ETH zürich

BIOMEDICAL INFORMATICS

GNN Graph Manipulation & Self-supervised Learning

Dr. Rita Kuznetsova
Institute for Machine Learning, Department of Computer Science

Outline for Today

1st slot - Lecture:

1. Graph Manipulation
   a. Motivation
   b. Feature Augmentation
   c. Structure Augmentation
2. Self-supervised Learning for GNNs
   a. Motivation & Overview
   b. Contrastive Learning
   c. Predictive Learning
3. Take-home Messages

2nd slot: Paper Presentation

Graph Manipulation

Why Manipulate on Graphs

Graph Feature manipulation

- The input graph **lacks features** → **feature augmentation**

Graph Structure manipulation

- The graph is **too sparse** → Add virtual nodes / edges
- The graph is **too dense** → Sample neighbors when doing message passing
- The graph is **too large** → Sample subgraphs to compute embeddings

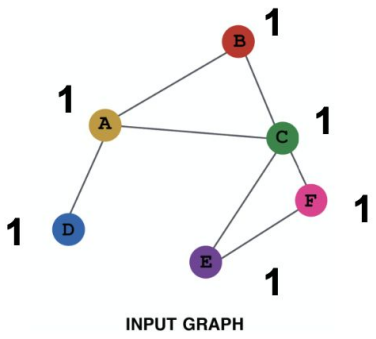
Credit: [1]

Feature Augmentation on Graphs

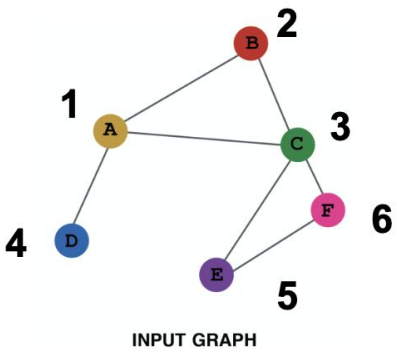
Problem: Input graph does not have node features.

Standard approaches:

1. Assign constant values to nodes.



2. Use one-hot encoding.

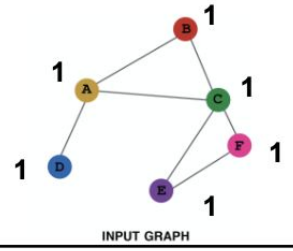
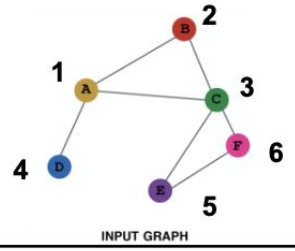


One-hot vector for node with ID = 5:

[0, 0, 0, 0, 1, 0]

Credit: [1]

Constant vs One-hot

	Constant node feature	One-hot node feature
		
Expressive power	**Medium**. All the nodes are identical, but GNN can still learn from the graph structure	**High**. Each node has a unique ID, so node-specific information can be stored
Inductive learning (Generalize to unseen nodes)	**High**. Simple to generalize to new nodes: we assign constant feature to them, then apply our GNN	**Low**. Cannot generalize to new nodes: new nodes introduce new IDs, GNN doesn't know how to embed unseen IDs
Computational cost	**Low**. Only 1 dimensional feature	**High**. High dimensional feature, cannot apply to large graphs
Use cases	Any graph, inductive settings (generalize to new nodes)	Small graph, transductive settings (no new nodes)

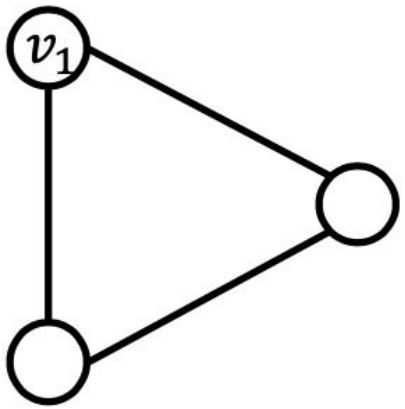
Credit: [1]

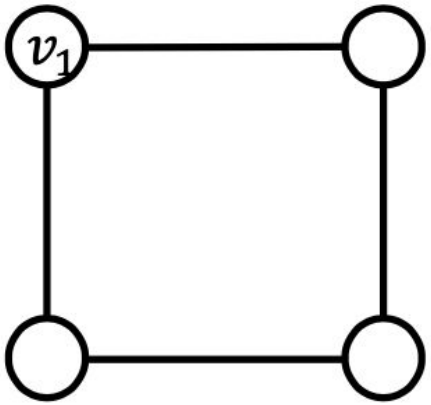
Feature Augmentation on Graphs

Problem: Certain structures are hard to learn by GNN.

Example: Cycle count feature.

- Can GNN learn the length of a cycle that $v1$ resides in?



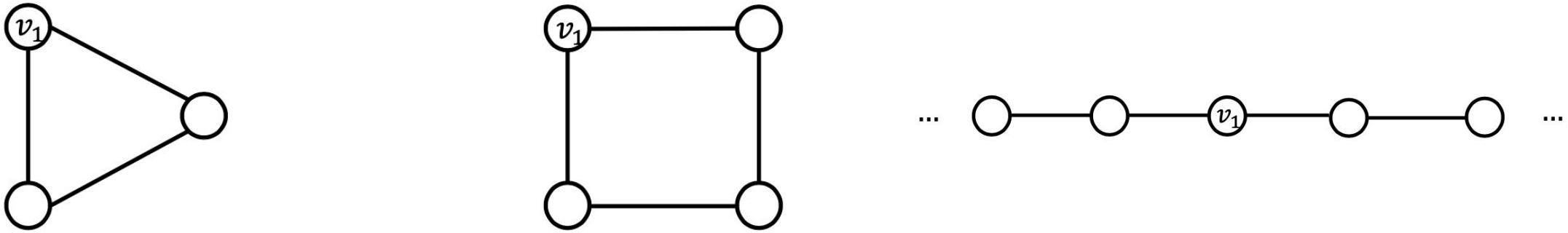


Feature Augmentation on Graphs

Problem: Certain structures are hard to learn by GNN.

Example: Cycle count feature.

- Can GNN learn the length of a cycle that $v1$ resides in?



The computational graphs for node $\boldsymbol{v1}$ are always the same!

Credit: [1]

Feature Augmentation on Graphs

Problem: Certain structures are hard to learn by GNN.

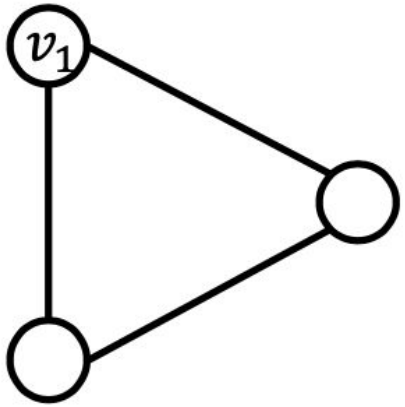
Example: Cycle count feature.

- cycle count as augmented node features.

Augmented node feature for $v1$

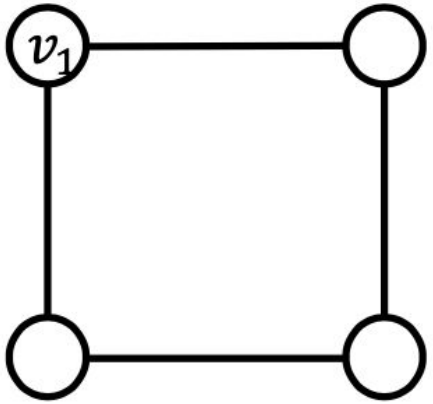
[1, 0, 0]

a cycle with length 3



Augmented node feature for $v1$

[0, 1, 0]



a cycle with length 4

Feature Augmentation on Graphs

Hand-crafted features (see Lecture 2):

1. Node degree,
2. Node centrality,
3. Number of triangles,
4. …

Other options:

- node2vec (not generalizable to the new nodes!)

