Machine Learning on graphs. Graph Neural Networks

I. Makarov & L.E. Zhukov

BigData Academy MADE from Mail.ru Group

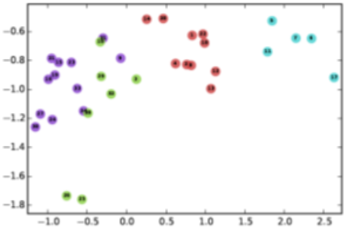
Network Science

Lecture outline

1. Graph Embeddings: Recap
2. Graph Neural Networks
   - Graph Convolutional Networks
   - Graph ATtention
   - GraphSAGE & Inductive Learning
3. PinSAGE & Large-Scale Recommendations
4. Open Problems
5. Modern Models
6. Application to other CS Domains

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 → close embeddings

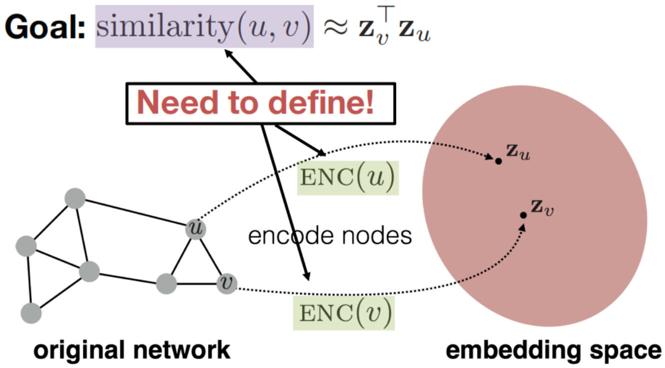


from Leskovec et al., 2018[1]

[1] 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

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

Short conclusion for structural Graph Embeddings

- Random walks are powerfull tool for fast network embedding
- Proximity-aware embeddings, random walks can be modelled in terms of each other (and even deep neural networks !)
- complexity and space are important to choose the embedding model
- provided models are used for transductive learning only, inductive learning require additional regularizations and local optimizations
- large graphs are hard to fit with handcrafted sampling strategies
- no clear way to support features

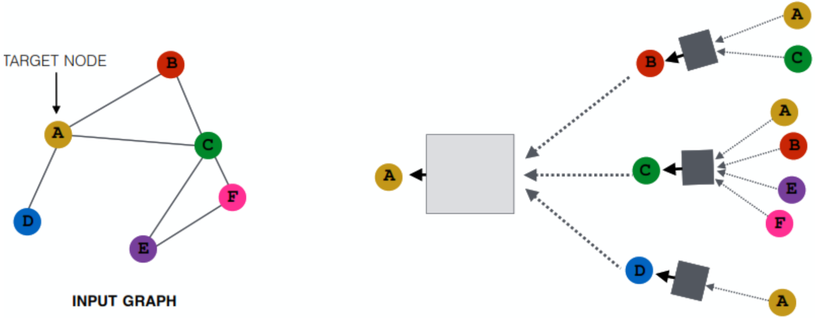
GNN

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

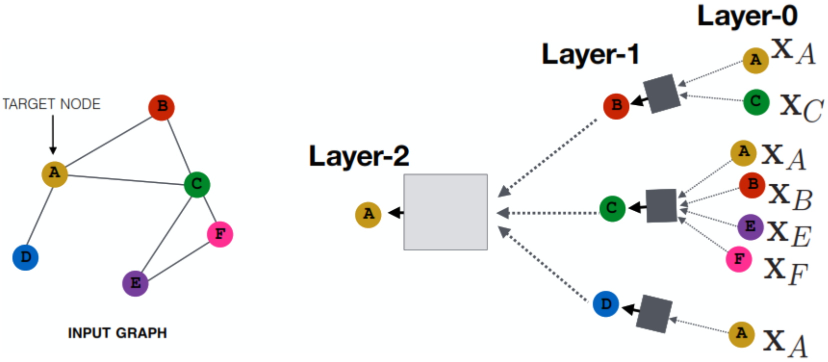
- Assign weights only to information obtained from neighbors
- Include node itself via loop with trainable weight
- Each node generate its own computational graph



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

$$\boxed{\mathbf{h}_v^0 = \mathbf{x}_v}$$

previous layer embedding of v

$$\boxed{\mathbf{h}_v^k} = \boxed{\sigma}\left(\mathbf{W}_k \boxed{\sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}} + \mathbf{B}_k \boxed{\mathbf{h}_v^{k-1}}\right), \quad \forall k > 0$$

