Introduction to GNNs and Self-supervised Learning on Graphs

I. Makarov & V. Pozdnyakov & D. Kiselev

BigData Academy MADE from Mail.ru Group

Graph Neural Networks and Applications

Class Details

- Instructor: Ilya Makarov
- Tutor: Vitaly Pozdnyakov, Dmitrii Kiselev
- Invited lecturers: to be announced
- Course length: 11-12 lectures/classes
- Lecturer's Telegram: @iamakarov (urgent)
- Tutor's Telegram: @pozdnyakov_vitaliy (urgent), @dkiselev (urgent)
- Discord (questions, deadlines, grading)
- Programming: Python, iPython notebooks, Anaconda distribution
- Python libraries: NetworkX, pyG, DGL

Prerequisites

- Network Science
- Deep Learning
- Machine Learning
- Differential Equations
- Programming in Python

Resources

- "CS224W: Machine Learning with Graphs". Jure Leskovec. Stanford, 2020-2021.
- "SNAP Project". http://snap.stanford.edu/
- "PyG & PyG Temporal". https://pytorch-geometric.readthedocs.io/en/latest/, https://pytorch-geometric-temporal.readthedocs.io/en/latest/
- "Network Science", Albert-Laszlo Barabasi, Cambridge University Press, 2016. http://networksciencebook.com
- "Network Science" course by Leonid E. Zhukov.

Topics (max list)

1. Self-supervised learning on graphs
2. Subgraph Embedding & Deep Sets
3. Scaling GNNs: Efficient models, Sampling models
4. Compressing GNNs: Quantization, Distillation
5. Deep Generative Graph Models & adversarial robustness
6. Temporal graph embeddings
7. GNN Explanations
8. GNN & RecSys & KG
9. Query Embeddings for KGs
10. Transport GNNs + Combinatorics
11. Guest lectures: Applications to Biology, Medicine, Bioinformatics, Chemistry, Physics
12. OGB benchmark (optional challenge)

Recap

Go to Lecture 1: SSL on Graphs

Network Science

1. Real-world networks
2. Random graphs
3. Centrality & Prestige node measures
4. Structural similarity in networks
5. Community detection
6. Modeling dynamics on graphs
   - continuous: diffusion, epidemics
   - discrete: information propagation, segregation
7. Graph Machine Learning:
   - label propagation and node similarities
   - node and edge embeddings
   - graph neural networks
8. Knowledge Graphs

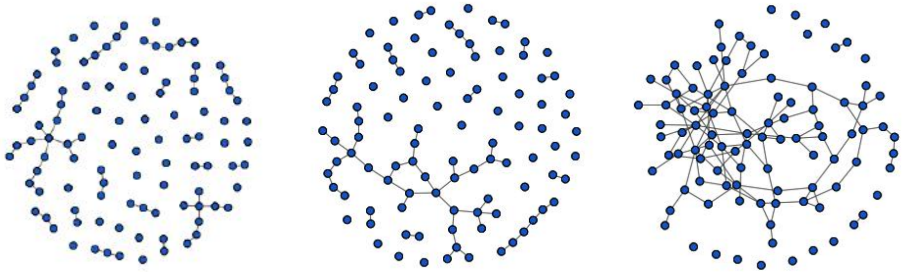
Complex networks (nor regular, nor random)

1. Power law node degree distribution: "scale-free" networks
2. Small diameter and average path length: "small world" networks
3. Hight clustering coefficient: transitivity

Random graph model

Consider $G_{n,p}$ as a function of p

- $p = 0$, empty graph - $\langle k \rangle = 0$
- $p = 1$, complete (full) graph - $\langle k \rangle = n - 1$
- n_G -largest connected component, $s = \frac{n_G}{n}$

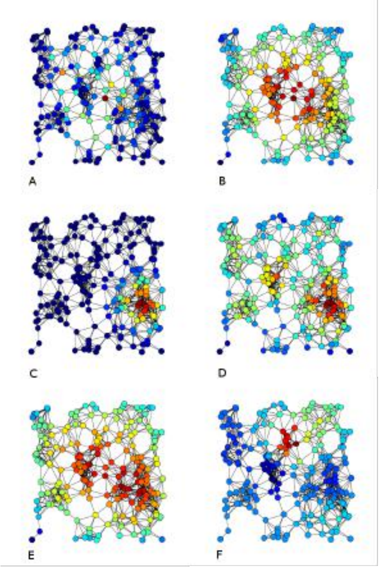


p

Random graph models comparison

	Random	BA model	WS model	Empirical networks
P(k)	$\frac{\lambda^k e^{-\lambda}}{k!}$	k^{-3}	poisson like	power law
C	$\langle k \rangle / N$	$N^{-0.75}$	const	large
$\langle L \rangle$	$\frac{\log(N)}{\log(\langle k \rangle)}$	$\frac{\log(N)}{\log\log(N)}$	$\log(N)$	small

Centrality examples

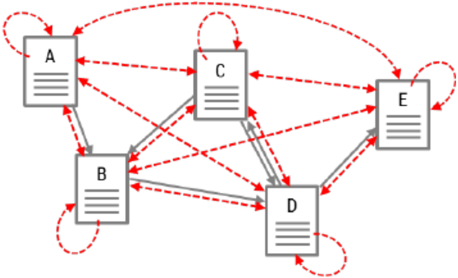


from Wikipedia

- A) Betweenness centrality
- B) Closeness centrality
- C) Eigenvector centrality
- D) Degree centrality
- F) Harmonic centrality
- E) Katz centrality

PageRank

"PageRank can be thought of as a model of user behavior. We assume there is a "random surfer" who is given a web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The **probability** that the random surfer visits a page is its **PageRank**."



