Graph Machine Learning

I. Makarov

University of Ljubljana

Introduction to Network Analysis

Lecture outline

Lecture topic

Graph Machine Learning

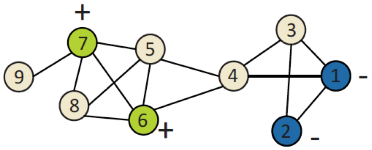
Graph machine learning

- Node classification (attribute inference)
- Link prediction (missing/hidden links inference)
- Community detection (clustering nodes in graph)
- Graph visualization (cluster projections)

Node classification

Node classification

- Node classification - labeling of all nodes in a graph structure
- Subset of nodes is labeled: categorical/numeric/binary values
- Extend labeling to all nodes on the graph (class/class probability/regression)
- Classification in networked data, network classification, structured inference, relational learning



Node classification

- Structure can help only if labels/values of linked nodes are correlated
- Social networks show assortative mixing - bias in favor of connections between network nodes with similar characteristics:
  – homophily: similar characteristics $\rightarrow$ connections
  – influence: connections $\rightarrow$ similar characteristics
- Can apply to constructed (induced) similarity networks
- Node classification by label propagation

Node classification

Supervised learning approach

- Given graph nodes $V = V_l \cup V_u$:
  – nodes V_l given labels Y_l
  – nodes V_u do not have labels
- Need to find Y_u
- Labels can be binary, multi-class, real values
- Features (attributes) can be computed for every node ϕ_i:
  – local node features (if available)
  – link features available (labels from neighbors, attributes from neighbors, node degrees, connectivity patterns)

Iterative relational classifiers

- Weighted-vote relational neighbor classifier:

$$P(y_i = c|\mathcal{N}_i) = \frac{1}{Z} \sum_{j \in \mathcal{N}_i} A_{ij} P(y_j = c|\mathcal{N}_j)$$

- Network only Naive Bayes classifier:

$$P(y_i = c|\mathcal{N}_i) = \frac{P(\mathcal{N}_i|c)P(c)}{P(\mathcal{N}_i)}$$

where

$$P(\mathcal{N}_i|c) = \frac{1}{Z} \prod_{j \in \mathcal{N}_i} P(y_j = \hat{y}_j|y_i = c)$$

Semi-supervised learning

- Graph-based semi-supervised learning
- Given partially labeled dataset
- Data: $\mathrm{X} = X_l \cup X_u$
  – small set of labeled data (X_l, Y_l)
  – large set of unlabeled data X_u
- Similarity graph over data points $G(V, E)$, where every vertex v_i corresponds to a data point x_i
- Transductive learning: learn a function that predicts labels Y_u for the unlabeled input X_u

Random walk methods

- Consider random walk with absorbing states - labeled nodes V_l
- Probability $\hat{y}_i[c]$ for node $v_i \in V_u$ to have label c,

$$\hat{y}_i[c] = \sum_{j \in V_l} p_{ij}^{\infty} y_j[c]$$

  where $y_i[c]$ - probability distribution over labels,
  $p_{ij} = P(i \to j)$ - one step probability transition matrix
- If output requires single label per node, assign the most probable
- In matrix form

$$\hat{Y} = P^{\infty} Y$$

  where $Y = (Y_l, 0)$, $\hat{Y} = (Y_l, \hat{Y}_u)$

Random walk methods

- Random walk matrix: $P = D^{-1}A$
- Random walk with absorbing states

$$P = \begin{pmatrix} P_{ll} & P_{lu} \\ P_{ul} & P_{uu} \end{pmatrix} = \begin{pmatrix} I & 0 \\ P_{ul} & P_{uu} \end{pmatrix}$$

- At the $t \to \infty$ limit:

$$\lim_{t\to\infty} P^t = \begin{pmatrix} I & 0 \\ \left(\sum_{n=0}^{\infty} P_{uu}^n\right) P_{ul} & P_{uu}^{\infty} \end{pmatrix} = \begin{pmatrix} I & 0 \\ (I - P_{uu})^{-1} P_{ul} & 0 \end{pmatrix}$$

Random walk methods

- Matrix equation

$$\begin{pmatrix} \hat{Y}_l \\ \hat{Y}_u \end{pmatrix} = \begin{pmatrix} I & 0 \\ (I - P_{uu})^{-1} P_{ul} & 0 \end{pmatrix} \begin{pmatrix} Y_l \\ Y_u \end{pmatrix}$$

- Solution

$$\begin{aligned} \hat{Y}_l &= Y_l \\ \hat{Y}_u &= (I - P_{uu})^{-1} P_{ul} Y_l \end{aligned}$$

- $(I - P_{uu})$ is non-singular for all label connected graphs (is always possible to reach a labeled node from any unlabeled node)

Label propagation

```
**Algorithm:** Label propagation, Zhu et. al 2002
**Input:** Graph $G(V, E)$, labels $Y_l$
**Output:** labels $\hat{Y}$
Compute $D_{ii} = \sum_j A_{ij}$
Compute $P = D^{-1}A$
Initialize $Y^{(0)} = (Y_l, 0)$, t=0
repeat
    $Y^{(t+1)} \leftarrow P \cdot Y^{(t)}$
    $Y_l^{(t+1)} \leftarrow Y_l^{(t)}$
until $Y^{(t)}$ converges;
$\hat{Y} \leftarrow Y^{(t)}$
```

Solution: $\hat{Y} = \lim_{t\to\infty} Y^{(t)} = (I - P_{uu})^{-1} P_{ul} Y_l$

Label spreading

```
**Algorithm:** Label spreading, Zhou et. al 2004
**Input:** Graph $G(V, E)$, labels $Y_l$
**Output:** labels $\hat{Y}$
Compute $D_{ii} = \sum_j A_{ij}$ ,
Compute $\mathcal{S} = D^{-1/2} A D^{-1/2}$
Initialize $Y^{(0)} = (Y_l, 0)$, t=0
repeat
    $Y^{(t+1)} \leftarrow \alpha \mathcal{S} Y^{(t)} + (1-\alpha) Y^{(0)}$
    $t \leftarrow t+1$
until $Y^{(t)}$ converges;
```

Solution: $\hat{Y} = (1-\alpha)(I - \alpha\mathcal{S})^{-1} Y^{(0)}$

Node regression

Regression on graphs

Find labeling $\hat{Y} = (\hat{Y}_l, \hat{Y}_u)$ that

- Consistent with initial labeling:

$$\sum_{i \in V_l} (\hat{y}_i - y_i)^2 = ||\hat{Y}_l - Y_l||^2$$

- Consistent with graph structure (regression function smoothness):

$$\frac{1}{2} \sum_{i,j \in V} A_{ij} (\hat{y}_i - \hat{y}_j)^2 = \hat{Y}^T (D - A) \hat{Y} = \hat{Y}^T L \hat{Y}$$

- Stable (additional regularization):

$$\epsilon \sum_{i \in V} \hat{y}_i^2 = \epsilon ||\hat{Y}||^2$$

Regularization on graphs

Minimization with respect to $\hat{Y}$, $\arg\min_{\hat{Y}} Q(\hat{Y})$

- Label propagation [Zhu, 2002]:

$$Q(\hat{Y}) = \frac{1}{2}\sum_{i,j\in V} A_{ij}(\hat{y}_i - \hat{y}_j)^2 = \hat{Y}^T L \hat{Y}, \quad \text{with fixed} \quad \hat{Y}_l = Y_l$$

- Label spread [Zhou, 2003]:

$$Q(\hat{Y}) = \frac{1}{2}\sum_{ij\in V} A_{ij}\left(\frac{\hat{y}_i}{\sqrt{d}_i} - \frac{\hat{y}_j}{\sqrt{d}_j}\right)^2 + \mu \sum_{i\in V}(\hat{y}_i - y_i)^2$$

$$Q(\hat{Y}) = \hat{Y}^T \mathcal{L} \hat{Y} + \mu||\hat{Y} - Y||^2$$

$\mathcal{L} = I - \mathcal{S} = I - D^{-1/2} A D^{-1/2}$

Regularization on graphs

- Laplacian regularization [Belkin, 2003]

$$Q(\hat{Y}) = \frac{1}{2}\sum_{ij \in V} A_{ij}\,(\hat{y}_i - \hat{y}_j)^2 + \mu \sum_{i \in V_l} (\hat{y}_i - y_i)^2$$

$$Q(\hat{Y}) = \hat{Y}^T L \hat{Y} + \mu ||\hat{Y}_l - Y_l||^2$$

- Use eigenvectors $(e_1..e_p)$ from smallest eigenvalues of $L = D - A$:

$$Le_j = \lambda_j e_j$$

- Construct classifier (regression function) on eigenvectors

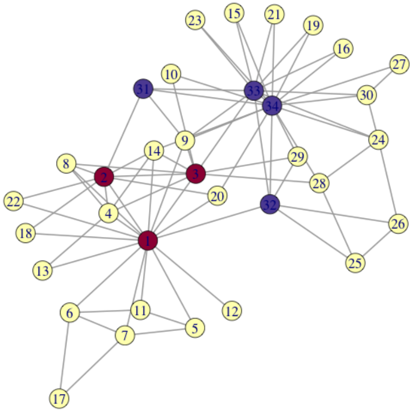
$$Err(a) = \sum_{i \in V_l} (y_i - \sum_{j=1}^{p} a_j e_{ji})^2$$