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

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

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

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

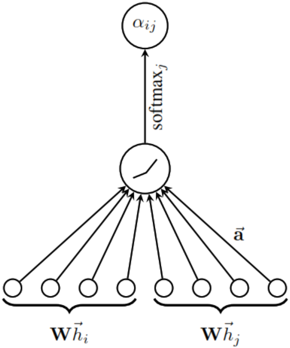
$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

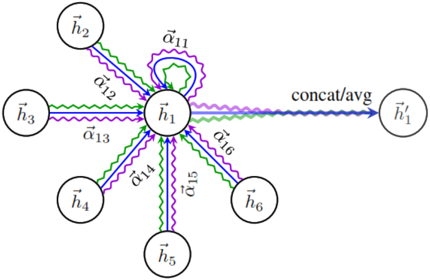
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j\right)$$

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

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

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

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.

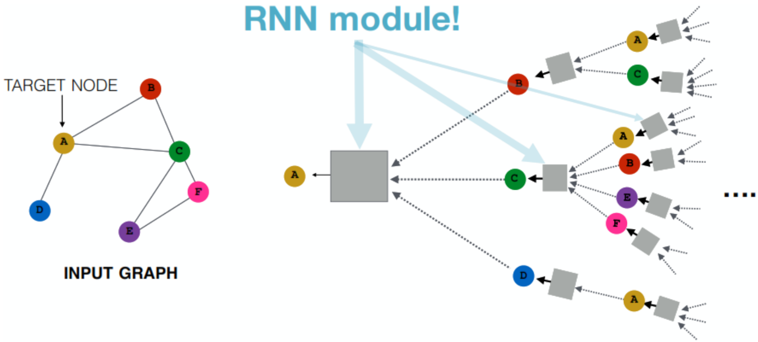
$$\mathrm{AGG} = \mathrm{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

from Leskovec et al., 2018

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:

$$\mathbf{m}_v^k = \boxed{\mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}}$$

aggregation function does not depend on k

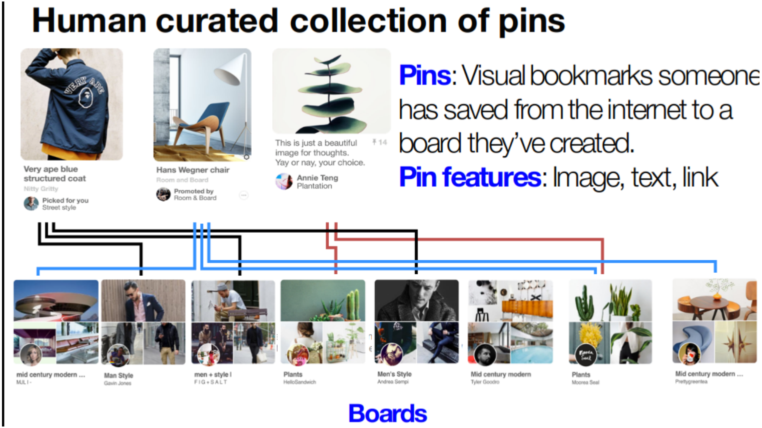
2. Update node "state" using Gated Recurrent Unit (GRU). New node state depends on the old state and the message from neighbors:

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

from Leskovec et al., 2018

Large Scale RecSys: PinSAGE

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



from Leskovec et al., 2018

Large Scale RecSys: PinSAGE

Recommendations pipeline:

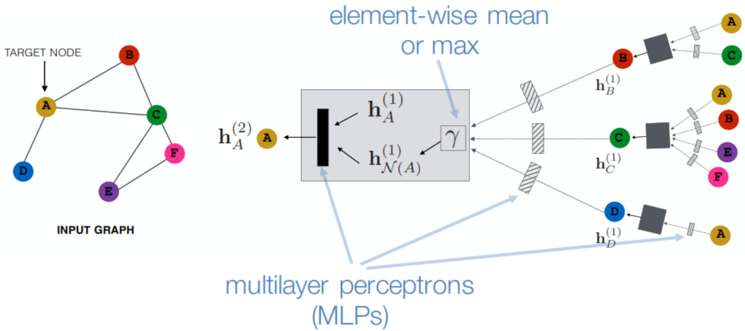
- Collect consequent clicks
- Train system using metric learning approach
- Generate embeddings
- Recommend via k-NN