Sergey Brin and Larry Page, 1998

PageRank formulation

- Power iterations:

$$\mathsf{p} \leftarrow \alpha \mathsf{P}^T \mathsf{p} + (1-\alpha)\frac{\mathsf{e}}{n}, \quad \alpha \text{ - teleportation coefficient}$$

- Sparse linear system:

$$(\mathsf{I} - \alpha \mathsf{P}^T)\mathsf{p} = (1-\alpha)\frac{\mathsf{e}}{n}$$

- Eigenvalue problem ($\lambda = 1$):

$$\left(\alpha \mathsf{P}^T + (1-\alpha)\mathsf{E}\right)\mathsf{p} = \lambda \mathsf{p}$$

$$\mathsf{P} = \mathsf{D}^{-1}\mathsf{A}$$

PageRank variations

- Power iterations

$$p \leftarrow \alpha P^T p + (1-\alpha)v, \quad v \text{ - teleportation vector}$$

$$P' = \alpha P + (1-\alpha)ev^T$$

$$p \leftarrow P'^T p, \ ||p|| = 1$$

- Topic specific PageRank

v - set of pages on specific topics

- TrustRank

v - set of trusted pages

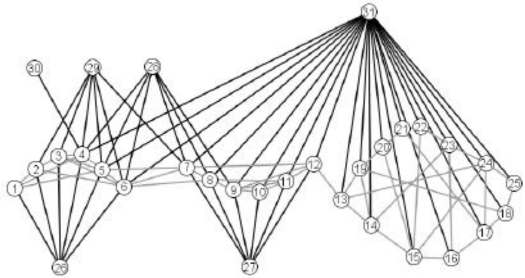
- Personalized PageRank

v - set of personal preference pages

Structural similarity

Definition

Two nodes are similar to each other if they share many neighbors.



Similarity measures

- Jaccard similarity

$$J(v_i, v_j) = \frac{|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|}{|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)|}$$

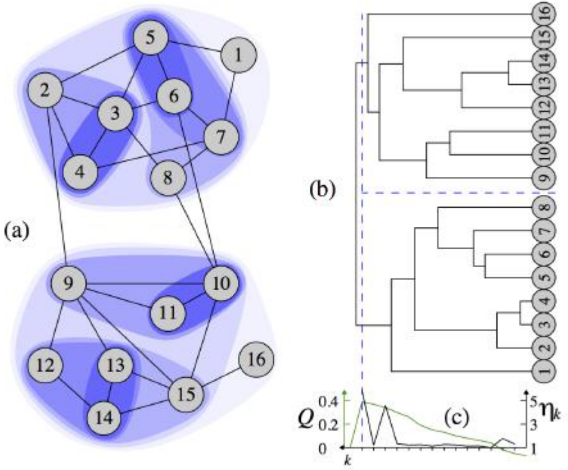
- Cosine similarity (vectors in *n*-dim space)

$$\sigma(v_i, v_j) = cos(\theta_{ij}) = \frac{\mathrm{v}_i^T \mathrm{v}_j}{|\mathrm{v}_i||\mathrm{v}_j|} = \frac{\sum_k A_{ik} A_{kj}}{\sqrt{\sum A_{ik}^2}\sqrt{\sum A_{jk}^2}}$$

- Pearson correlation coefficient:

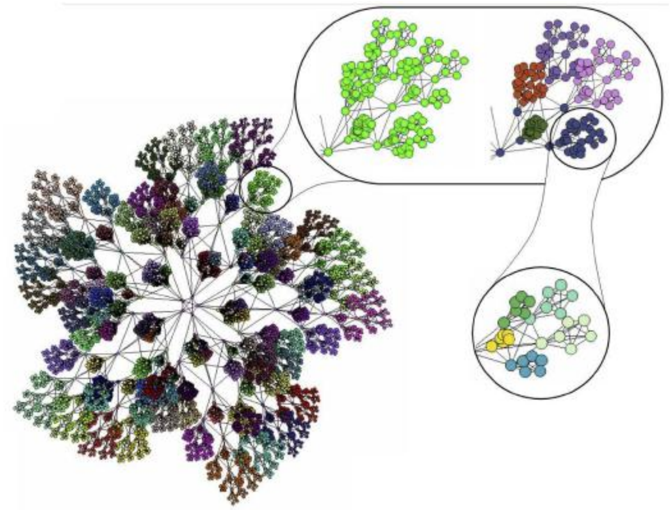
$$r_{ij} = \frac{\sum_k (A_{ik} - \langle A_i \rangle)(A_{jk} - \langle A_j \rangle)}{\sqrt{\sum_k (A_{ik} - \langle A_i \rangle)^2}\sqrt{\sum_k (A_{jk} - \langle A_j \rangle)^2}}$$