Check the benchmark of the different approaches in [13].

Credit: [1, 13]

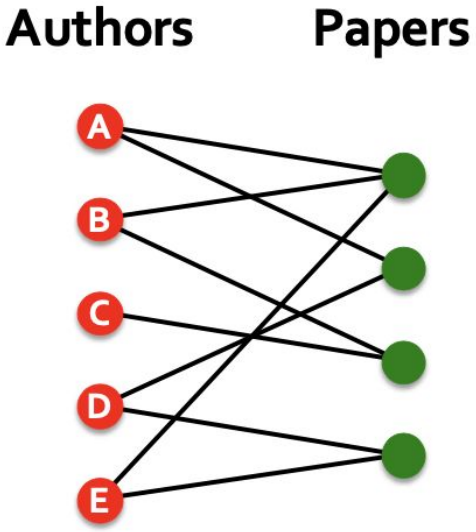
Add virtual nodes/edges

Problem: Graph is too sparse.

Approach: Connect 2-hop neighbors via virtual edges.

Use case:

- Author-to-papers (they authored).
- 2-hop virtual edges make an author-author collaboration graph.



Expander Graph Propagation

Andreea Deac Marc Lackenby Petar Veličković

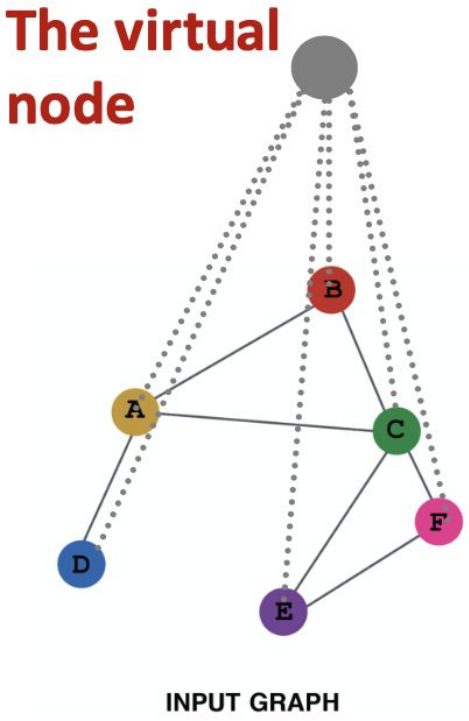
Credit: [1]

Add virtual nodes/edges

Problem: Graph is too sparse.

Approach: add virtual nodes.

- Shorten the distance between nodes.
- Greatly improves message passing in sparse graphs.

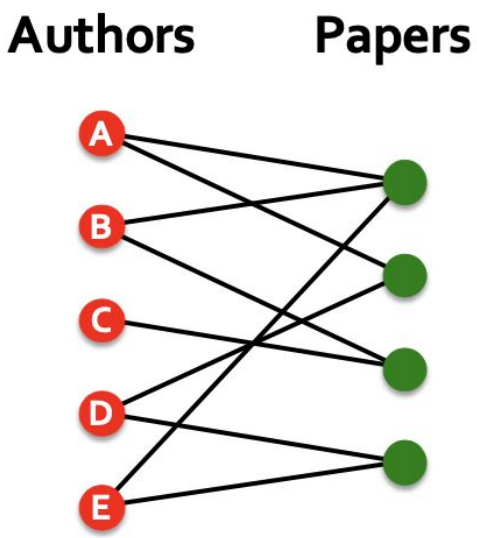


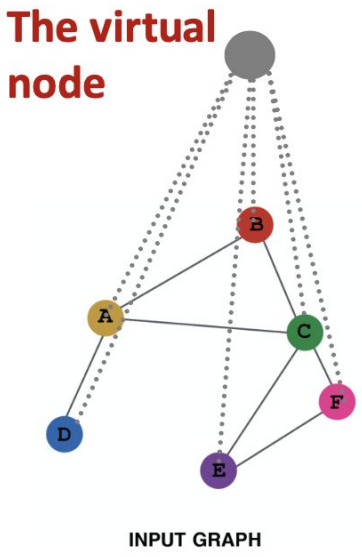
Credit: [1]

Add virtual nodes/edges

Problem: Graph is too sparse.

Approach: add virtual nodes/edges.





Breaks the graph structure!

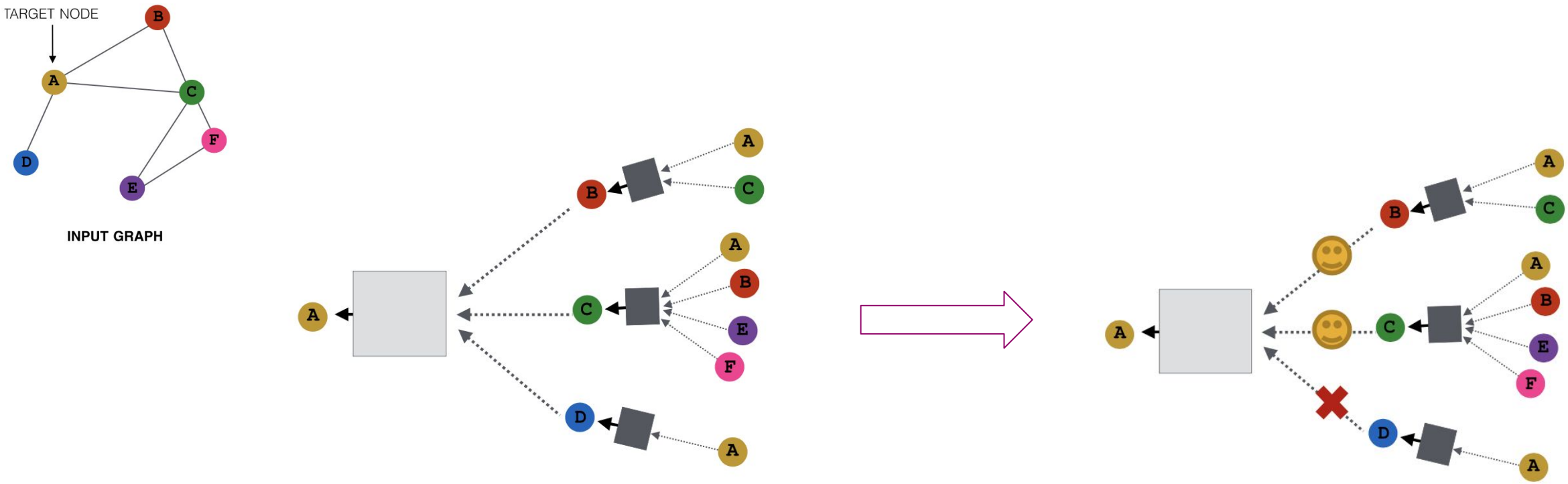
Hint: use message passing on the original graph, then employ virtual nodes/edges.

Credit: [1]

Add virtual nodes/edges

Problem: Graph is too dense.

Approach: sample the node's neighbourhood for the message passing (GraphSAGE).



Credit: [1]

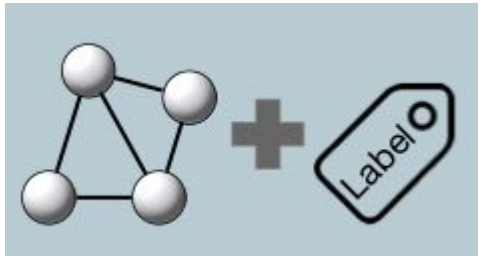
Self-supervised Learning for GNNs

Motivation

For supervised training, a large number of labels are required, which make it inapplicable in many real- world scenarios.

GNNs:

- mostly require task-dependent labels to learn rich representations.
- annotating graphs is challenging.
- labeling graphs procedurally using domain knowledge is costly.