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

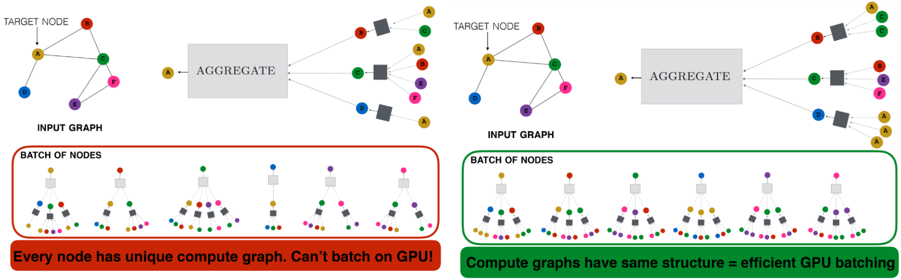
- Train so that pins that are consecutively clicked have similar embeddings, use smart negative sampling



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

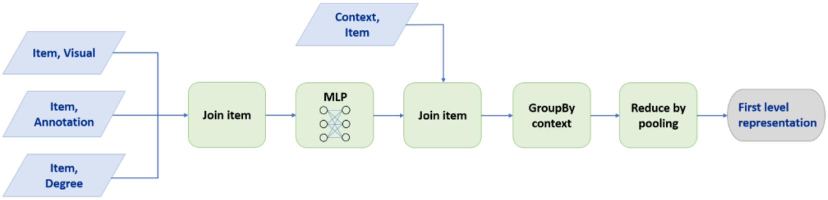
Large Scale RecSys: Training

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

- 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

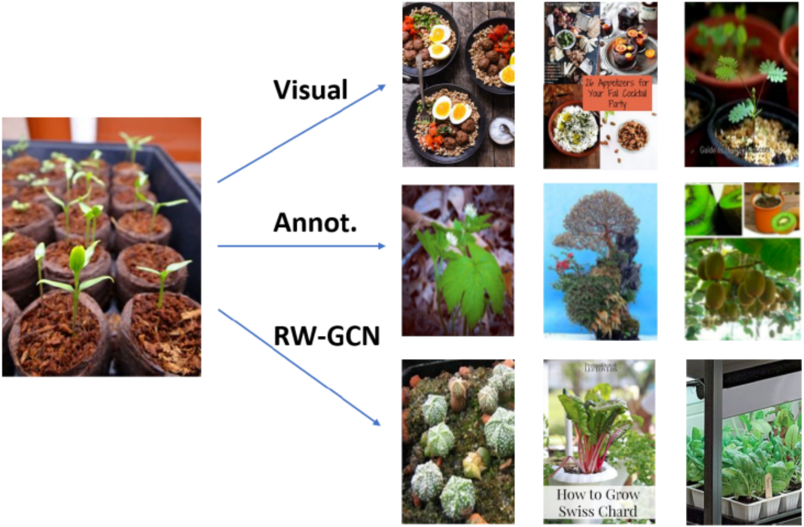
Positive

Easy negative

Hard negative

from Leskovec et al., 2018

Large Scale RecSys: Visual Comparison



Open Problems

Open Problems: Edge embedding

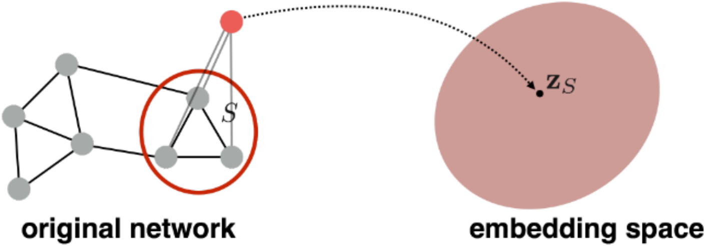
- What is the best way to compose edge feature?

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

from Makarov et al., 2019

Open Problems: Subgraph embedding

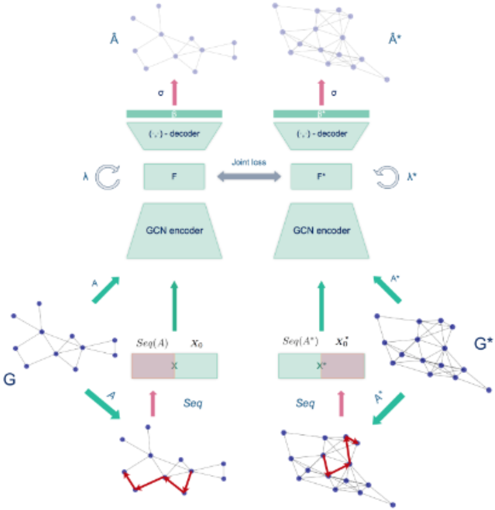
- Even for triangle it is an open question.
- Use sum of embeddings
- Use virtual supernode (same as for whole graph embedding)



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?