Walktrap



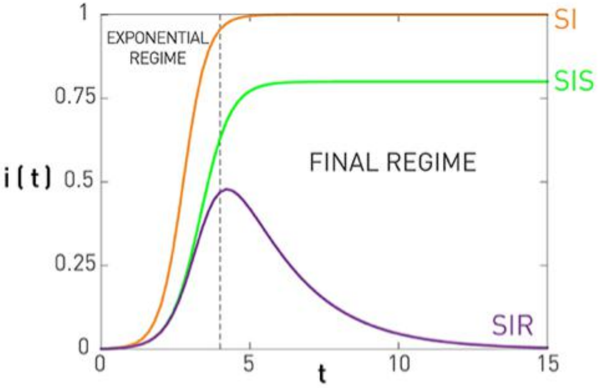
P. Pons and M. Latapy, 2006

Fast community unfolding



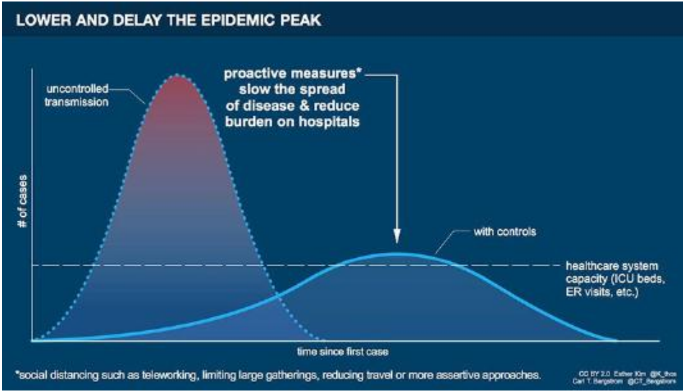
V. Blondel et.al., 2008

Compartmental models summary



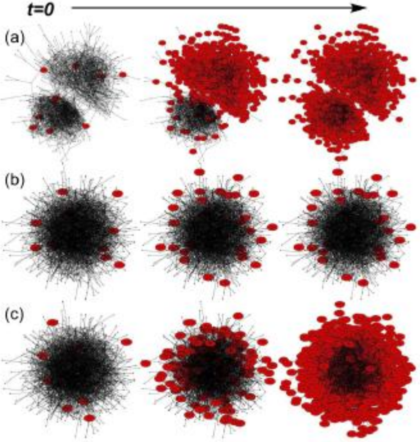
from Barabasi, 2016

Flatten the curve!



Cascades in random networks

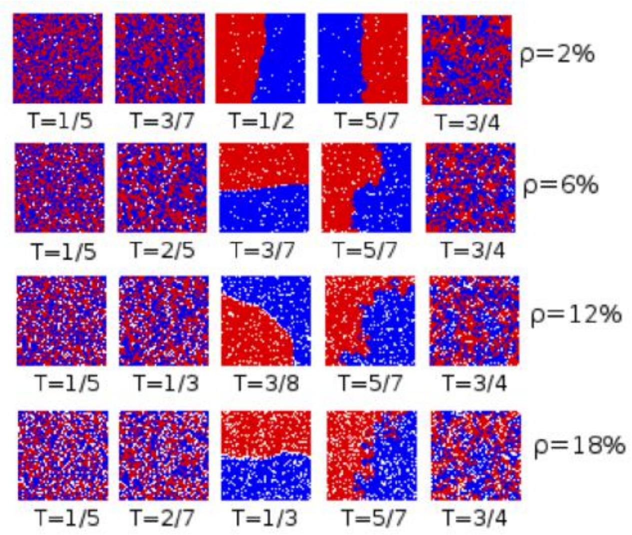
multiple seed nodes



(a) Empirical network; (b), (c) - randomized network

P. Singh, 2013

Spatial segregation



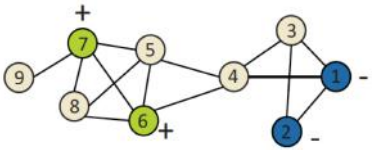
L. Gauvin et.al. 2009

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 - 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



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

$Y^{(t+1)} \leftarrow P \cdot Y^{(t)}$

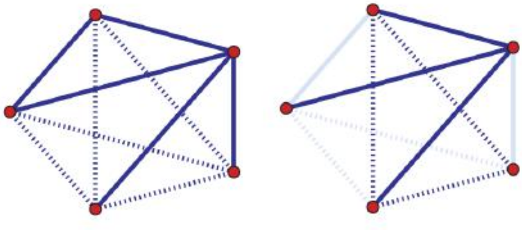
$Y_l^{(t+1)} \leftarrow Y_l^{(t)}$

until $Y^{(t)}$ *converges*;

$\hat{Y} \leftarrow Y^{(t)}$

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

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 $O(\frac{1}{|V|^2})$

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:

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

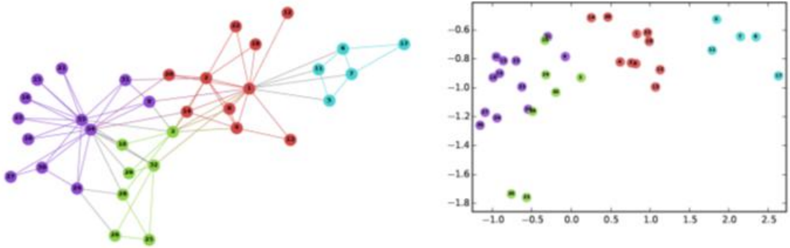
  Adamic/Adar:

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

Liben-Nowell and Kleinberg, 2003

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

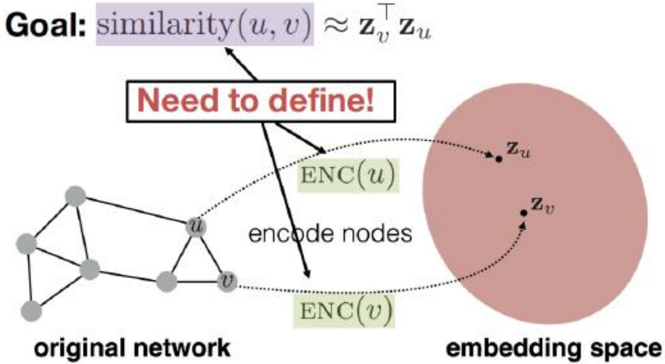


from Leskovec et al., 2018[1]

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

Graph Embeddings

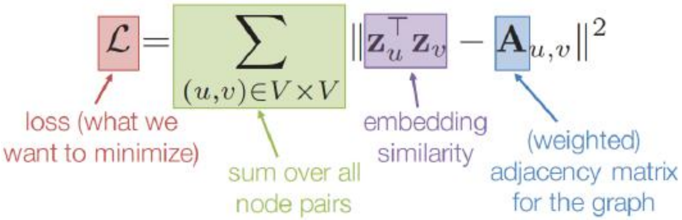
- Define **Encoder**
- Define **Similarity**/graph feature to preserve graph properties
- Define similarity/distance in the embedding space
- **Optimize** loss to fit embedding with similarity computed on graph



from Leskovec et al., 2018

First-order Proximity

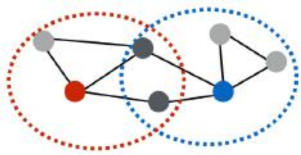
- Similarity between u and v is A_{uv}
- MSE Loss
- Variant of Matrix Decomposition



from Leskovec et al., 2018

Multi-order Proximity

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



$$\mathcal{L} = \sum_{(u,v)\in V\times V} \| \underbrace{\boxed{\mathbf{z}_u^\top \mathbf{z}_v}}_{\text{embedding similarity}} - \underbrace{\boxed{\mathbf{S}_{u,v}}}_{\text{multi-hop network similarity (i.e., any neighborhood overlap measure)}} \|^2$$