- Predict value (classify) $\hat{y}_i = \sum_{j=1}^{p} a_j e_{ji}$, class $c_i = sign(\hat{y}_i)$

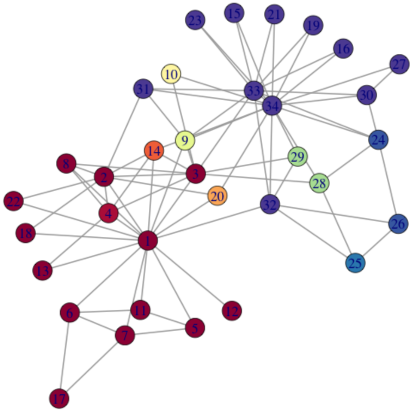
Laplacian regularization

```
Algorithm: Laplacian regularization, Belkin and Niyogy, 2003
Input: Graph $G(V, E)$, labels $Y_l$
Output: labels $\hat{Y}$
Compute $D_{ii} = \sum_j A_{ij}$
Compute $L = D - A$
Compute $p$ eigenvectors $e_1..e_p$ with smallest eigenvalues of $L$, $Le = \lambda e$
Minimize over $a_1...a_p$
arg $\min_{a_1,...a_p} \sum_{i=1}^{l}(y_i - \sum_{j=1}^{p} a_j e_{ji})^2, \quad a = (E^T E)^{-1} E^T Y_l$
Label $v_i$ by the $sign(\sum_{j=1}^{p} a_j e_{ji})$
```

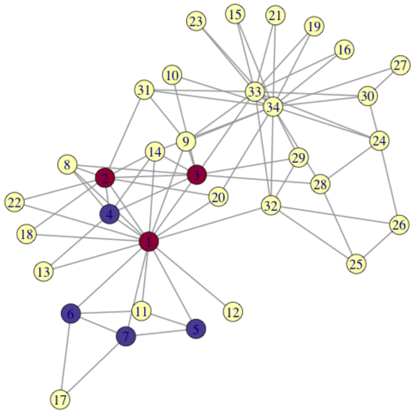
Label propagation example



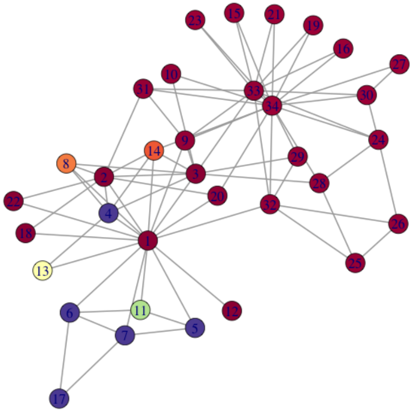
Label propagation example



Label propagation example



Label propagation example



Matrix Factorization

Low-rank approximations

- Low-rank approximation (truncated SVD)

$$A = \sum_{k}^{n} U_k S_k V_k^T \rightarrow \sum_{k}^{r} U_k S_k V_k^T = A', r < n$$

$$\underset{m \times n}{\overset{\hat{X}}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}}} \approx \underset{m \times r}{\overset{U}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}}} \underset{r \times r}{\overset{S}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}}} \underset{r \times n}{\overset{V^{\mathsf{T}}}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}}}$$

Matrix Factorization: Dimension Reduction

The idea of solving node classification lies in decomposing structural and context features from graph for efficient node representation.

- Multidimensional scaling (MDS): Approximating MSE over $A_{ij} - |u_i - u_j|_2^2$
- Indexing by latent semantic analysis (LSI): SVD decomposition of A adjacency matrix
- Dimension reduction for A: PCA (principal components analysis), LDA (linear discriminant analysis), etc.

from Makarov et al., 2021[1]

[1] `https://peerj.com/articles/cs-357/`

Matrix Factorization: Proximity Matrix

Instead of extracting features from A alone, take into account node neighbors in the approximation framework.
A Global Geometric Framework for Nonlinear Dimensionality Reduction (**Isomap**)

- Take graph as an input from some metric learning task, for e.g.
- Compute its k-distance matrix by Floyd-Warshall algorithm.
- Use dimension reduction to extract meaningful components.

Nonlinear Dimensionality Reduction by Locally Linear Embedding (**LLE**)

$$LLE_{error}(W) = MSE(A - W^t U)$$

where U contains neighbors of points from A. In this way, locally, each point is presented as linear combinations of neighbor vector representations.

from Makarov et al., 2021[2]

[2] `https://peerj.com/articles/cs-357/`

Matrix Factorization: Spectral Decomposition

Find eigen-vector decomposition, producing low-dimensional space representation.

Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering (**LE**)

- Take graph as an input from some metric learning task, and allow heat kernels for weights from features F.
- Solve the equation $Lx = \lambda Dx$, $L = D - A$ is Laplacian
- $X = (x_1 \cdots x_n)$, $X^t F$ get a low dimension representation.

The goal for Laplacian Eigenmaps class of models lies in preserving first-order similarities giving a larger penalty using graph Laplacian if two nodes with larger similarity are embedded far apart.

from Makarov et al., 2021[3]

[3] `https://peerj.com/articles/cs-357/`

Matrix Factorization: Spectral Decomposition

Find eigen-vector decomposition, producing low-dimensional space representation.

Locality Preserving Projections (**LPP**)

- Take graph as an input from some metric learning task, and allow heat kernels for weights from features F.
- Solve the equation $FLF^t x = \lambda FDF^t x$, $L = D - A$ is Laplacian
- $X = (x_1 \cdots x_n)$, $X^t F$ get a low dimension representation.

from Makarov et al., 2021[4]

[4] https://peerj.com/articles/cs-357/

Matrix Factorization: Second-order proximities

Continuous nonlinear dimensionality reduction by **Kernel Eigenmaps** present a kernel-based mixture of affine maps from the ambient space to the target space, in which local PCA can be run.

Cauchy Graph Embedding enhance the local topology preserving with the similarity relationships of the original data.

from Makarov et al., 2021[5]

[5] `https://peerj.com/articles/cs-357/`

Matrix Factorization: Second-order proximities

Structure Preserving Embedding (**SPE**) aims to use LE combined with preserving spectral decomposition representing the cluster structure of the graph. SPE is formulated as a semidefinite program that learns a low-rank kernel matrix constrained by a set of linear inequalities which captures the input graph.

Graph Factorization minimize $MSE(A_{ij}, < Z_i, Z_j >)$ with L_2 regularization on 'Z' representations.

from Makarov et al., 2021[6]

[6] `https://peerj.com/articles/cs-357/`

Lecture outline

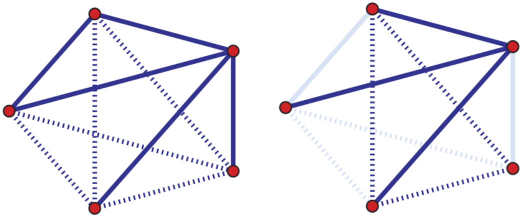
Link Prediction

Link prediction

- **Link prediction**. A network is changing over time. Given a snapshot of a network at time t, predict edges added in the interval (t, t')
- **Link completion** (missing links identification). Given a network, infer links that are consistent with the structure, but missing (find unobserved edges)
- **Link reliability**. Estimate the reliability of given links in the graph.