Credit: [2]

Motivation

Self-supervised learning (SSL) enables the training of deep models on unlabeled data, removing the need of excessive annotated labels.

No labeled data

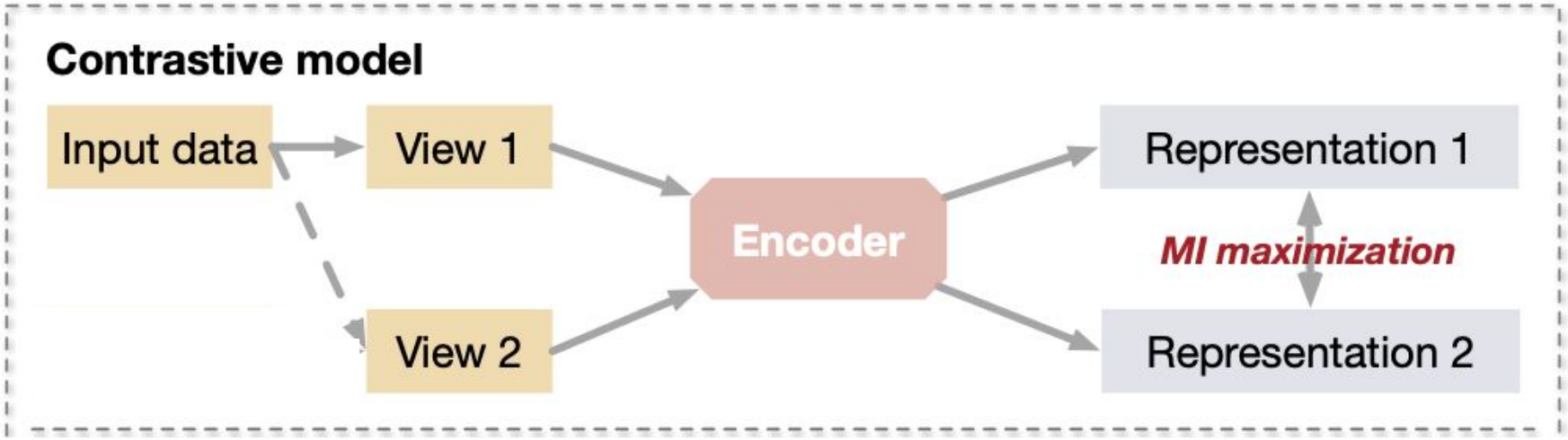
SSL serves as an approach to learn representations from unlabeled data itself.

Limited labeled data

SSL from unlabeled data can be used as a pre-training process. After labeled data are used to fine-tune the pre-trained models for downstream tasks.

Credit: [2]

SSL methods: Contrastive model vs Predictive model

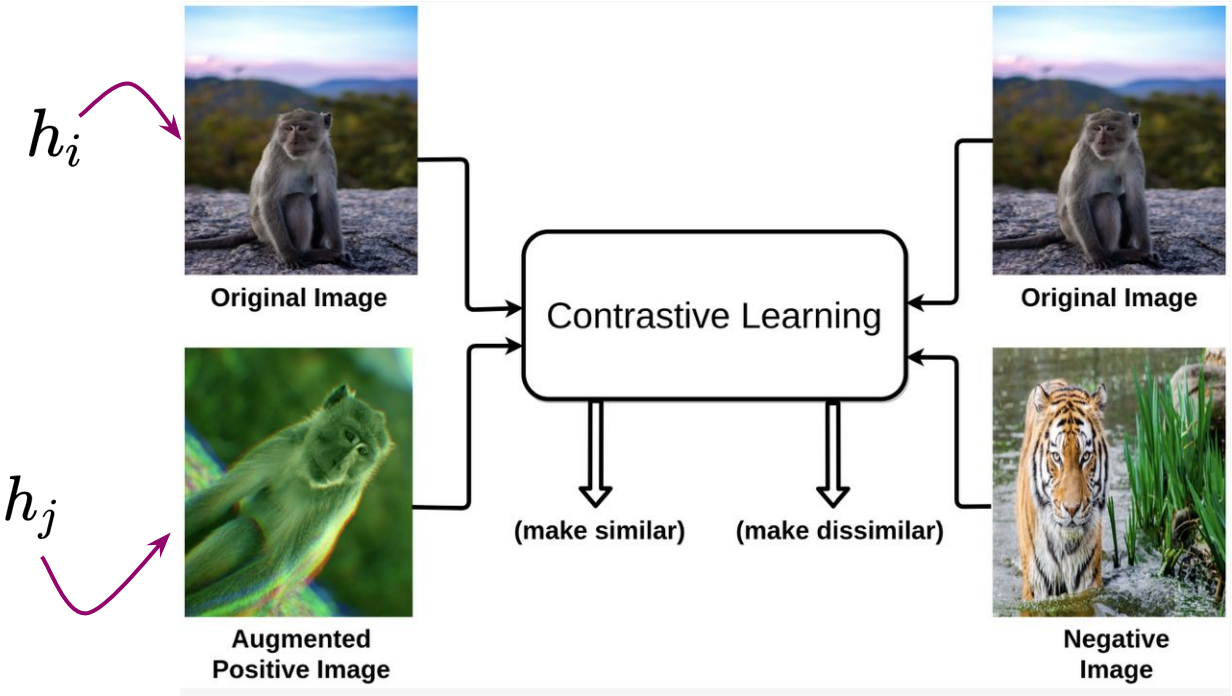


Credit: [2]

Contrastive Learning

Contrastive Learning

Self-supervised learning approach that relies on data augmentation to construct pairs/views of samples that share the same semantics.



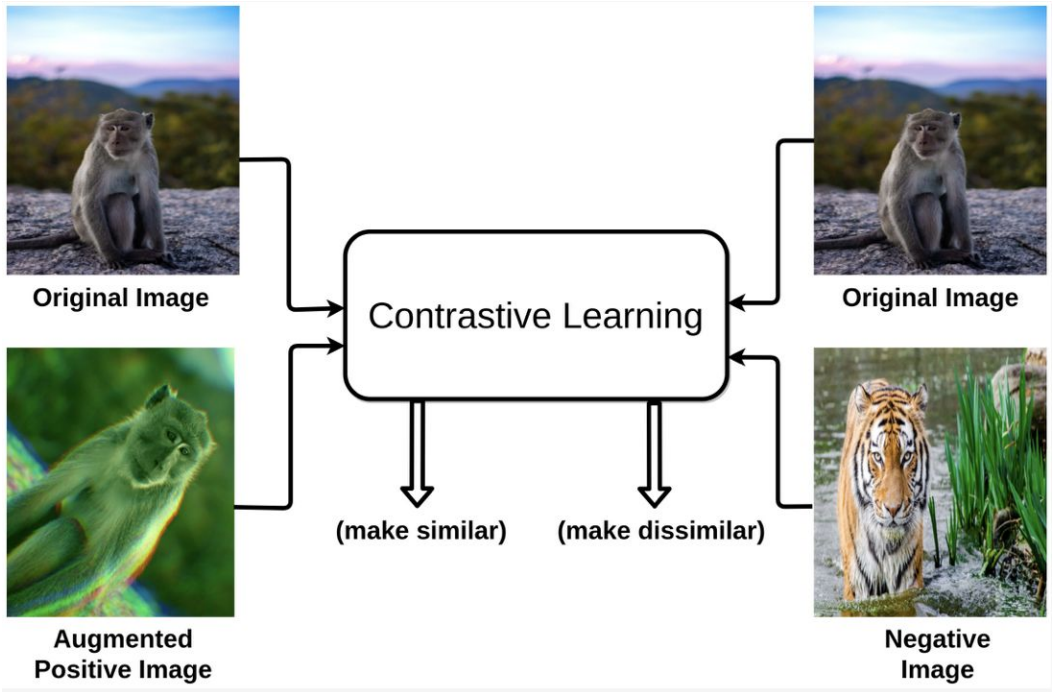
Given $\boldsymbol{x}$, we create representations $\boldsymbol{h}$:

$$\mathcal{L}_{InfoNCE} = -\frac{1}{N+1}\sum_{x\in B}\log\frac{\exp\left(\mathrm{sim}(h_i,\, h_j)\right)}{\sum_{x'\,\in\, B\setminus\{x\}}\exp\left(\mathrm{sim}\left(h_i,\, h'_j\right)\right)}$$