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

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

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

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

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k} \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

random distribution over all nodes

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

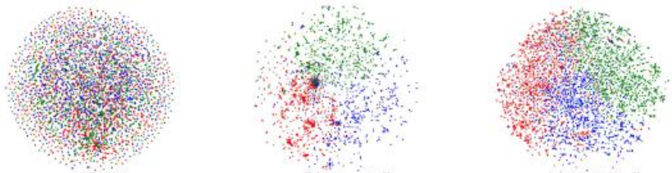
from Makarov et al., 2019

Random Walk Strategies

- Finite unbiased random walks (DeepWalk)
- 1-hops & 2-hops for half of the embedding, arbitrary random walks for the second half of the embedding (LINE)
- Diffusion for sampling (Diff2Vec)
- Biased random walks combining BFS and DFS (Node2Vec)

DeepWalk & LINE

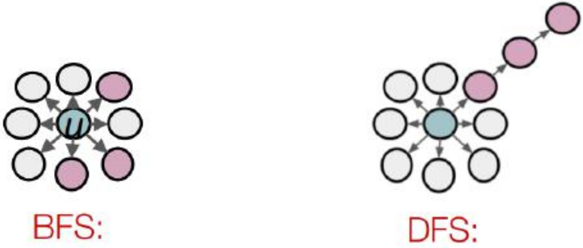
- DeepWalk: Unbiased random walks with fixed length and number
- LINE: Combination of first-order and second-order proximity aggregation via concatenation



(a) GF (b) DeepWalk (c) LINE(2nd)

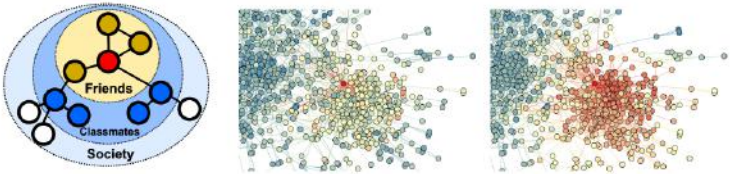
Node2Vec

- BFS samples local neighborhood, DFS goes for global features



- Two parameters p and q to control sampling
- Second order Markov process

- Grarep: Approximate normalized A^k efficiently
- Walklets: approximate attention over k-distance neighbors



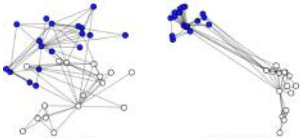
(a) A student (in red) is a member of several increasing larger social communities.

(b) WALKLETS Fine Representation

(c) WALKLETS Coarse Representation

GEMSEC

- GEMSEC model cluster information adding regularization term over community labels given number of classes
- parameter γ balance cluster error and cluster structure given model hyper-parameters



(a) DeepWalk (b) GEMSEC

$$\mathcal{L} = \underbrace{\sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]}_{\text{Embedding cost}} + \underbrace{\gamma \cdot \sum_{v \in V} \min_{c \in C} \| f(v) - \mu_c \|_2}_{\text{Clustering cost}}. \qquad (5)$$

VERSE for Structural Similarity

- VERSE uses conditional probability not on the embedding, but on the similarity rank.
- $O(|E|)$ performance

$$sim_{\mathrm{E}}(v,\cdot) = \frac{\exp(W_v W^\top)}{\sum_{i=1}^{n} \exp(W_v \cdot W_i)}$$

$$\mathcal{L} = -\sum_{v \in V} sim_{\mathrm{G}}(v,\cdot) \log\left(sim_{\mathrm{E}}(v,\cdot)\right)$$

$$\mathcal{L}_{NCE} = \sum_{\substack{u \sim \mathcal{P} \\ v \sim sim_{\mathrm{G}}(u,\cdot)}} \Big[\log \Pr_W(D = 1 | sim_{\mathrm{E}}(u,v)) + s\mathbb{E}_{\widetilde{v} \sim Q(u)} \log \Pr_W(D = 0 | sim_{\mathrm{E}}(u,\widetilde{v})) \Big]$$

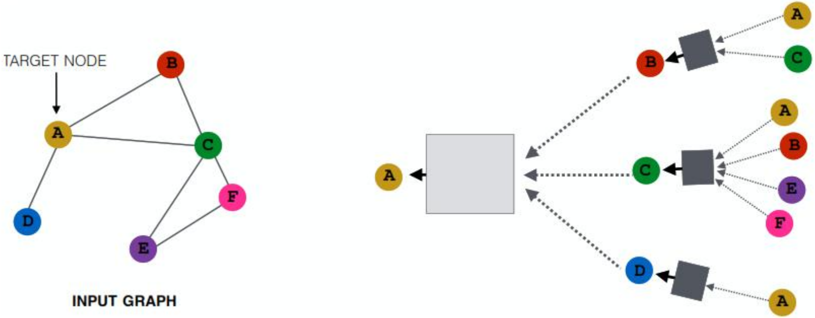
GNN

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

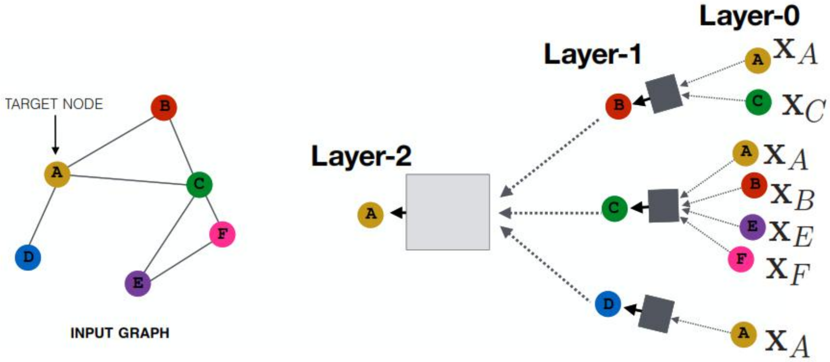
- Assign weights only to information obtained from neighbors
- Include node itself via loop with trainable weight
- Each node generate its own computational graph