- Predictions: link existence, link weight, link type

Link prediction



- Graph G(V,E)
- Number of "missing edges": $|V|(|V|-1)/2-|E|$
- In sparse graphs $|E| \ll |V|^2$, Prob. of correct random guess $O(\frac{1}{|V|^2})$

Similarity based algorithms - unsupervised

Link prediction by proximity scoring

1. For each pair of nodes compute proximity (similarity) score $c(v_1, v_2)$
2. Sort all pairs by the decreasing score
3. Select top n pairs (or above some threshold) as new links
4. Quality measurements - precision $TP/(TP + FP)$, precision at top N

Local similarity indices

Local neighborhood of v_i and v_j

- Number of common neighbors:

$$s_{ij} = |\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|$$

- Jaccard's coefficient:

$$s_{ij} = \frac{|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|}{|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)|}$$

- Resource allocation:

$$s_{ij} = \sum_{w \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} \frac{1}{|\mathcal{N}(w)|}$$

Adamic/Adar:

$$s_{ij} = \sum_{w \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} \frac{1}{\log |\mathcal{N}(w)|}$$

Liben-Nowell and Kleinberg, 2003

Local similarity indices

- Preferential attachment:

$$s_{ij} = k_i \cdot k_j = |\mathcal{N}(v_i)| \cdot |\mathcal{N}(v_j)|$$

  or

$$s_{ij} = k_i + k_j = |\mathcal{N}(v_i)| + |\mathcal{N}(v_j)|$$

- Clustering coefficient:

$$s_{ij} = CC(v_i) \cdot CC(v_j)$$

  or

$$s_{ij} = CC(v_i) + CC(v_j)$$

Quasi-Local similarity indices

- Local Path Index:

$$s_{lp} = A^2 + \alpha A^3$$

- High-order LPI:

$$s_{lp(n)} = \sum_{i=2}^{n} \alpha^{i-2} A^i$$

or

$$s_{ij} = CC(v_i) + CC(v_j)$$

Path based methods

Paths and ensembles of paths between v_i and v_j

- Shortest path:

$$s_{ij} = -\min_s \{path_{ij}^s > 0\}$$

- Katz score:

$$s_{ij} = \sum_{s=1}^{\infty} \beta^s |paths^{(s)}(v_i, v_j)| = \sum_{s=1}^{\infty} (\beta A)_{ij}^s = (I - \beta A)^{-1} - I$$

- Personalized (rooted) PageRank:

$$PR = \alpha (D^{-1} A)^T PR + (1 - \alpha) \cdot (e_i + e_j)$$

Liben-Nowell and Kleinberg, 2003

Path based indeces

- Expected number of random walk steps:
  hitting time: $s_{ij} = -H_{ij}$
  commute time $s_{ij} = -(H_{ij} + H_{ji})$
  normalized hitting/commute time $s_{ij} = -(H_{ij}\pi_j + H_{ji}\pi_i)$

- SimRank:

$$SimRank(v_i, v_j) = \frac{C}{|\mathcal{N}(v_i)| \cdot |\mathcal{N}(v_j)|} \sum_{m \in \mathcal{N}(v_i)} \sum_{n \in \mathcal{N}(v_j)} SimRank(m, n)$$

Liben-Nowell and Kleinberg, 2003

Community based methods

- Within-inter community/cluster of $v_i, v_j \in C$

$$\sum_{w \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} \frac{|w \in C|}{|w \notin C|}$$

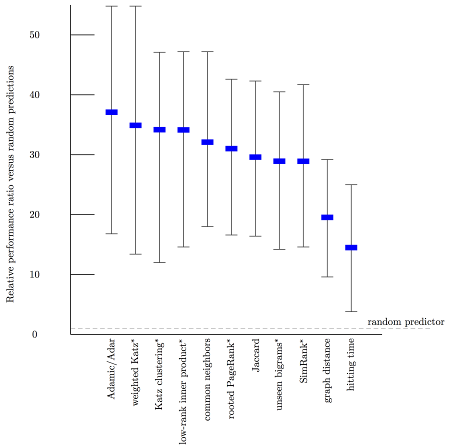
- Common neighbors with community information, $v_i, v_j \in C$, $f(w) = 1$ if $w \in C$

$$|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)| + \sum_{w \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} f(w)$$

- Resource allocation index with community information (soundarajan-hopcroft), $v_i, v_j \in C$, $f(w) = 1$ if $w \in C$

$$\sum_{w \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} \frac{f(w)}{|\mathcal{N}(w)|}$$

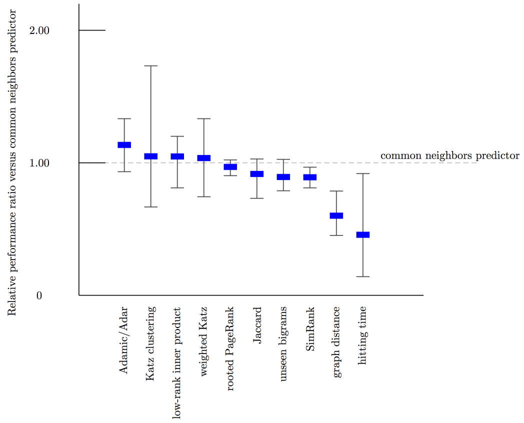
Evaluation of scoring prediction



Ratio of predictor performance over the baseline, averaged 5 datasets

Liben-Nowell and Kleinberg, 2007

Evaluation of scoring prediction



Ratio of predictor performance over the baseline, averaged 5 datasets

Liben-Nowell and Kleinberg, 2007

Classification for link prediction

Challenging classification problem:

- Computational cost of evaluating of very large number of possible edges (quadratic in number of nodes)
- Highly imbalanced class distribution: number of positive examples (existing edges) grows linearly and negative quadratically with number on nodes

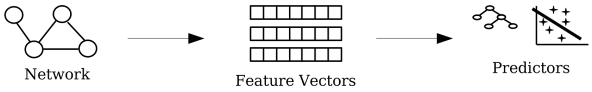
Link prediction with supervised learning

Supervised learning:

1. Features generation
2. Model training
3. Testing (model application)

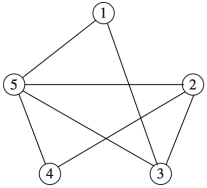
Features:

- Topological proximity features
- Aggregated features
- Content based node proximity features

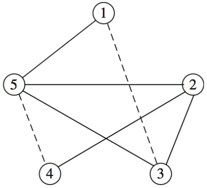


Simple evaluation

Simple "hold out set" evaluation



Whole graph



Training graph

Evaluation metrics

- Precision and Recall, F-measure

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}$$

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

- True positive rate (TPR), False positive rate (FPR), ROC curve, AUC

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}$$

Training and testing

Evaluation for evolving networks

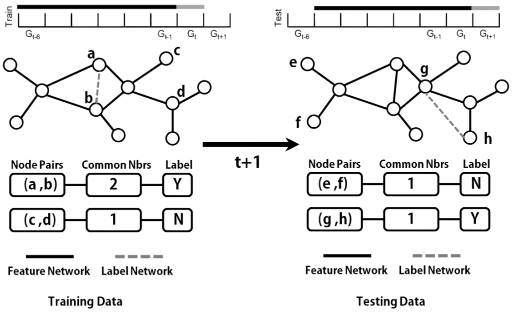


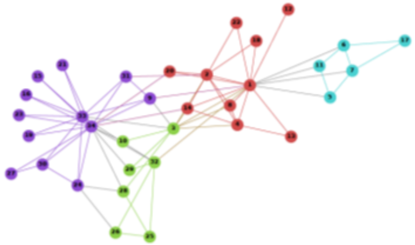
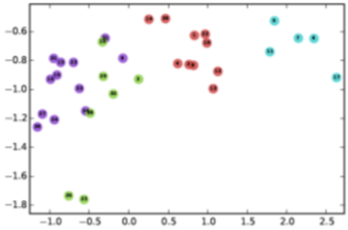
image from Y. Yang et.al, 2014

Lecture outline

Graph Embeddings

Graph Embeddings

- Necessity to automatically select features
- Reduce domain- and task- specific bias
- Unified framework to vectorize network
- Preserve graph properties in vector space
- Similar nodes $\rightarrow$ close embeddings

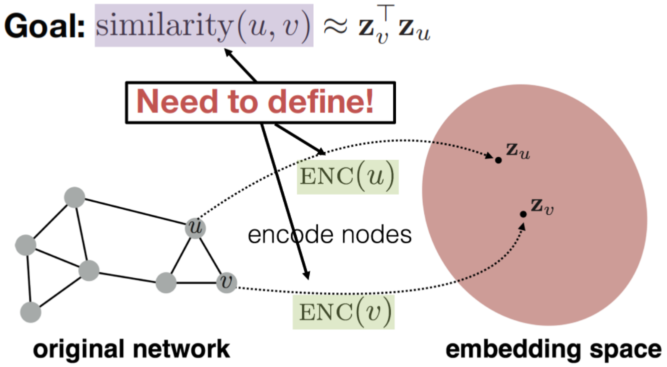



from Leskovec et al., 2018[7]

[7]http://snap.stanford.edu/proj/embeddings-www/

Graph Embeddings

- Define **Encoder**
- Define **Similarity**/graph feature to preserve graph properties
- Define similarity/distance in the embedding space
- **Optimize** loss to fit embedding with similarity computed on graph



from Leskovec et al., 2018

Structural Graph Embeddings

- Embedding look-up (each node - separate vector)
- Different similarity measures (adjacency, common neighbours, distances, exact function, etc.)
- Quadratic optimization for MSE loss
- Fast models via random walks