Credit: [3]

Contrastive Learning

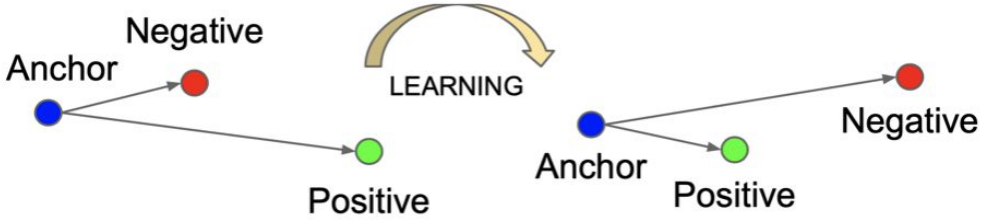
Self-supervised learning approach that relies on data augmentation to construct pairs/views of samples that share the same semantics.



1. $\mathcal{L}_{InfoNCE}$

2. JS estimator

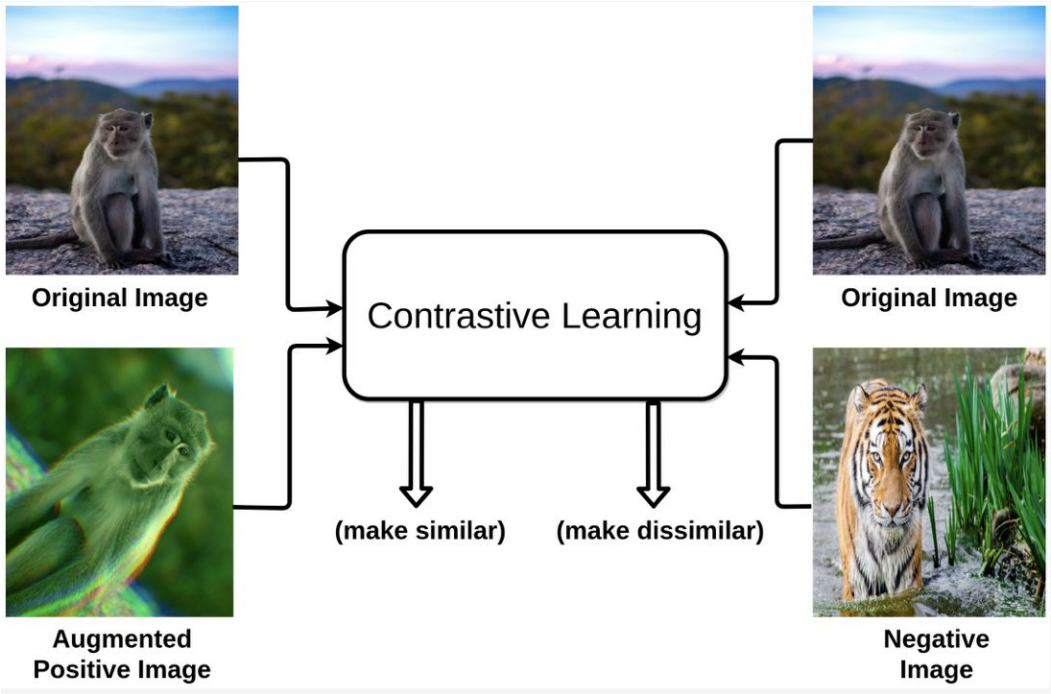
3. Triplet Loss



Credit: [2, 3]

Contrastive Learning

Self-supervised learning approach that relies on data augmentation to construct pairs/views of samples that share the same semantics.



Positive pairs:

1. Different augmentations of one object;
2. Original object and its augmentation.

Negative pairs:

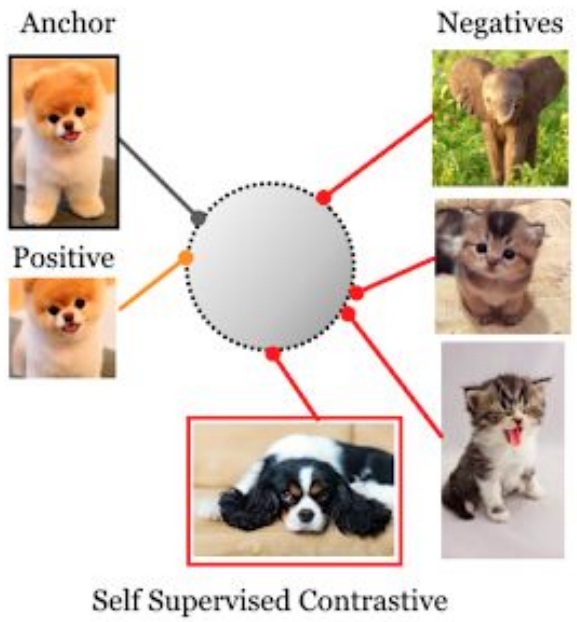
1. The rest of objects from the batch;
2. Sampling;
3. Apply corruption functions.

Credit: [2, 3, 14]

Contrastive Learning

Gained a lot of attention in the past couple of years for the generalisation properties of such representations.

Imaging data augmentation is easy - the data is humanly understandable and diverse.



What about Graphs?

Credit: [4]

Overview of Contrastive Learning Framework

Given a graph $(\boldsymbol{A}, \boldsymbol{X})$, key components that specify a contrastive learning framework:

- Transformations $\mathcal{T}_1, \ldots, \mathcal{T}_k$ that compute multiple views of the graph:

$$w_i = \mathcal{T}_i(A, X), i = 1, \ldots, k$$

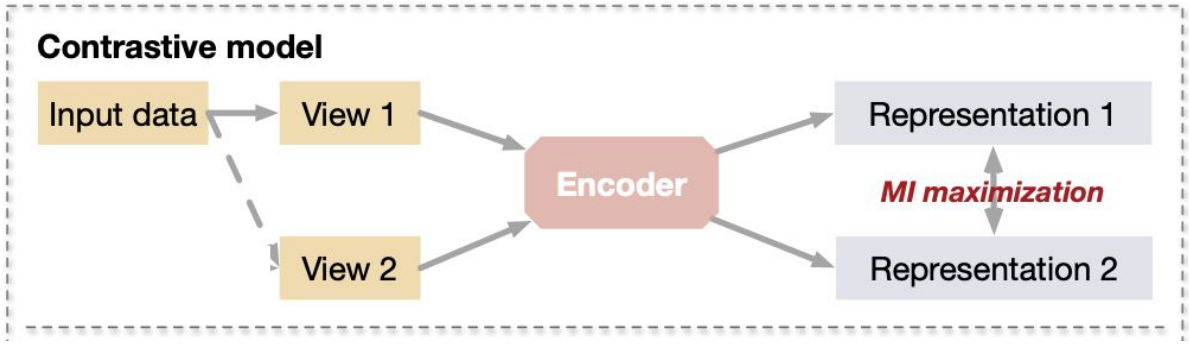
- Encoders $f_1, \ldots, f_k$ (or shared encoder f), that compute the representation for each view:

$$h_i = f_i(w_i), i = 1, \ldots, k$$

- Learning objective to optimize encoder parameters: maximize mutual information between between a pair of representations.