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

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

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

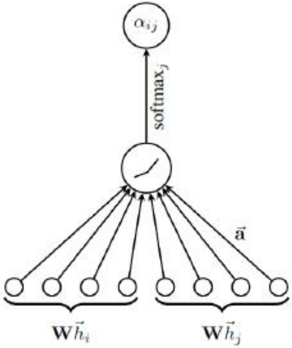
$$\vec{h}'_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

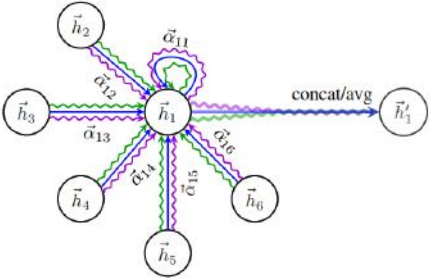
$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

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

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

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.

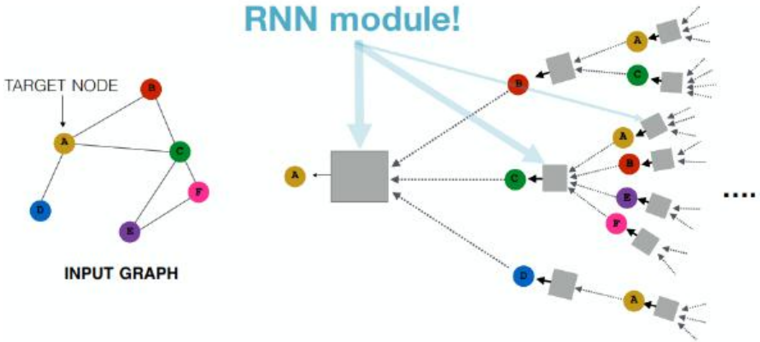
$$\mathrm{AGG} = \mathrm{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

from Leskovec et al., 2018

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:

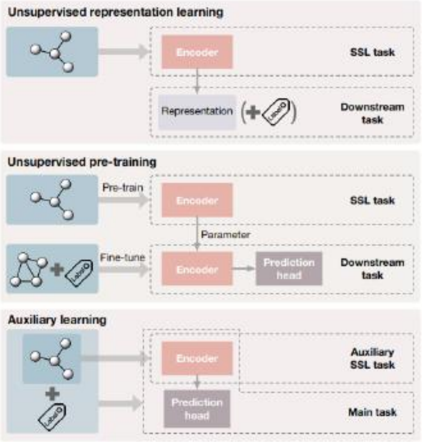
$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

from Leskovec et al., 2018

Already too complex?
What comes next?

Self-supervised Learning

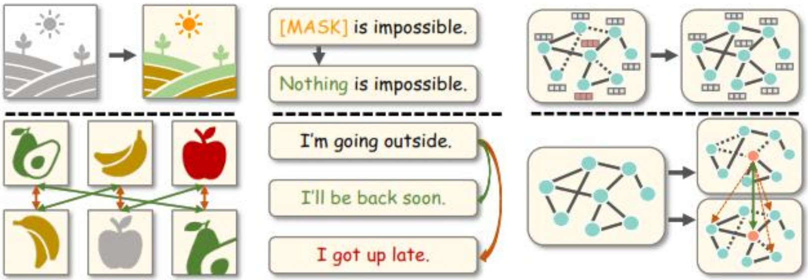
- Train data representation before using it in downstream tasks