First-order Proximity

- Similarity between u and v is A_{uv}
- MSE Loss
- Variant of Matrix Decomposition

$$\underbrace{\boxed{\mathcal{L}}}_{\text{loss (what we want to minimize)}} = \underbrace{\boxed{\sum_{(u,v)\in V\times V}}}_{\text{sum over all node pairs}} \|\underbrace{\boxed{\mathbf{z}_u^\top \mathbf{z}_v}}_{\text{embedding similarity}} - \underbrace{\boxed{\mathbf{A}}}_{\text{(weighted) adjacency matrix for the graph}}{}_{u,v}\|^2$$

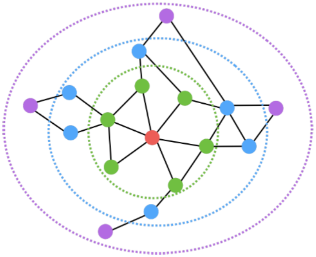
from Leskovec et al., 2018

First-order Proximity

- Pros:
  - Use SGD for scalable optimization
  - Matrix factorization (SVD) or decomposition (QR) may be applicable
- Cons:
  - Quadratic complexity
  - Large embeddings space
  - No indirect graph properties are preserved

Multi-order Proximity

- Similarity of neighborhoods of u and v via indices or k-hop paths
- Direct optimization of exact similarity metric



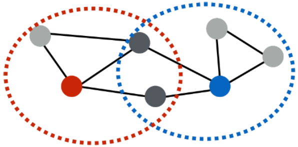
- **Red:** Target node
- **Green**: 1-hop neighbors
  - $\mathbf{A}$ (i.e., adjacency matrix)
- **Blue:** 2-hop neighbors
  - $\mathbf{A}^2$
- **Purple:** 3-hop neighbors
  - $\mathbf{A}^3$

$$\mathcal{L} = \sum_{(u,v)\in V\times V} \|\mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{u,v}^k\|^2$$

from Leskovec et al., 2018

Multi-order Proximity

- Similarity score S_{uv} as Jaccard/Common Neighbours, etc. (HOPE)



$$\mathcal{L} = \sum_{(u,v)\in V\times V} \| \boxed{\mathbf{z}_u^\top \mathbf{z}_v} - \boxed{\mathbf{S}_{u,v}} \|^2$$

embedding similarity

multi-hop network similarity (i.e., any neighborhood overlap measure)

- Weighted k-hop paths with different k (GraRep)

$$\tilde{\mathbf{A}}_{i,j}^k = \max\left(\log\left(\frac{(\mathbf{A}_{i,j}/d_i)}{\sum_{l\in V}(\mathbf{A}_{l,j}/d_l)^k}\right)^k - \alpha, 0\right)$$

node degree

constant shift

from Leskovec et al., 2018

- Even worse complexity

Random Walks

- Similarity between u and v is probability to co-occur on a random walk
- Sample each vertex u neighborhood $N_R(u)$ (multiset) by short random walks via strategy R
- Optimize similarity considering independent neighbor samples via MLE (remind Word2Vec)

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

from Leskovec et al., 2018

Random Walks

- $P(v|z_u)$ is approximated via softmax over similarity $z_u^T \cdot z_v$

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

- Problem in second Σ over all nodes
- Hard to find optimal solution

Negative Sampling

- Use *Negative Sampling* to approximate denominator

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

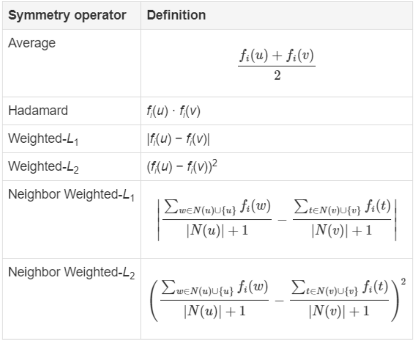
$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k} \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim \overset{\text{random distribution over all nodes}}{P_V}$$

from Leskovec et al., 2018

- Sample in proportion to node degree
- Experiment with k to impact negative prior and robustness
- No need to sample non-connected edges — same as random

Feature representation

- How to construct pair of nodes representation having node embeddings?
- Will it be more efficient than $\sigma(z_i^t \cdot z_j)$



Symmetry operator	Definition
Average	$\frac{f_i(u)+f_i(v)}{2}$
Hadamard	$f_i(u) \cdot f_i(v)$
Weighted-L_1	$\lvert f_i(u) - f_i(v) \rvert$
Weighted-L_2	$(f_i(u) - f_i(v))^2$
Neighbor Weighted-L_1	$\left\lvert \frac{\sum_{w \in N(u)\cup\{u\}} f_i(w)}{\lvert N(u)\rvert + 1} - \frac{\sum_{t \in N(v)\cup\{v\}} f_i(t)}{\lvert N(v)\rvert + 1} \right\rvert$
Neighbor Weighted-L_2	$\left(\frac{\sum_{w \in N(u)\cup\{u\}} f_i(w)}{\lvert N(u)\rvert + 1} - \frac{\sum_{t \in N(v)\cup\{v\}} f_i(t)}{\lvert N(v)\rvert + 1} \right)^2$

DOI: 10.7717/peerj-cs.172/table-2

from Makarov et al., 2019

Feature representation

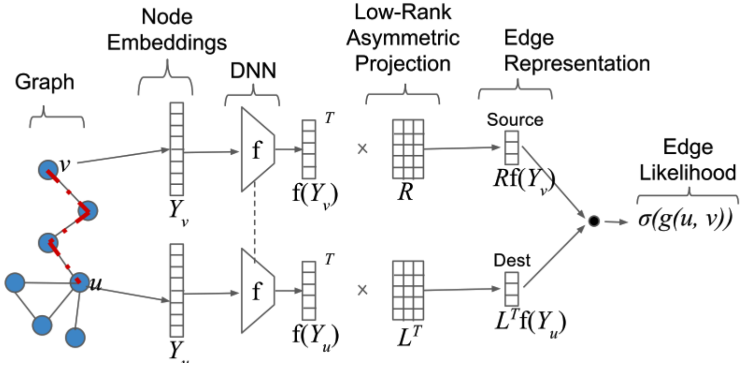
- How efficient simple solution?

- Works for undirected networks
- Samples neighbor information for low cost
- Not stable across different datasets (L_1 works in general better than L_2)
- For weighted networks it is better to solve binary classification stacked with regression rather then directly solve link regression problem

from Makarov et al., 2019

Directed network link prediction

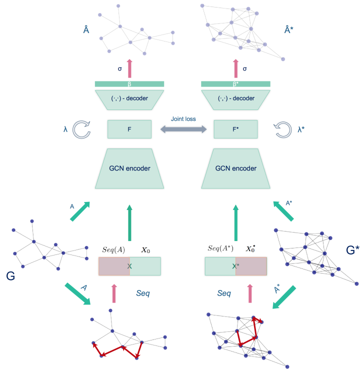
- When order matters, how to build classifier (see HOPE also)?
- Concat works not good probably - use asymmetric encoding via bi-linear form of compressed embeddings
  $M = LR,\ g(u, v) = f(Y_u)^t M f(Y_v)$



from Abu-El-Haija et al., 2018

Self-supervised learning via Line graph

- Edge-vertex dual (Line) graph allows to build dual representation and learn any edge embedding function
- Joint constraints on original and Line graph under bijective closure with agglutination of nodes embeddings in dual representation



from Makarov et al., 2021

Lecture outline

Graph Neural Networks

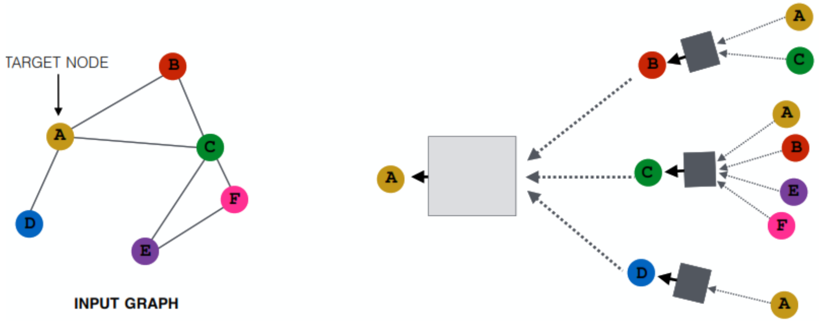
GNN

Graph Neural Network: Setting

- We have a graph $G(V, E)$ defined by adjacency matrix A and feature matrix $X \in \mathrm{R}^{f,|V|}$
- Confirmed relation between closeness of feature space and graph structure
- Non-graph features are vectorized separately (images, texts, one-hot encoding for labels, numeric features)