Credit: [2]

Graph View Generation



1. Feature transformations perform the transformation on the feature matrix $\boldsymbol{X}$:

$$\mathcal{T}_{feat}(A, X) = (A,\ \mathcal{T}_X(X)),\ \text{where}\ \mathcal{T}_X : \mathbb{R}^{|V|\times d} \rightarrow \mathbb{R}^{|V|\times d}$$

2. Structure transformations perform the transformation on the adjacency matrix $\boldsymbol{A}$:

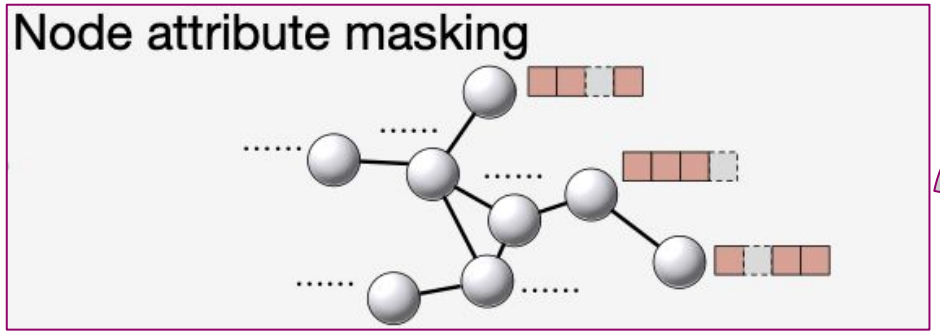
$$\mathcal{T}_{struct}(A, X) = (\mathcal{T}_A(A),\ X),\ \text{where}\ \mathcal{T}_A : \mathbb{R}^{|V|\times|V|} \rightarrow \mathbb{R}^{|V|\times|V|}$$

3. Sampling-based transformations …

$$\mathcal{T}_{\text{sample}}(A, X) = (A\,[S,\ S],\ X[S]),$$
$$\text{where}\ S \subseteq V;\ [\cdot]\ \text{selects certain rows (columns).}$$

Credit: [2]

Feature Transformations

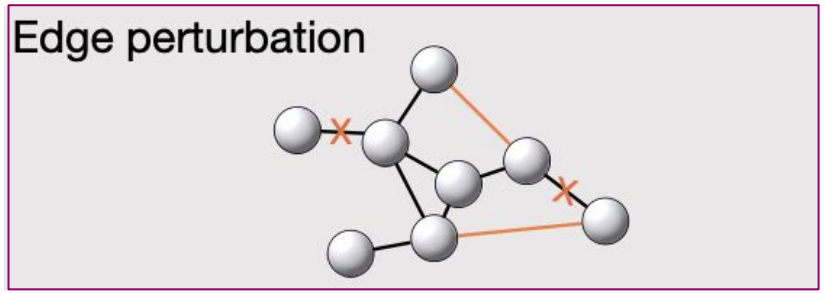


Adding Gaussian noise

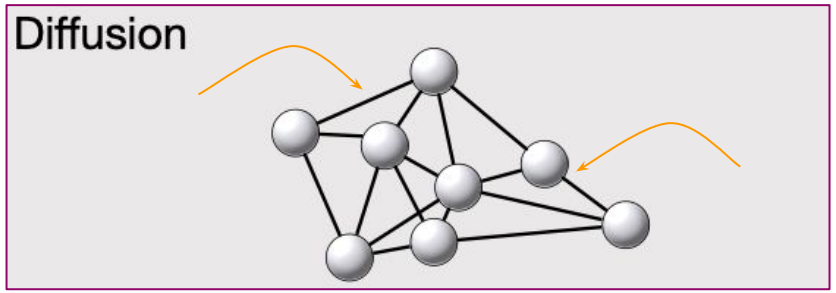
Enhance model robustness by encouraging invariance to input perturbations.

Credit: [2, 5]

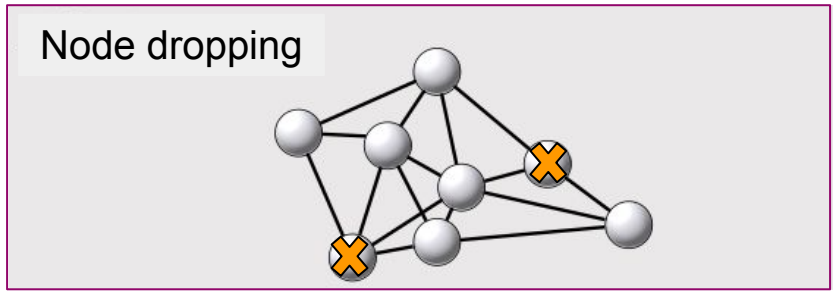
Structure Transformations



Randomly adds or drops edges in a given graph.



Adding new connections, sampled from diffusion matrix*. Enhances connectivity.

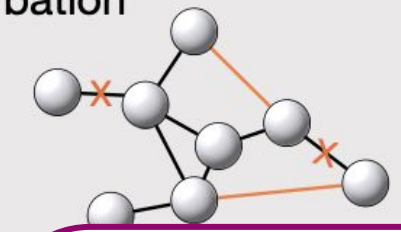


Randomly discard certain portion of vertices along with their connections.

*A diffusion matrix is formed by scaling a graph's adjacency matrix with its degree matrix to reflect transition probabilities for information diffusion. Credit: [2, 5, 6]

Structure Transformations

Edge perturbation

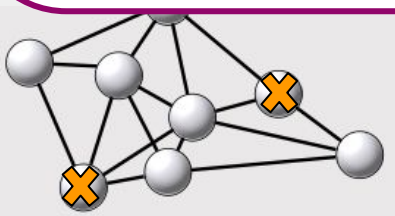


Randomly adds or drops edges in a given graph.

Diffusion

ections,
on matrix.
ctivity.

Node droppi

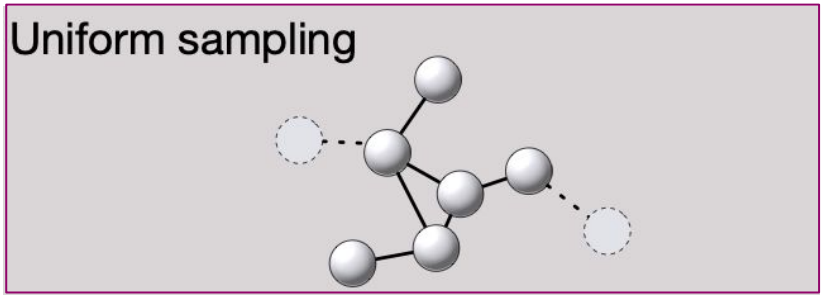


Randomly discard certain portion of vertices along with their connections.

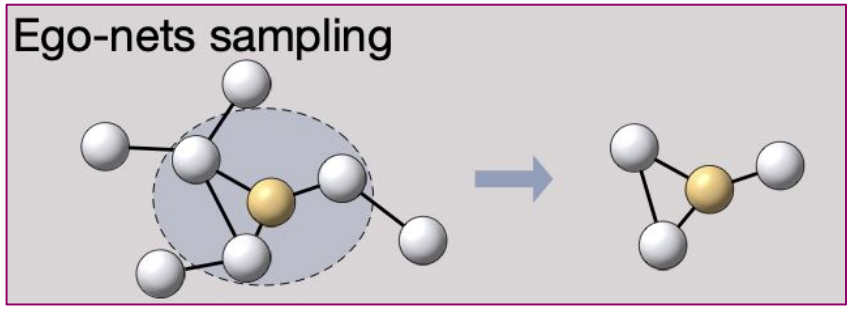
Semantic robustness against connectivity variations/missing vertices.