from Ji et al., 2021

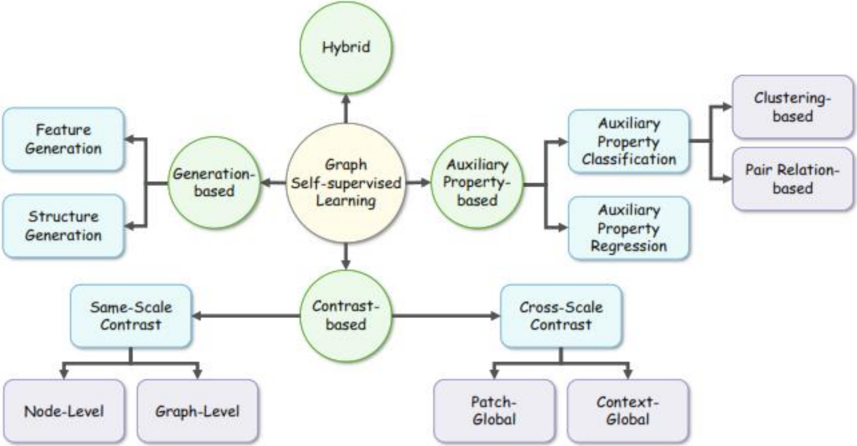
Self-supervised Learning

- Various data type lead to different data augmentation strategies



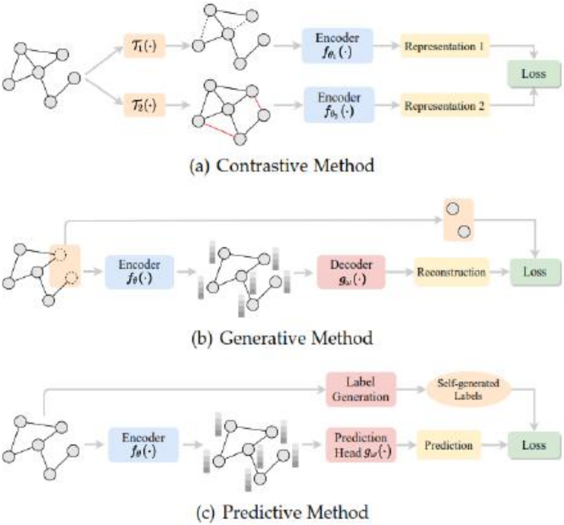
from Zhou et al., 2021

SSL for Graphs



from Zhou et al., 2021

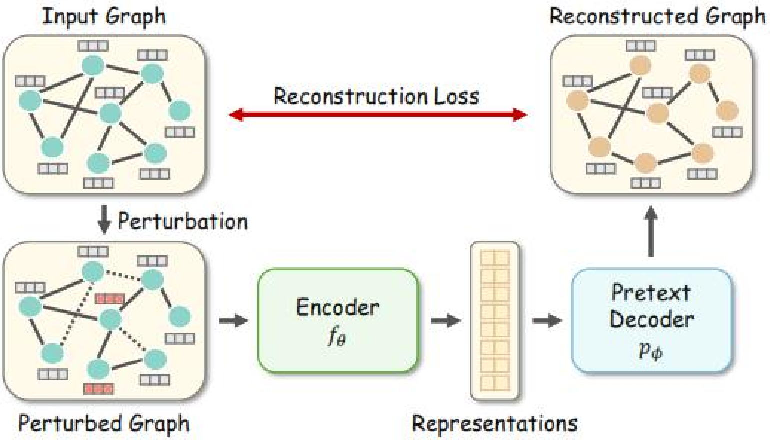
SSL for Graphs



from Li S. et al., 2021

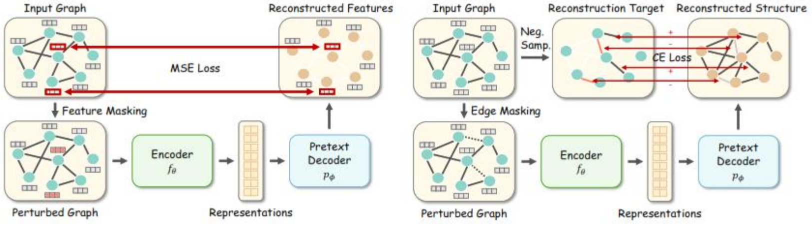
1. Generative SSL

- Use generative models trained via reconstruction loss
- Model input is generated by (optional) graph perturbation. Decoder tries to recover the original graph



1. Generative SSL methods

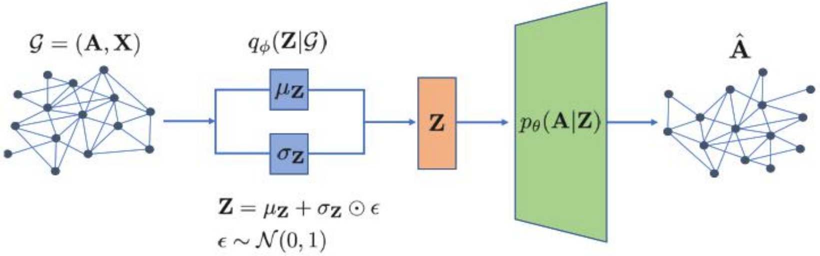
- Attribute or structure reconstruction/prediction



from Zhou et al., 2021

1. Generative SSL: Variational Graph Auto-Encoder

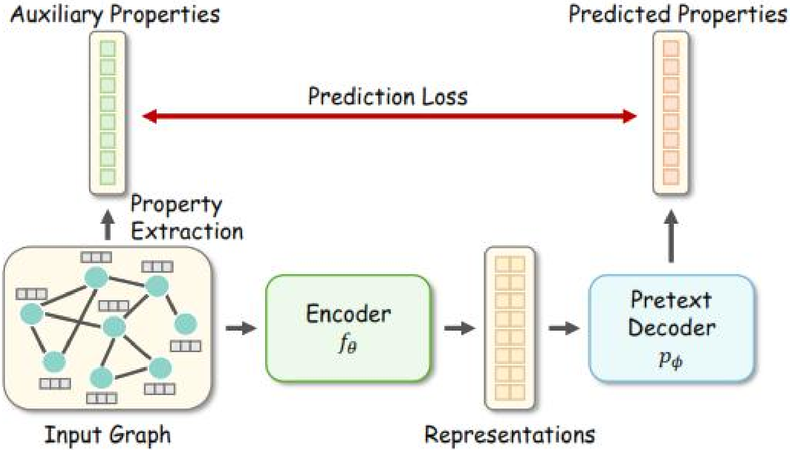
- VGAE is a variational counter-part of GAE
- Instead of direct reconstruction in GAE via LPP, VGAE learns a conditional distribution over adjacency matrices generated from embeddings



from Hamilton W.L., 2020

2. Predictive SSL

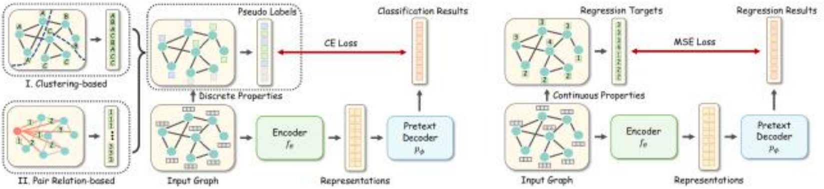
- Auxiliary properties are extracted from graphs freely
- Classification or regression is trained via CE/MSE loss