Graph Neural Network: Idea

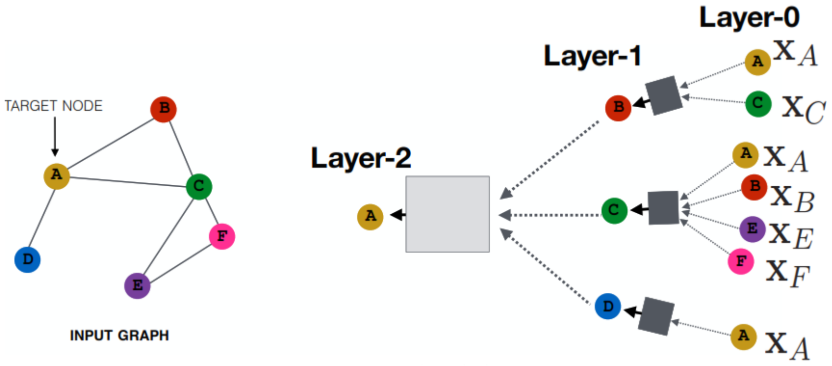
- Assign weights only to information obtained from neighbors
- Include node itself via loop with trainable weight
- Each node generate its own computational graph



from Leskovec et al., 2018

Graph Neural Network: Layer structure

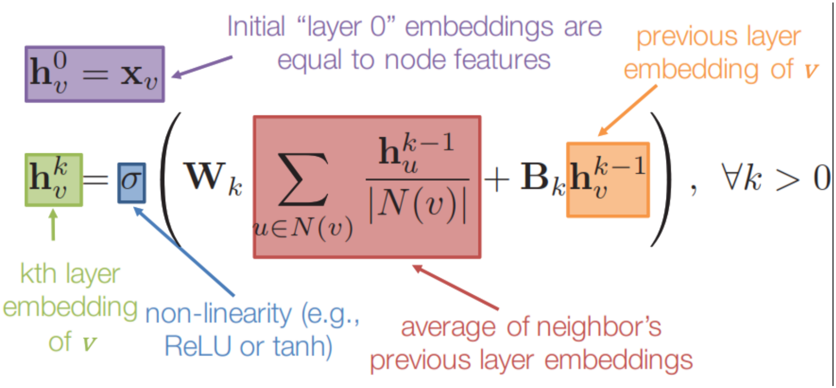
- Each aggregation defines new layer
- Zero-level embedding is non-graph feature
- Arbitrary depth but remember on “law of six handshakes”



from Leskovec et al., 2018

Graph Neural Network: Basic Approach

- Aggregation over weighted sum of neighbor input and node itself under non-linearity
- Use simple neural network construction



Initial "layer 0" embeddings are equal to node features
$\mathbf{h}_v^0 = \mathbf{x}_v$
previous layer embedding of v
$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k > 0$
kth layer embedding of v
non-linearity (e.g., ReLU or tanh)
average of neighbor's previous layer embeddings

from Leskovec et al., 2018

Graph Neural Network: Training

- Stop at K-th layer and feed h_v^K as embeddings to task-dependent loss; use SGD to optimize
- Unsupervised training uses reconstruction loss of adjacency matrix A (MSE, CE)
- (Semi-)Supervised loss feeds node embeddings to FC layer to predict labels under CE loss with possible Laplacian regularization
- When no features available, unsupervised training uses either one hot encoding for nodes (each node - separate label), or pretrains some structural embedding and feed them into feature matrix

Graph Neural Network: General Pipeline

- Define Aggregator
  - Different aggregators support only transductive learning for static graph
  - Sharing layer-wise weights allows inductive learning and inference on unseen nodes
- Define Loss
- Train on batches of nodes
- Generate output embeddings

Graph Convolutions

GCN

Graph Convolutional Network

- Aggregation over shared weights between node and its neighbors
- Normalization to stabilize training for high-degree nodes

Basic Neighborhood Aggregation

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

VS.

GCN Neighborhood Aggregation

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

same matrix for self and neighbor embeddings

per-neighbor normalization

from Leskovec et al., 2018

Graph Convolutional Network

- Efficient batch computation in matrix form
- Obtained $O(\mid E \mid)$ complexity (see pyG, DGL libraries)

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k)}\mathbf{W}_k\right)$$

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$$

$$\mathbf{D}_{ii} = \sum_j \mathbf{A}_{i,j}$$

from Leskovec et al., 2018

Graph ATtention

GAT

Graph ATtention Network

- Not all the neighbors are equal

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)}$$

$\|$ is the concatenation operation.

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

from Bengo et al., 2018

Graph ATtention Network

- Multi-head attention works better like in different convolution filters
- Final layer require pooling isntead of concatenation

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

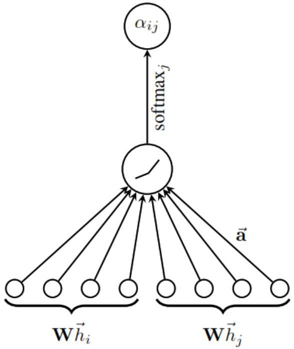
$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{k}\mathbf{W}^{k}\vec{h}_j\right)$$

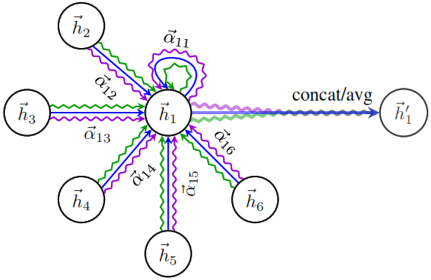
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{k}\mathbf{W}^{k}\vec{h}_j\right)$$

from Bengo et al., 2018

Graph ATtention Network

- Feature aggregation via attention over learned weights
- Different patterns for the same structure





from Bengo et al., 2018

Message Aggregation

GraphSAGE

GraphSAGE: Feature Pyramid

- Vary feature space across layers
- Aggregate from neighbors and concatenate with self-representation

Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

GraphSAGE:

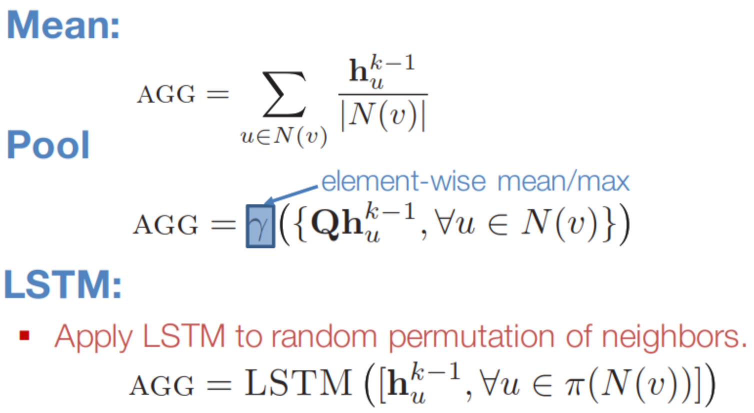
concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \boxed{\mathrm{AGG}\left(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)}, \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

from Leskovec et al., 2018

GraphSAGE: Differentiable Aggregators



Mean:

$$\mathrm{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

Pool

element-wise mean/max

$$\mathrm{AGG} = \boxed{\gamma}\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

LSTM:

- Apply LSTM to random permutation of neighbors.

$$\mathrm{AGG} = \mathrm{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

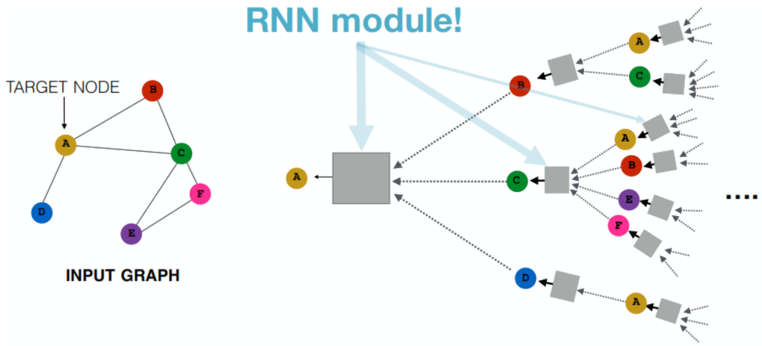
from Leskovec et al., 2018

More layers?