Credit: [2, 5, 6]

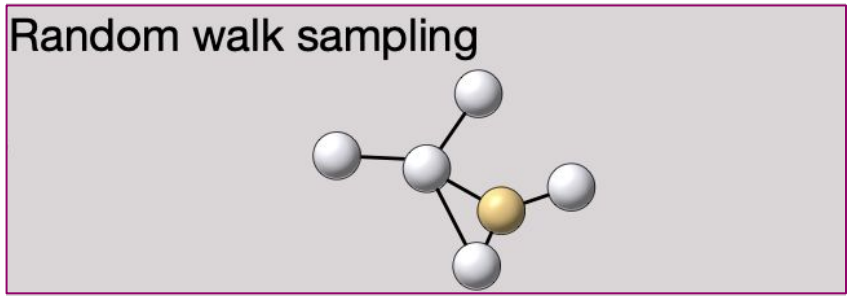
Sampling-Based Transformations



Sample sub-graphs by uniformly sampling a given number of nodes and edges of the sampled nodes.



Node-level representations from L-ego-net (node-level encoder with L layers).

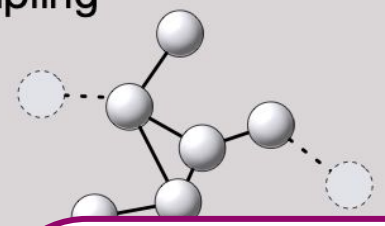


Sample sub-graphs based on random walks starting from a given node.

Credit: [2, 5, 6, 7, 8]

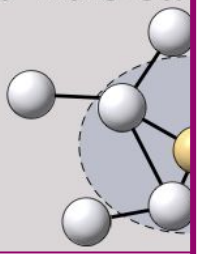
Sampling-Based Transformations

Uniform sampling



Sample sub-graphs by uniformly sampling a given number of nodes and edges of the sampled

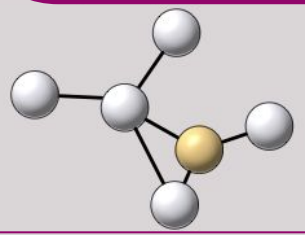
Ego-nets sa



ations from
el encoder
s).

Local structure can hint the full semantics.

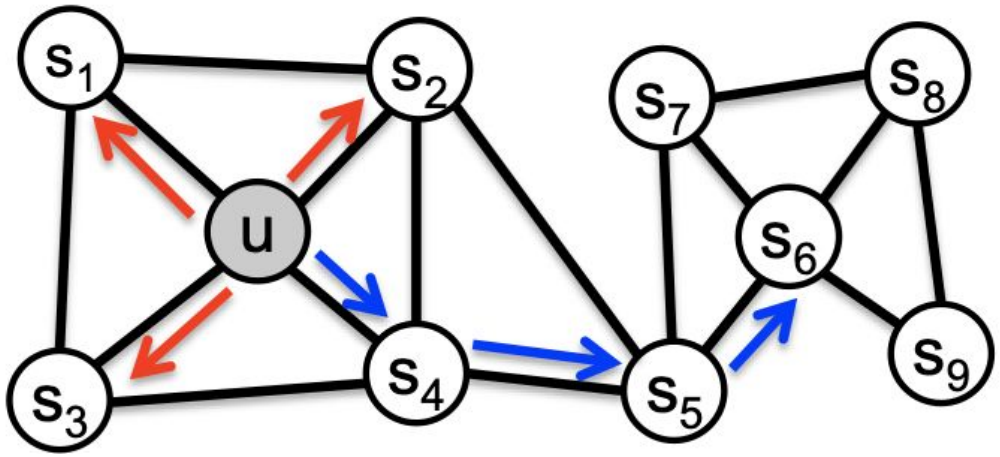
Random walk

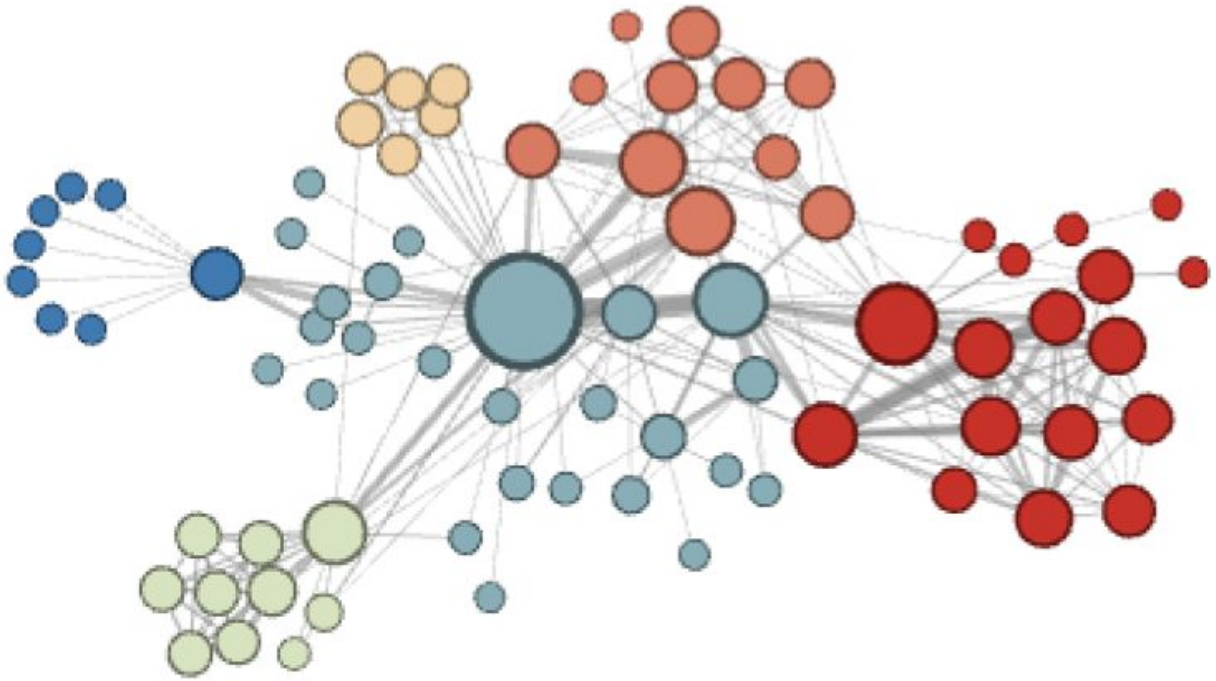


Sample sub-graphs based on random walks starting from a given node.

Credit: [2, 5, 6, 7, 8]