from Zhou et al., 2021

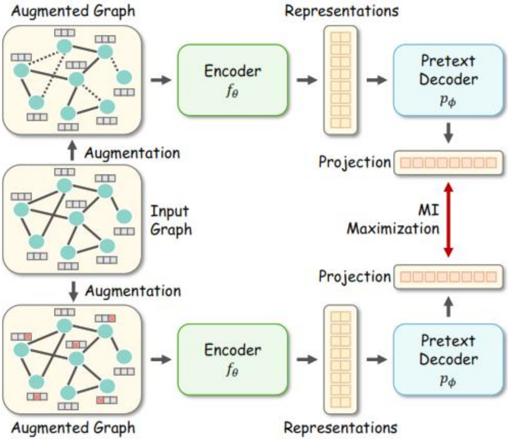
2. Predictive SSL methods

- Clustering/similarity/centrality/prestige based labels for CE/MSE loss



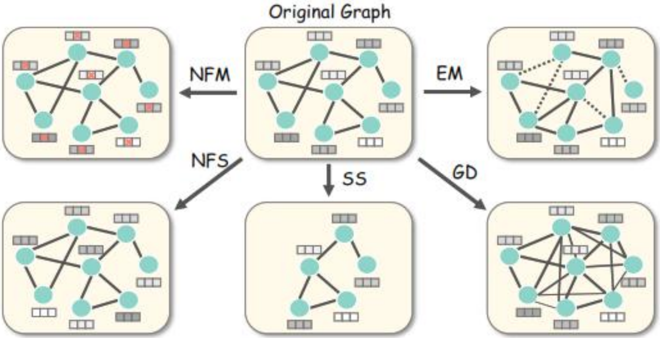
from Zhou et al., 2021

- Two different augmented views are constructed from the original graph.
- Model is trained by maximizing the Mutual Information (MI) between two views.



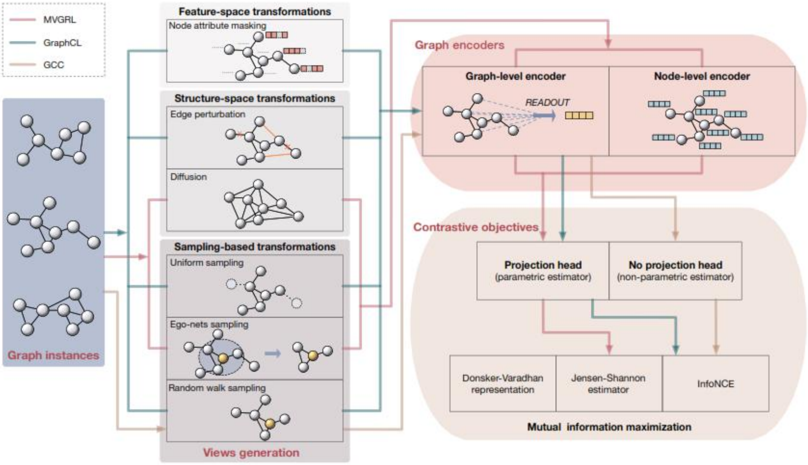
3. Contrastive SSL: Graph augmentations

- Typical graph augmentations: Node Feature Masking (NFM), Node Feature Shuffle (NFS), Edge Modification (EM), Graph Diffusion (GD), and Subgraph Sampling (SS).



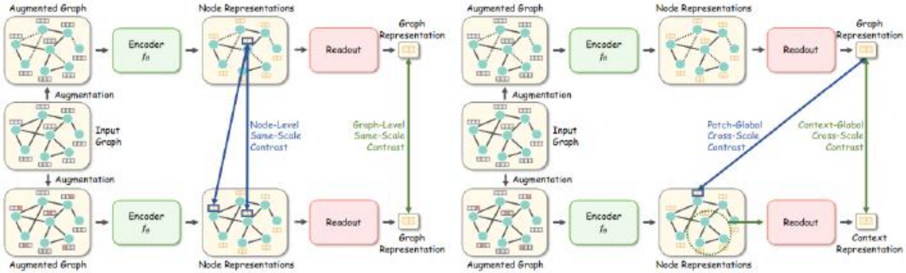
from Zhou et al., 2021

- Different combinations of augmentations



Ji et al., 2021

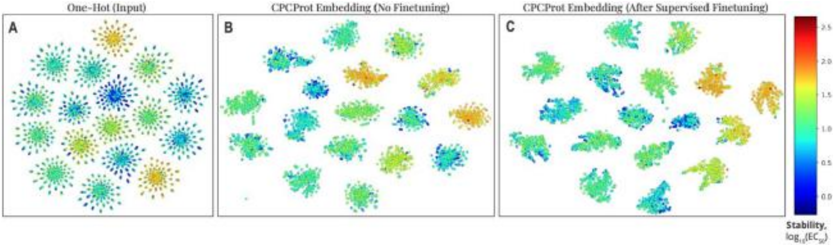
- Hierarchical augmentations on different graph levels



from Zhou et al., 2021

3. Contrastive SSL methods: SSL for Protein sequences

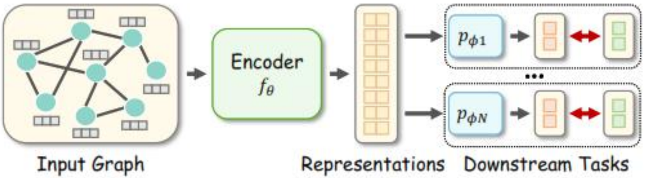
- Encoding protein sequences one can use contrastive learning to group common sequences by either, sequential patterns of predicted labels from downstream tasks



from Moses et al., 2020

4. Hybrid SSL methods

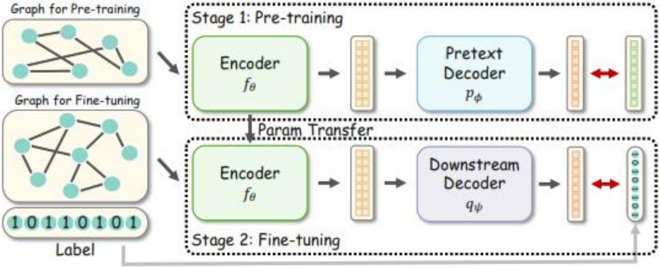
- Multiple pretext tasks are designed to train the model together in a multi-task learning manner.



