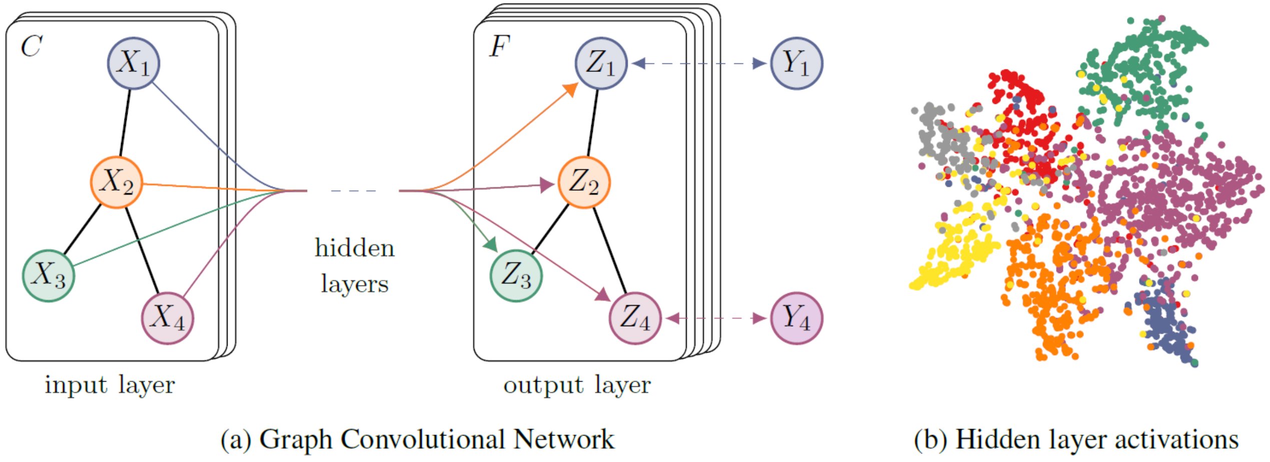
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), where\ T_o(x) = 1, T_1(x) = x$$

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x$$

Hammond, D. et al., (2011) Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2):129–150

Forward computation in the GNN [Kipf & Welling 2017]

C input channels
F maps in the output layer
Y_i label of node X_i



$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A} X W^{(0)}\Big)\, W^{(1)}\Big) .$$

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$$

$$\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} ,$$

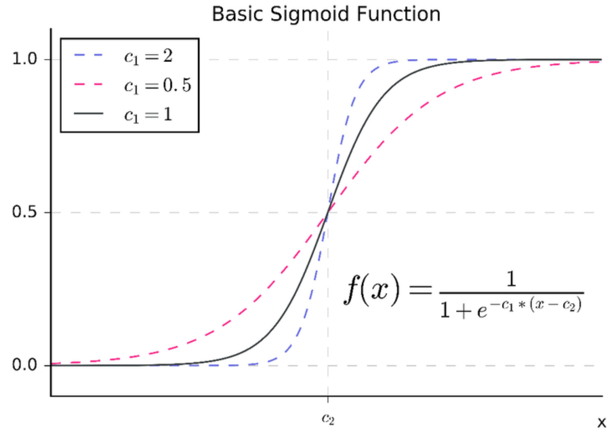
Loss function is the cross-entropy, where Y_{lf} are the nodes with labels and

Kipf, Thomas N., and Max Welling. (2017) "Semi-supervised classification with graph convolutional networks." ICRL, *arXiv preprint arXiv:1609.02907*
Code to reproduce: https://github.com/tkipf/gcn

Softmax [1]

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^\top x)}$$

where, $x \in \mathbb{R}$ features and θ parameters to optimize



$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}$$

where, y are the classes of x

Cost function J

$$J(\theta) = -\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

Gradient of J

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta)\right)\right]$$

[1] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

END