Contrast on the Node Level

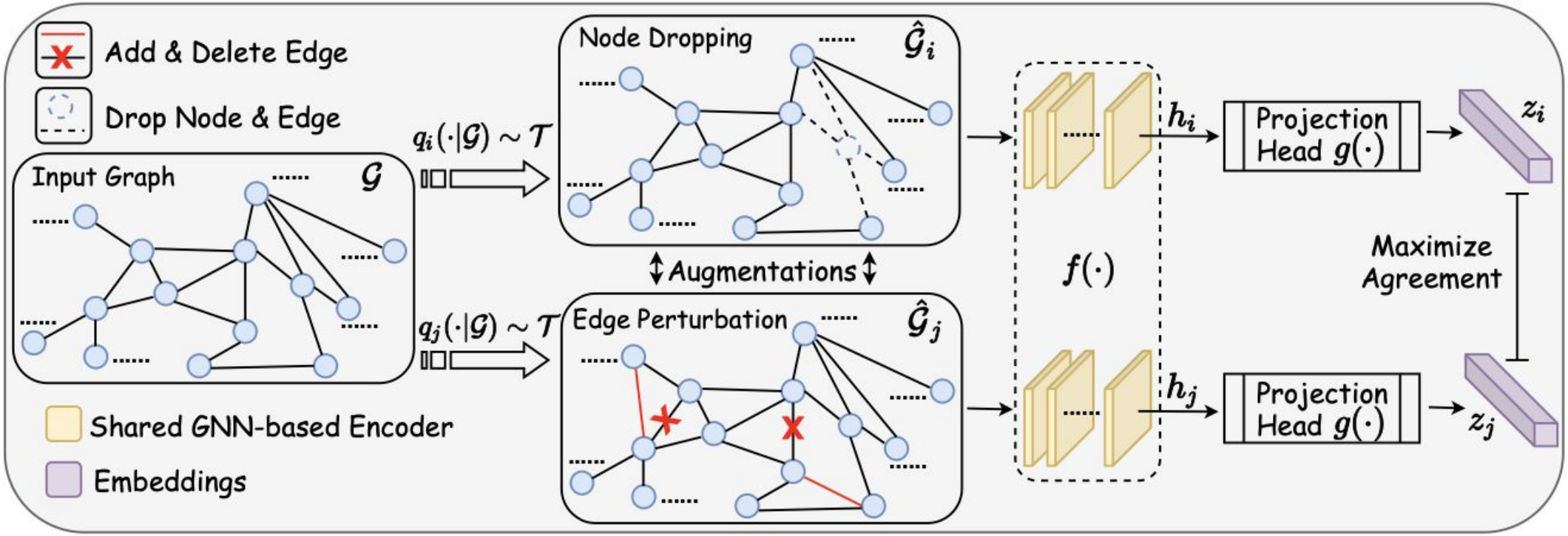




node2vec

Credit: [10]

Contrast on the Graph Level



Positive pairs:

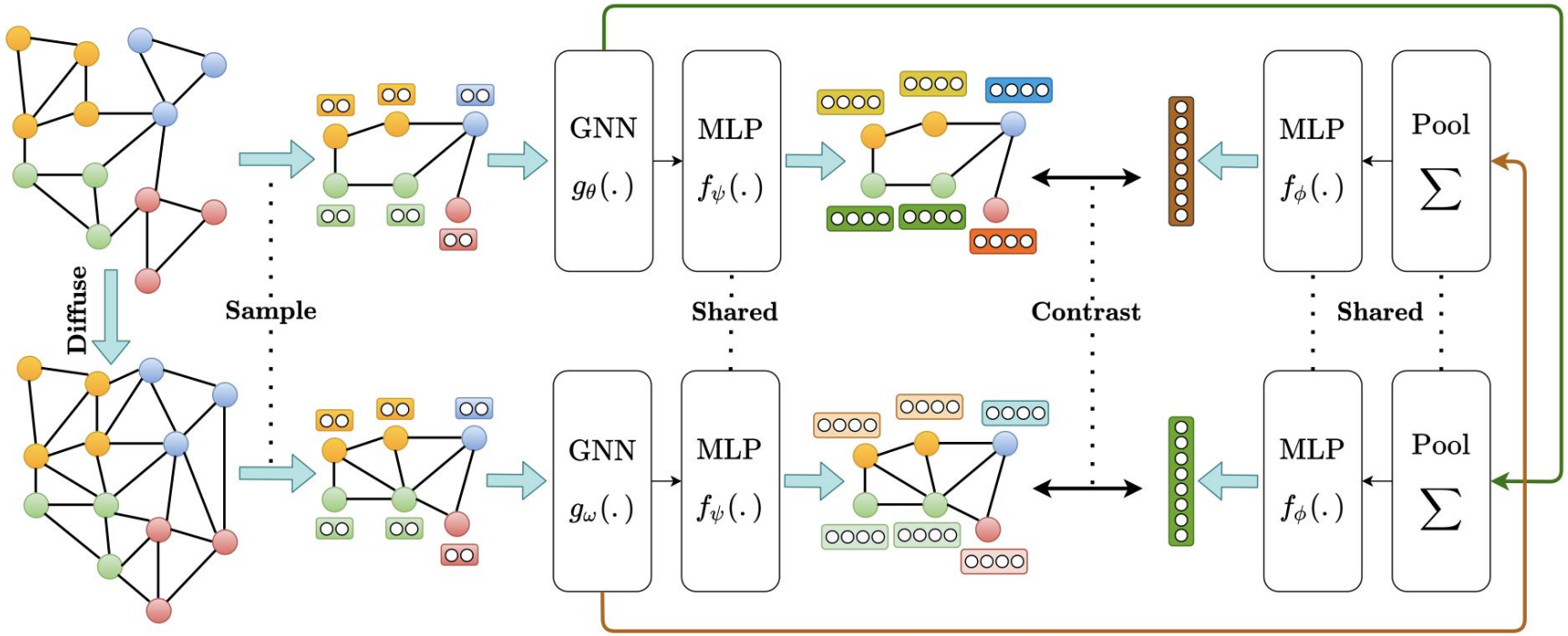
1. Sample different augmentations.
2. Use the original graph and its augmented version.

Negative pairs:

The rest from the batch.

Credit: [5]

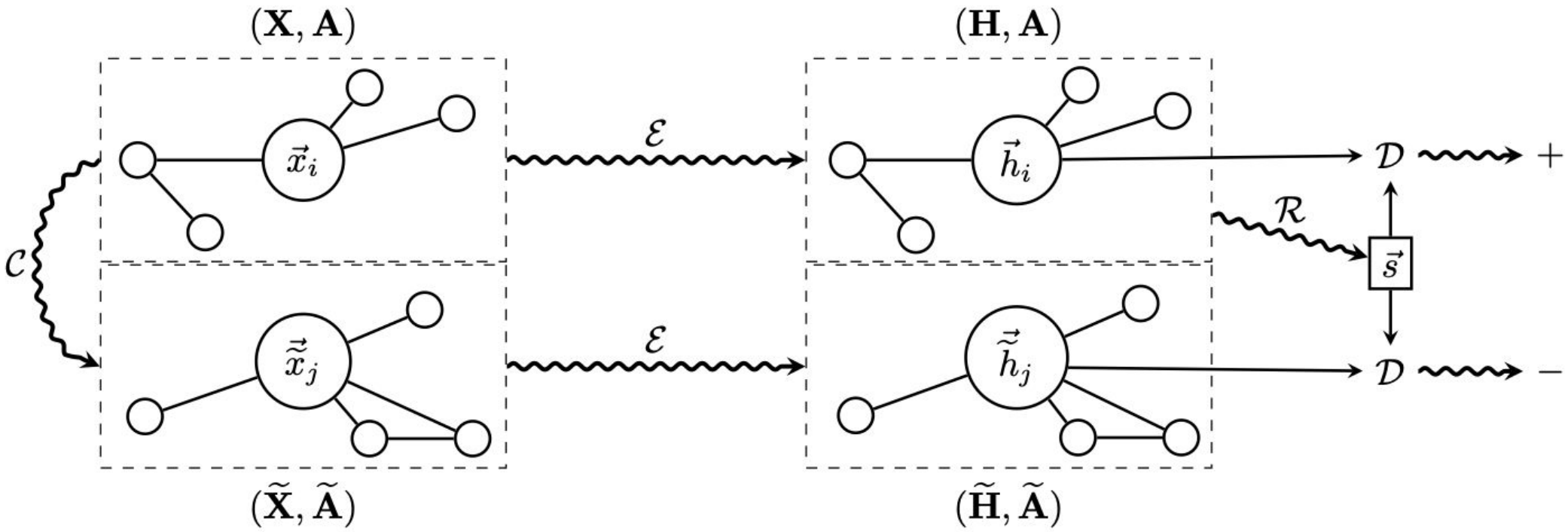
Contrast on the Node-Graph Level



- **Positive pairs**: contrasts node representations from one view with graph representation of another view and vice versa.
- **Negative pairs**: graph and its corrupted version.

Credit: [6]

Contrast on the Node-Graph Level



- **Positive pairs**: contrasts node representation with graph representation.
- **Negative pairs**: node representation in corrupted version of the graph with graph representation.