from Zhou et al., 2021

Training SSL models: Pretrain and Finetune

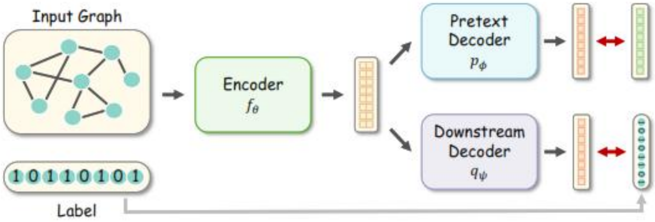
- Encoder is pretrained with pretext tasks in an unsupervised manner.
- Pretrained parameters are leveraged as the initial parameters in the fine-tuning phase for downstream tasks.



from Zhou et al., 2021

Training SSL models: Joint Learning

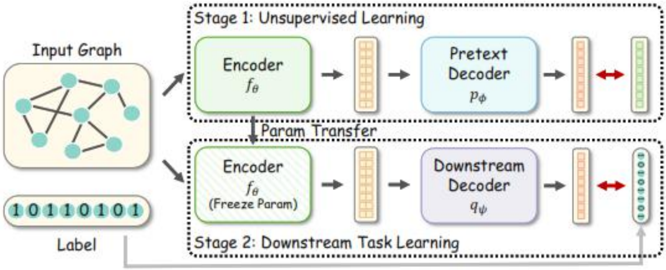
- The model is trained with pretext and downstream tasks simultaneously in a multi-task learning setting.



from Zhou et al., 2021

Training SSL models: Unsupervised Learning

- Training encoder with pretext tasks.
- Use obtained representations to learn the downstream decoder in the second phase.

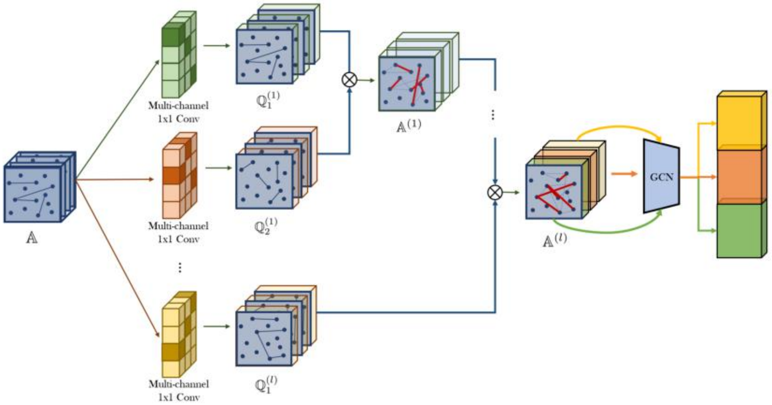


from Zhou et al., 2021

What about transformers?

Graph Transformers

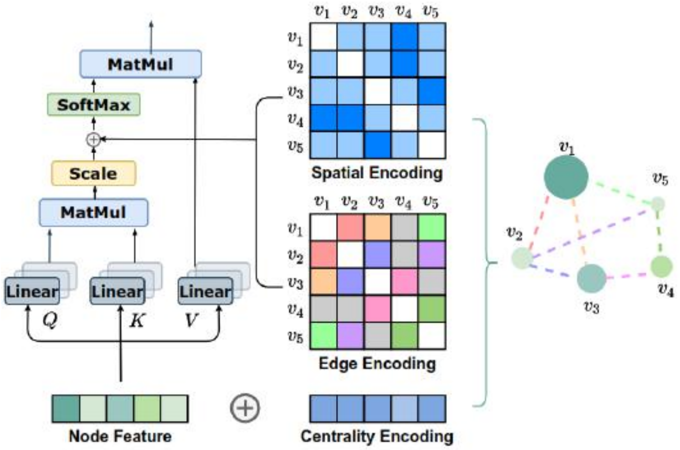
- Heterogeneous graph aggregation
- SSL Pretraining



from Kim et al., 2019

Graph Transformers

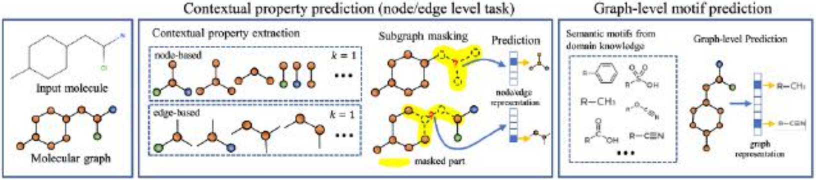
- Node and edge encodings
- Positional and similarity/distance encodings



from Liu et al., 2021

SSL for Graph Transformers

- Large molecules dataset
- Physics-/Chemistry- inspired augmentations make more sense than arbitrary augmentations
- Road towards Graph SSL Transformers!



from Huang et al., 2020

References

- Liu, Y., Pan, S., Jin, M., Zhou, C., Xia, F. and Yu, P.S., 2021. Graph self-supervised learning: A survey. arXiv preprint arXiv:2103.00111.
- Xie, Y., Xu, Z., Zhang, J., Wang, Z. and Ji, S., 2021. Self-supervised learning of graph neural networks: A unified review. arXiv preprint arXiv:2102.10757.
- Wu, L., Lin, H., Gao, Z., Tan, C. and Li, S., 2021. Self-supervised on Graphs: Contrastive, Generative, or Predictive. arXiv preprint arXiv:2105.07342.
- Hamilton, W.L., 2020. Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning, 14(3), pp.1-159.

References

- Lu, A.X., Zhang, H., Ghassemi, M. and Moses, A.M., 2020. Self-supervised contrastive learning of protein representations by mutual information maximization. BioRxiv.
- Yun, S., Jeong, M., Kim, R., Kang, J. and Kim, H.J., 2019. Graph transformer networks. Advances in Neural Information Processing Systems, 32, pp.11983-11993.
- Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y. and Liu, T.Y., 2021. Do Transformers Really Perform Bad for Graph Representation?. arXiv preprint arXiv:2106.05234.
- Rong, Y., Bian, Y., Xu, T., Xie, W., Wei, Y., Huang, W. and Huang, J., 2020. Self-supervised graph transformer on large-scale molecular data. arXiv preprint arXiv:2007.02835.