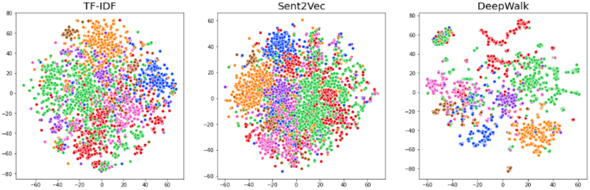


Figure 1. TF-IDF, Sent2Vec and DeepWalk embeddings visualization on Cora

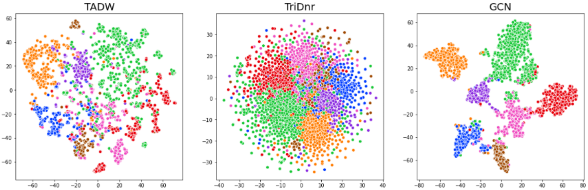
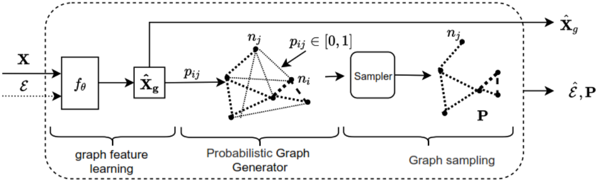


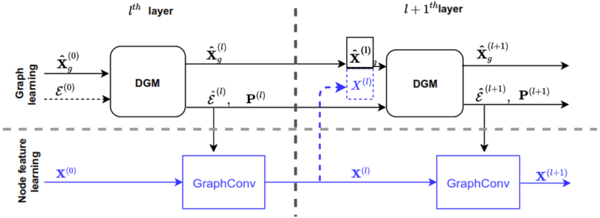
Figure 2. TADW, TriDnr and GCN embeddings visualization on Cora

from Makarov et al., 2021

Open Problems: Graphs from Metric Learning

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

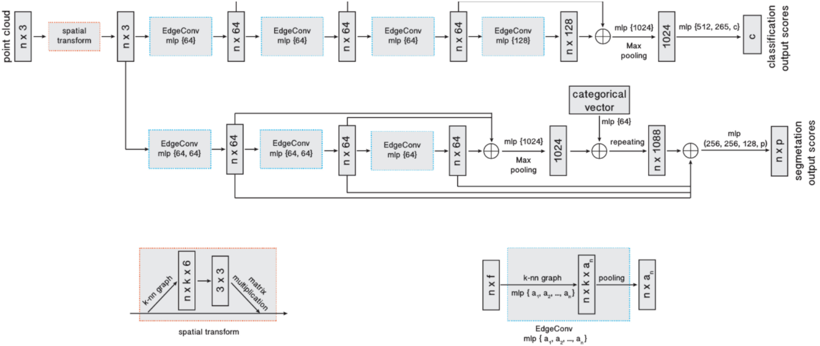




Differentiable Graph Module (DGM) for Graph Convolutional Networks from Bronshtein et al., 2020

Open Problems: Graphs from Metric Learning

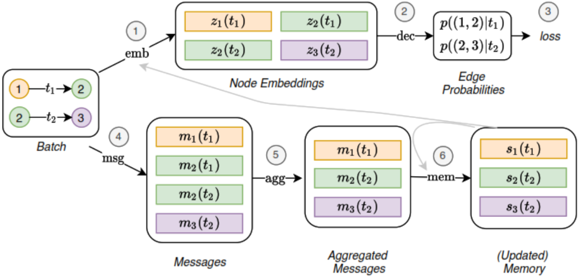
- How to work with non-stationary graph obtained from geometric learning?



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

Open Problems: Temporal Graphs

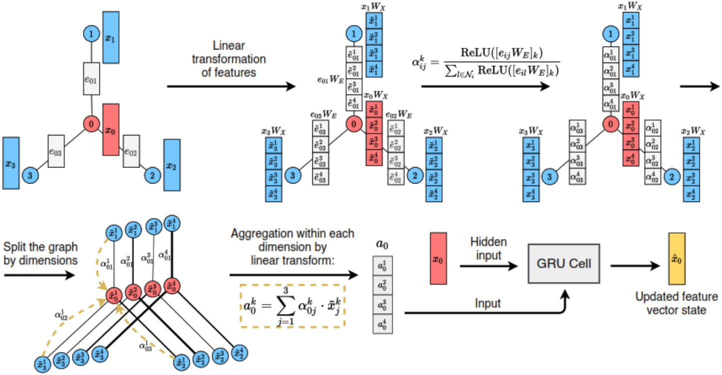
- How to work with large dynamic networks?