How to fight dimension curse

Model Depth

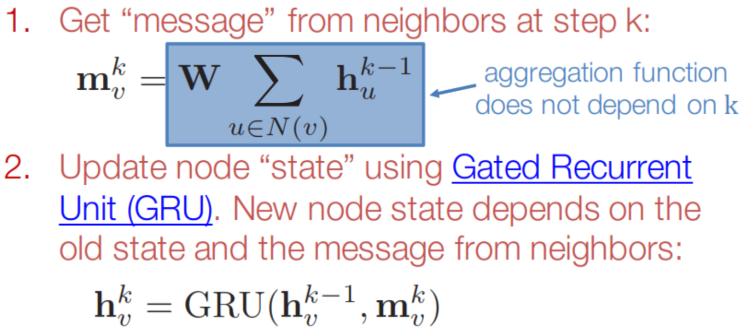
- Usually 2-3 layers for GCN / GraphSAGE
- More layers make method global
- Computation graph exceed memory limits
- Overfitting, vanishing gradient



from Leskovec et al., 2018

Gated GNN

- Use recurrent model with shared weights across all the layers, support any depth



1. Get “message” from neighbors at step k:

$$\mathbf{m}_v^k = \boxed{\mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}} \leftarrow \text{aggregation function does not depend on k}$$

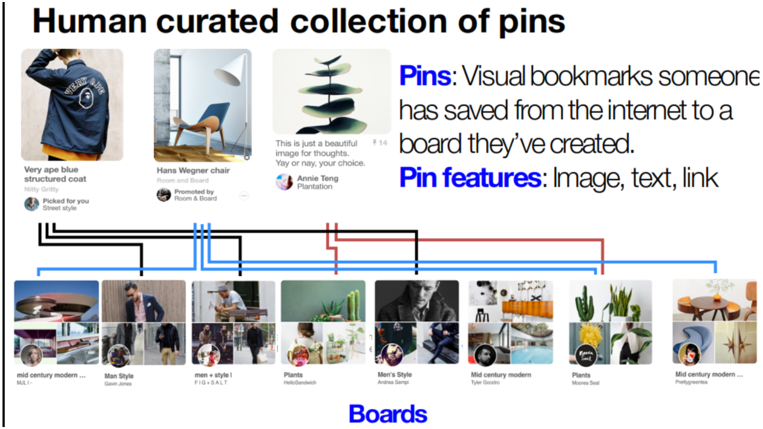
2. Update node “state” using Gated Recurrent Unit (GRU). New node state depends on the old state and the message from neighbors:

$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

from Leskovec et al., 2018

Large Scale RecSys: PinSAGE

- Pinterest: 3 billion pins and boards; 16 billion interactions; label, text and image features


Human curated collection of pins

Pins: Visual bookmarks someone has saved from the internet to a board they've created.

Pin features: Image, text, link

from Leskovec et al., 2018

Large Scale RecSys: PinSAGE

Recommendations pipeline:

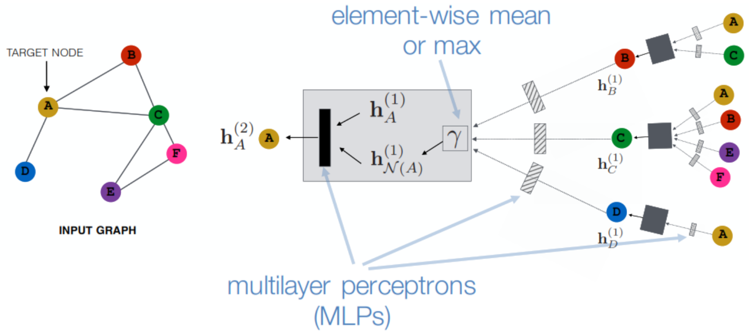
- Collect consequent clicks
- Train system using metric learning approach
- Generate embeddings
- Recommend via k-NN

Key advances:

- Sub-sample neighborhoods for efficient GPU batching
- Producer-consumer training pipeline
- Curriculum learning for negative samples
- MapReduce for efficient inference

Large Scale RecSys: RW-GCN

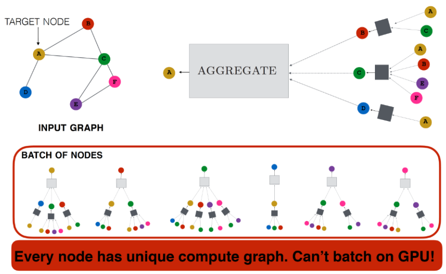
- Train so that pins that are consecutively clicked have similar embeddings, use smart negative sampling

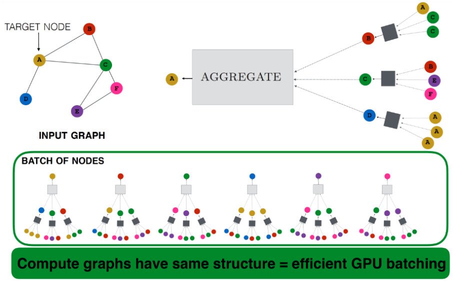


from Leskovec et al., 2018

Large Scale RecSys: Batch Sampling

- Use one computation graph, sample nodes according top-PPR among neighbors





from Leskovec et al., 2018

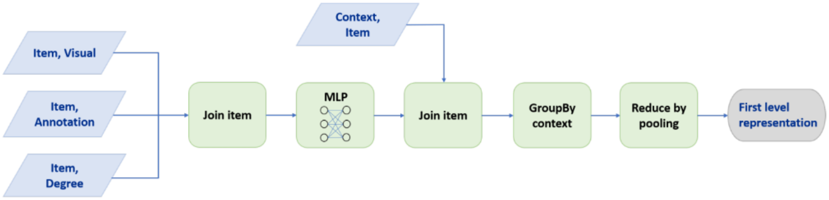
Large Scale RecSys: Training

CPU (producer):

- Select a batch of pins
- Run random walks (for PPR approximation)
- Construct their computation graphs

GPU (consumer):

- Multi-layer aggregations
- Loss computation
- Backprop



from Leskovec et al., 2018

Large Scale RecSys: Training

- Include more and more hard negative samples for each epoch

$$\mathcal{L} = \sum_{(u,v)\in \boxed{\mathcal{D}}} \max(0, -\boxed{\mathbf{z}_u^\top \mathbf{z}_v} + \mathbf{z}_u^\top \boxed{\mathbf{z}_n} + \boxed{\Delta})$$

set of training pairs from user logs

“positive”/true training pair

“negative” sample

“margin” (i.e., how much larger positive pair similarity should be compared to negative)

Source pin

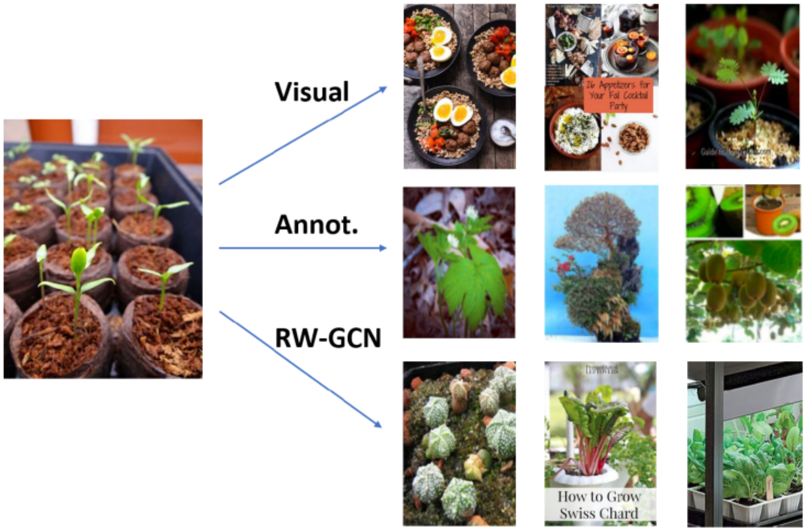
Positive

Easy negative

Hard negative

from Leskovec et al., 2018

Large Scale RecSys: Visual Comparison

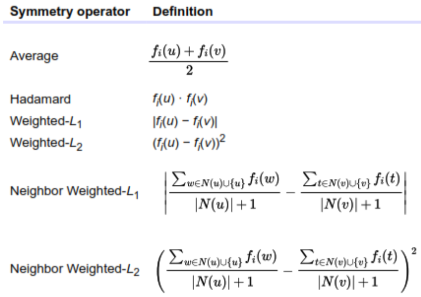


Is everything so simply?

Open Problems

Open Problems: Edge embedding

- What is the best way to compose edge feature?