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?



EWS-GCN by Sberbank, 2020

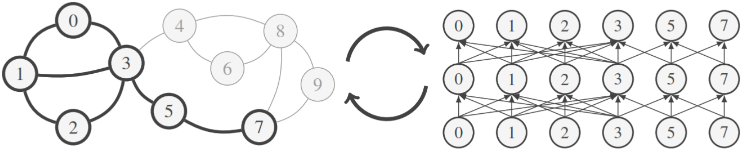
Open Problems: What else?

- 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!

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

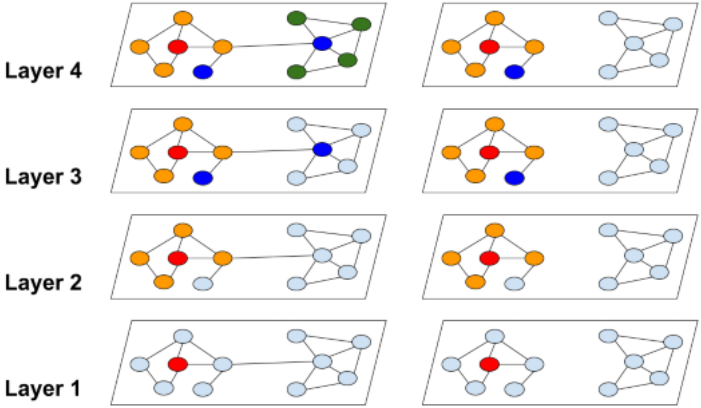
Input: Training graph $\mathcal{G}\left(\mathcal{V}, \mathcal{E}, \boldsymbol{X}\right)$; Labels $\overline{\boldsymbol{Y}}$; Sampler `SAMPLE`;
Output: GCN model with trained weights

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

University of Southern California, 2019

ClusterGCN

- Limit Sampling by Cluster properties via RWs

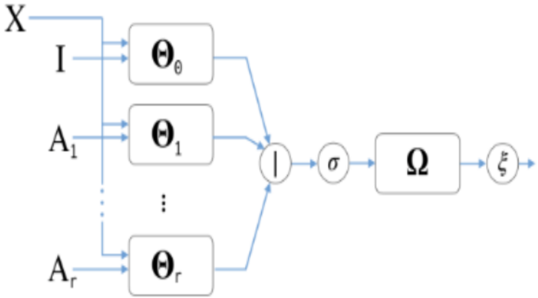


Google Research, University of California, 2020

SIGN

- Precompute diffusion-based sampling instead of stacking more layers

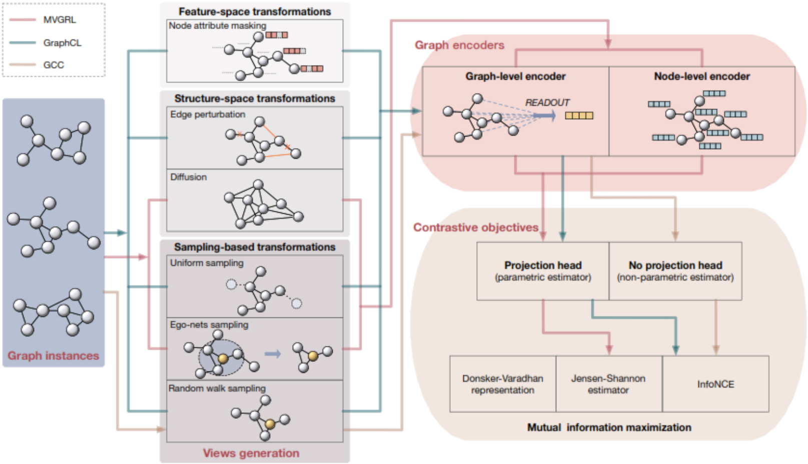
$$Y = \xi(\tilde{A}^L X \Theta^{(1)} \cdots \Theta^{(L)}) = \xi(\tilde{A}^L X \Theta).$$



Twitter, Imperial College London, 2020

Self-supervised GML

- Contrastive learning / graph augmentation



Amazon, Texas A&M University, 2021

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 (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.

References (structural)

- 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