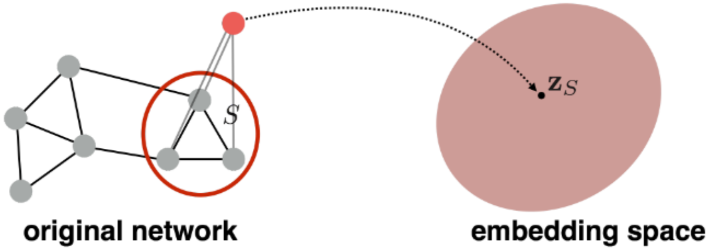


Symmetry operator	Definition
Average	$\frac{f_i(u)+f_i(v)}{2}$
Hadamard	$f_i(u) \cdot f_i(v)$
Weighted-L_1	$\lvert f_i(u) - f_i(v) \rvert$
Weighted-L_2	$(f_i(u) - f_i(v))^2$
Neighbor Weighted-L_1	$\left\lvert \frac{\sum_{w \in N(u) \cup \{u\}} f_i(w)}{\lvert N(u) \rvert + 1} - \frac{\sum_{t \in N(v) \cup \{v\}} f_i(t)}{\lvert N(v) \rvert + 1} \right\rvert$
Neighbor Weighted-L_2	$\left(\frac{\sum_{w \in N(u) \cup \{u\}} f_i(w)}{\lvert N(u) \rvert + 1} - \frac{\sum_{t \in N(v) \cup \{v\}} f_i(t)}{\lvert N(v) \rvert + 1} \right)^2$

from Makarov et al., 2019

Open Problems: Subgraph embedding

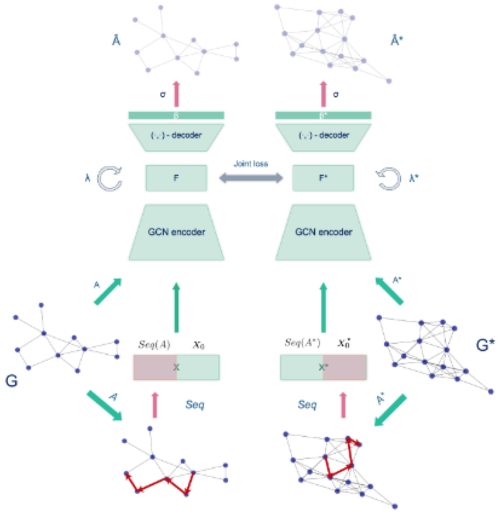
- Even for triangle it is an open question.
- Use sum of embeddings
- Use virtual supernode (same as for whole graph embedding)



from Leskovec et al., 2018

Open Problems: Node & Edge embedding

- How to optimize joint node and edge features?



Open Problems: Text + Graph Fusion

- How to fuse partially-correlated text embeddings and graph embeddings?

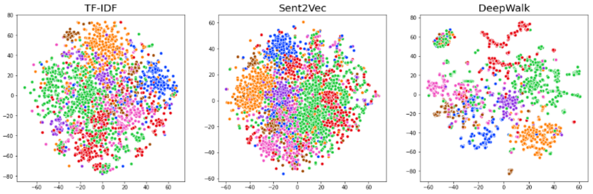


Figure 1. TF-IDF, Sent2Vec and DeepWalk embeddings visualization on Cora

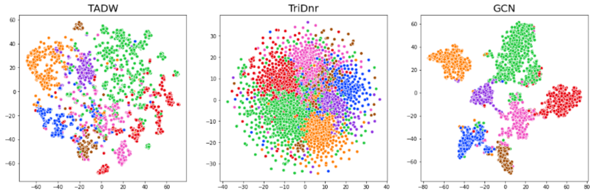
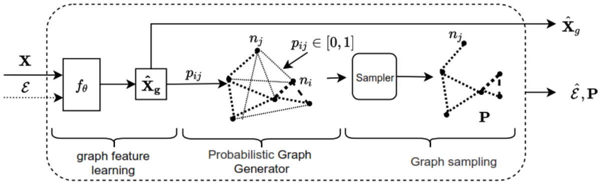


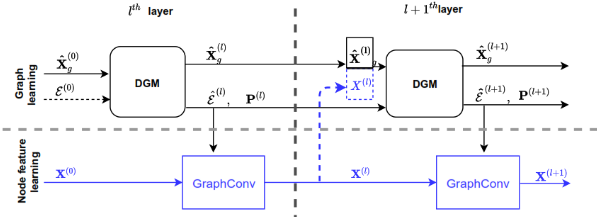
Figure 2. TADW, TriDnr and GCN embeddings visualization on Cora

from Makarov et al., 2021

Open Problems: Graphs from Metric Learning

- How to work with non-stationary graph obtained from geometric learning?

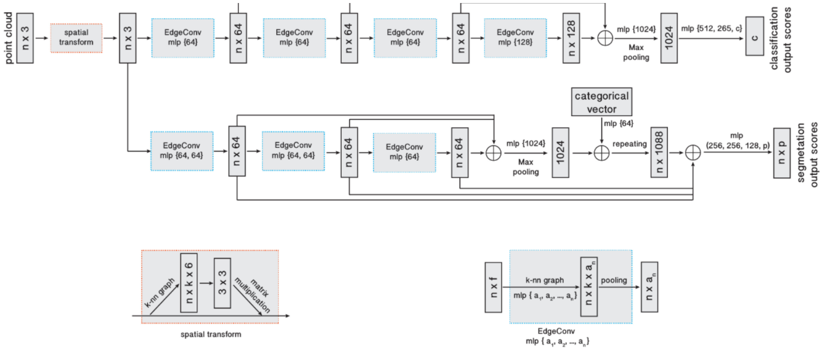




Differentiable Graph Module (DGM) for Graph Convolutional Networks from Bronshtein et al., 2020

Open Problems: Graphs from Metric Learning

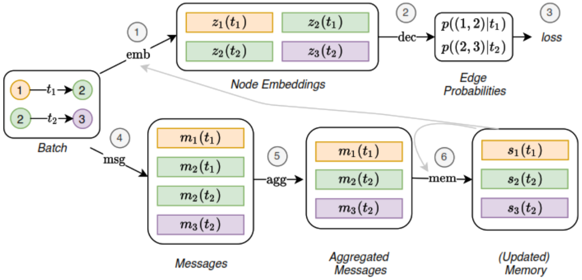
- How to work with non-stationary graph obtained from geometric learning?



Dynamic Graph CNN for Learning on Point Clouds from Solomon et al., 2019

Open Problems: Temporal Graphs

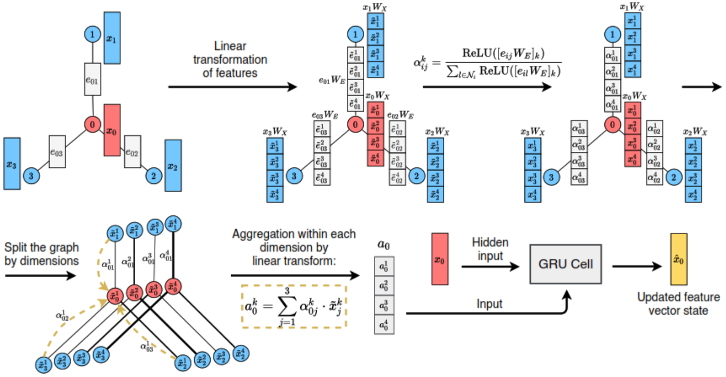
- How to work with large dynamic networks?



TEMPORAL GRAPH NETWORKS FOR DEEP LEARNING ON DYNAMIC GRAPHS from Bronshtein et al., 2019

Open Problems: Temporal Graphs

- How to work with large dynamic networks?


Linear transformation of features
$\alpha_{ij}^k = \frac{\mathrm{ReLU}([e_{ij}W_E]_k)}{\sum_{l\in\mathcal{N}_i}\mathrm{ReLU}([e_{il}W_E]_k)}$
Split the graph by dimensions
Aggregation within each dimension by linear transform:
$a_0^k = \sum_{j=1}^{3} \alpha_{0j}^k \cdot \tilde{x}_j^k$
Hidden input
Input
GRU Cell
Updated feature vector state

EWS-GCN by Sberbank, 2020

Open Problems: What else?

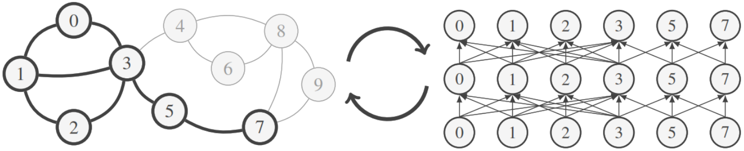
- How to choose embedding?
- How to mix embeddings and pretrain/initialize?
- How to fuse (heterogeneous) graphs and futures?
- How to speed-up GCN and other models?
- Graph RecSys still struggle from cold start problem!
- Transfer learning and GNN AutoML is hard to improve!
- Working with large dynamic graphs with changing features is still hard!

Modern Architectures

State-of-the-art

GraphSaint

- Sample from graph and train FC GCN



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$ Full GCN on $\mathcal{G}_s$

Algorithm 1 `GraphSAINT` training algorithm

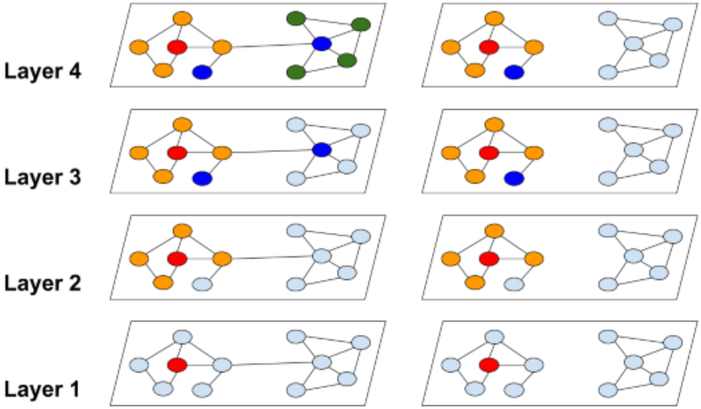
Input: Training graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{X})$; Labels $\overline{\boldsymbol{Y}}$; Sampler `SAMPLE`;
Output: GCN model with trained weights

```
1: Pre-processing: Setup SAMPLE parameters; Compute normalization coefficients $\alpha$, $\lambda$.
2: for each minibatch do
3:     $\mathcal{G}_s(\mathcal{V}_s, \mathcal{E}_s) \leftarrow$ Sampled sub-graph of $\mathcal{G}$ according to SAMPLE
4:     GCN construction on $\mathcal{G}_s$.
5:     $\{\boldsymbol{y}_v \mid v \in \mathcal{V}_s\} \leftarrow$ Forward propagation of $\{\boldsymbol{x}_v \mid v \in \mathcal{V}_s\}$, normalized by $\alpha$
6:     Backward propagation from $\lambda$-normalized loss $L(\boldsymbol{y}_v, \overline{\boldsymbol{y}}_v)$. Update weights.
7: end for
```

University of Southern California, 2019

ClusterGCN

- Limit Sampling by Cluster properties via RWs

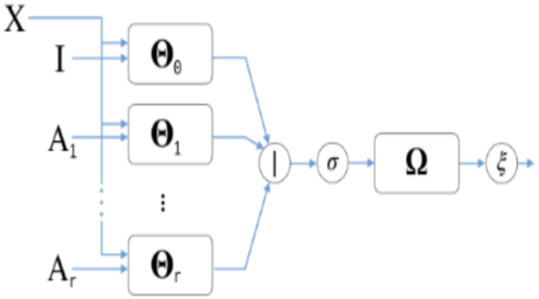


Google Research, University of California, 2020

SIGN

- Precompute diffusion-based sampling instead of stacking more layers
- Decouple graph convolutions as backbone

$$Y = \xi(\tilde{A}^L X \Theta^{(1)} \cdots \Theta^{(L)}) = \xi(\tilde{A}^L X \Theta).$$



Twitter, Imperial College London, 2020

Applications

- ML: NAS & AutoML
- NLP: context embeddings, BERT as transformer solves LP
- CV: 3D point clouds, few-shot learning, KG for captioning
- DM: KG extraction, mining relations
- RecSys: Embedding of everything, tensor decomposition
- RL: Model MDP states via GCN embeddings
- Biology/Chemistry: drug discovery, protein interaction, new materials

Libraries:

- DGL, pyG, DGM, etc.
- "awesome graph embedding"

References (Node Classification)

- S. A. Macskassy, F. Provost, Classification in Networked Data: A Toolkit and a Univariate Case Study. Journal of Machine Learning Research 8, 935-983, 2007
- Bengio Yoshua, Delalleau Olivier, Roux Nicolas Le. Label Propagation and Quadratic Criterion. Chapter in Semi-Supervised Learning, Eds. O. Chapelle, B. Scholkopf, and A. Zien, MIT Press 2006
- Smriti Bhagat, Graham Cormode, S. Muthukrishnan. Node classification in social networks. Chapter in Social Network Data Analytics, Eds. C. Aggrawal, 2011, pp 115-148
- D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In NIPS, volume 16, 2004.
- X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In ICML, 2003.
- M. Belkin, P. Niyogi, V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. J. Mach. Learn. Res., 7, 2399-2434, 2006

References (Node Classification)

- Kruskal J, Wish M. 1978. Multidimensional Scaling. New York: SAGE Publications
- Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. 1990. Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6):391-407
- Martinez AM, Kak AC. 2001. Pca versus lda. IEEE Transactions on Pattern Analysis and Machine Intelligence 23(2):228-233
- Tenenbaum JB, De Silva V, Langford JC. 2000. A global geometric framework for nonlinear dimensionality reduction. Science 290(5500):2319-2323
- Roweis ST, Saul LK. 2000. Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500):2323-2326
- He X, Niyogi P. 2004. Locality preserving projections

References (Matrix Factorization)

- Chung FR, Graham FC. 1997. Spectral graph theory. Rhode Island: American Mathematical Soc. 92
- Belkin M, Niyogi P. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering
- Brand M. 2003. Continuous nonlinear dimensionality reduction by kernel eigenmaps
- Luo D, Nie F, Huang H, Ding CH. 2011. Cauchy graph embedding
- Shaw B, Jebara T. 2009. Structure preserving embedding
- Ahmed A, Shervashidze N, Narayanamurthy S, Josifovski V, Smola AJ. 2013. Distributed large-scale natural graph factorization

References (Link Prediction)

- D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. Journal of the American Society for Information Science and Technology, 58(7):1019?1031, 2007
- R. Lichtenwalter, J.Lussier, and N. Chawla. New perspectives and methods in link prediction. KDD 10: Proceedings of the 16th ACM SIGKDD, 2010
- M. Al Hasan, V. Chaoji, S. Salem, M. Zaki, Link prediction using supervised learning. Proceedings of SDM workshop on link analysis, 2006
- M. Rattigan, D. Jensen. The case for anomalous link discovery. ACM SIGKDD Explorations Newsletter. v 7, n 2, pp 41-47, 2005
- M. Al. Hasan, M. Zaki. A survey of link prediction in social networks. In Social Networks Data Analytics, Eds C. Aggarwal, 2011.

References (Link Prediction)

- B. Perozzi, R. Al-Rfou, and S. Skiena. "Deepwalk: Online learning of social representations." In Proceedings of the 20th ACM SIGKDD international conference , pp. 701-710. 2014.
- Mutlu, Ece C., and Toktam A. Oghaz. "Review on graph feature learning and feature extraction techniques for link prediction." arXiv preprint arXiv:1901.03425 (2019).
- Makarov, Ilya, Olga Gerasimova, Pavel Sulimov, and Leonid E. Zhukov. "Dual network embedding for representing research interests in the link prediction problem on co-authorship networks." PeerJ Computer Science 5 (2019): e172.
- S. Abu-El-Haija, B. Perozzi, and R. Al-Rfou. "Learning edge representations via low-rank asymmetric projections." In Proceedings of the 2017 ACM CIKM conference, pp. 1787-1796. 2017.
- H. Cai, V.W. Zheng, and K.C.C. Chang. "A comprehensive survey of graph embedding: Problems, techniques, and applications." IEEE Transactions on Knowledge and Data Engineering 30, no. 9: 1616-1637, 2018

References (Graph Embeddings)

- H. Cai, V.W. Zheng, and K.C.C. Chang. "A comprehensive survey of graph embedding: Problems, techniques, and applications." IEEE Transactions on Knowledge and Data Engineering 30, no. 9: 1616-1637, 2018
- Makarov, Ilya, Dmitrii Kiselev, Nikita Nikitinsky, and Lovro Subelj. "Survey on graph embeddings and their applications to machine learning problems on graphs." PeerJ Computer Science 7 (2021).

References (Structural Embeddings)

- B. Perozzi, R. Al-Rfou, and S. Skiena. "Deepwalk: Online learning of social representations." In Proceedings of the 20th ACM SIGKDD international conference , pp. 701-710. 2014.
- J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. "Line: Large-scale information network embedding." In Proceedings of the 24th WWW international conference , pp. 1067-1077. 2015.
- A. Grover and J. Leskovec. "node2vec: Scalable feature learning for networks." In Proceedings of the 22nd ACM SIGKDD international conference, pp. 855-864. 2016.
- S. Abu-El-Haija, B. Perozzi, and R. Al-Rfou. "Learning edge representations via low-rank asymmetric projections." In Proceedings of the 2017 ACM CIKM conference, pp. 1787-1796. 2017.
- H. Cai, V.W. Zheng, and K.C.C. Chang. "A comprehensive survey of graph embedding: Problems, techniques, and applications." IEEE Transactions on Knowledge and Data Engineering 30, no. 9: 1616-1637, 2018

References (GNNs)

- Scarselli, Franco, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. "Graph neural networks for ranking web pages." In The 2005 IC on Web Intelligence (WI'05), pp. 666-672. IEEE, 2005.
- Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907. 2016.
- Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." arXiv preprint arXiv:1710.10903. 2017.
- Ying, Rex, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. "Graph convolutional neural networks for web-scale recommender systems." In Proceedings of the 24th ACM SIGKDD, pp. 974-983. 2018.

References (Modern GNNs)

- Zeng, Hanqing, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. "Graphsaint: Graph sampling based inductive learning method." arXiv preprint arXiv:1907.04931. 2019.
- Chiang, Wei-Lin, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 257-266. 2019.
- Rossi, Emanuele, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. "SIGN: Scalable Inception Graph Neural Networks." arXiv preprint arXiv:2004.11198. 2020.