Credit: [7]

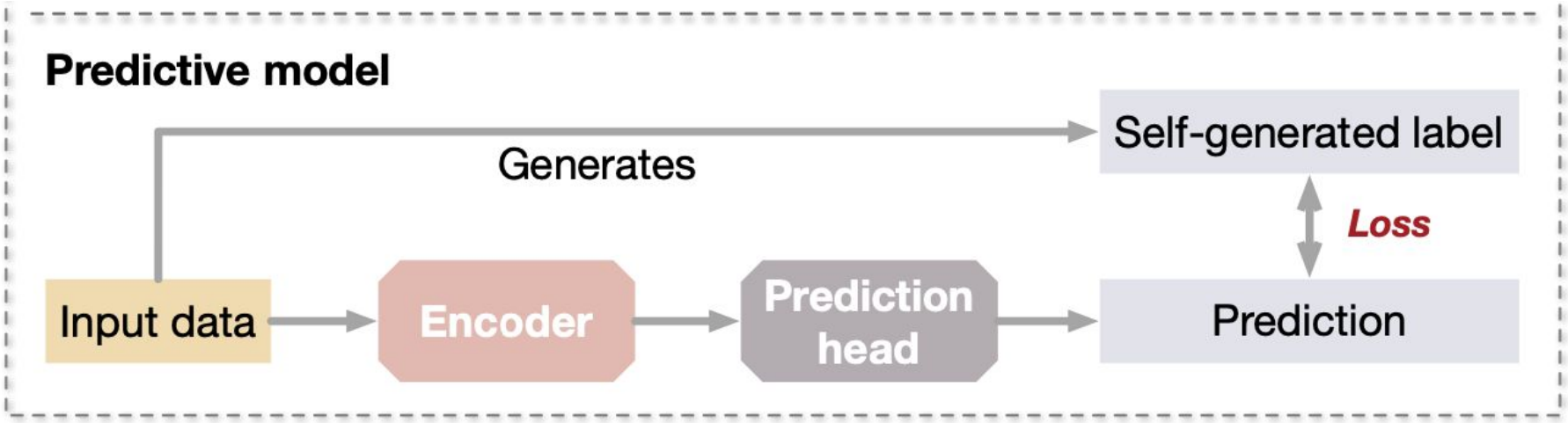
Some Observations

1. Composing different augmentations benefits more than using a pair (original graph, augmented graph).
2. Edge perturbation benefits performance on social networks but hurts molecules.
3. Applying attribute masking achieves better performance in denser graphs.
4. Node dropping and subgraph are generally beneficial across datasets.
5. In many tasks, performance is comparable to or outperforms that of supervised training.

Credit: [5, 6]

Predictive Learning

Predictive Learning

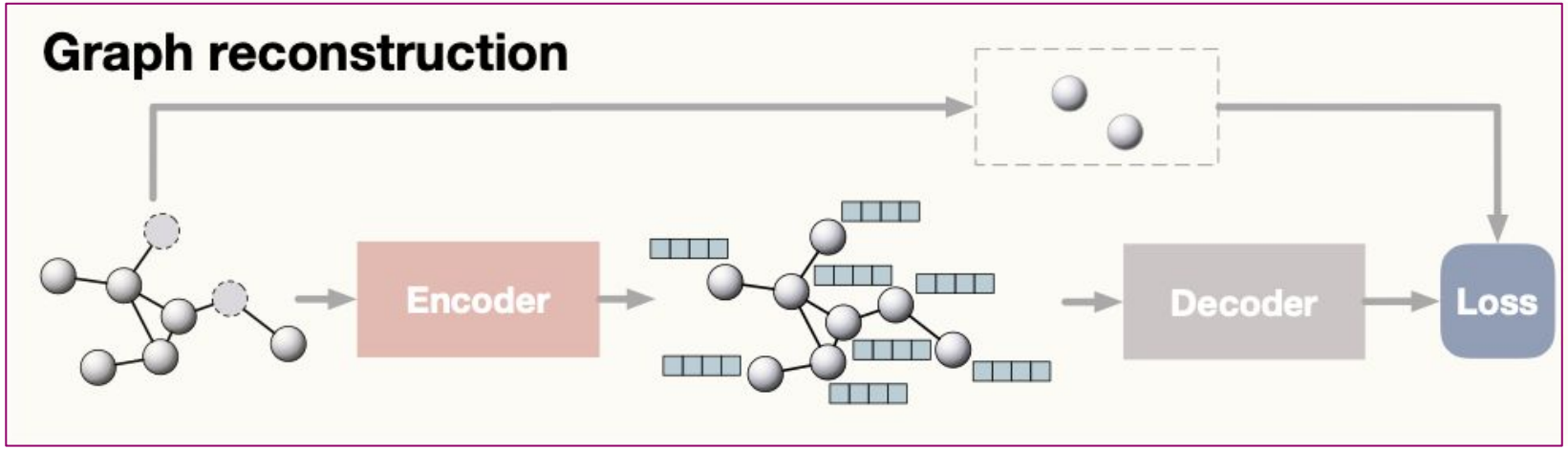


Require data-label pairs, where the labels are self-generated from the data:

- Based on the certain properties of the data.
- By selecting certain parts of the data.

Credit: [2]

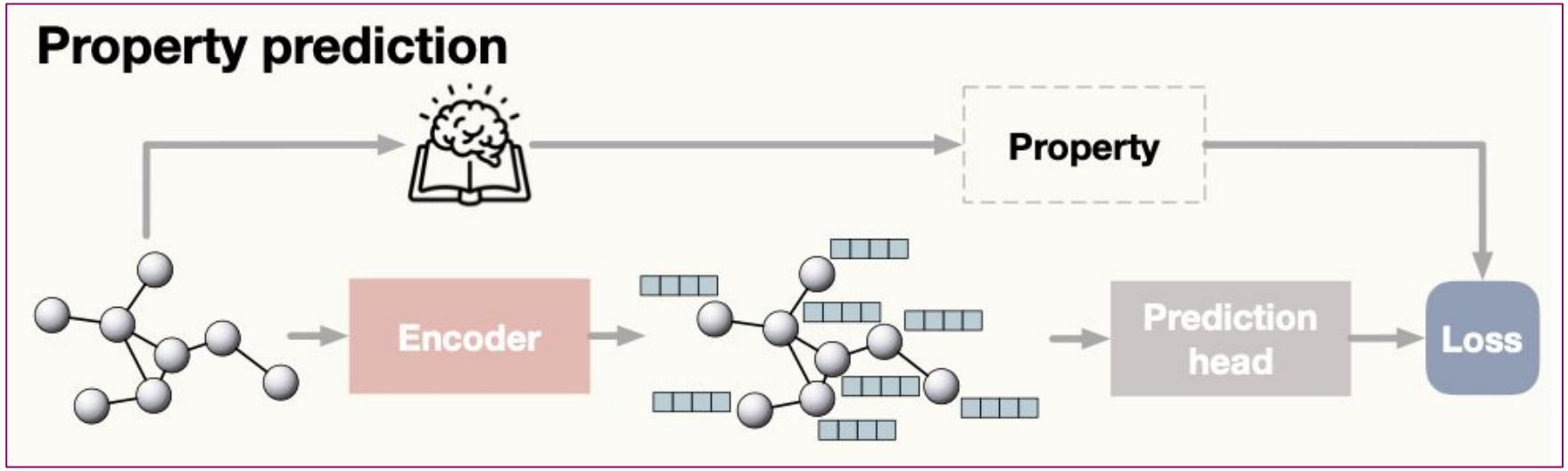
Graph Autoencoders



1. Autoencoder: reconstruction on the adjacency matrix A from the input graph (A, X).
2. Denoising Autoencoder: performs reconstructions of feature matrix X, based on corrupted feature matrix $\tilde{X}$.
3. Attribute Masking is another strategy to pre-train.
4. Variational Autoencoder will be covered on the next lectures.

Credit: [2]

Property prediction

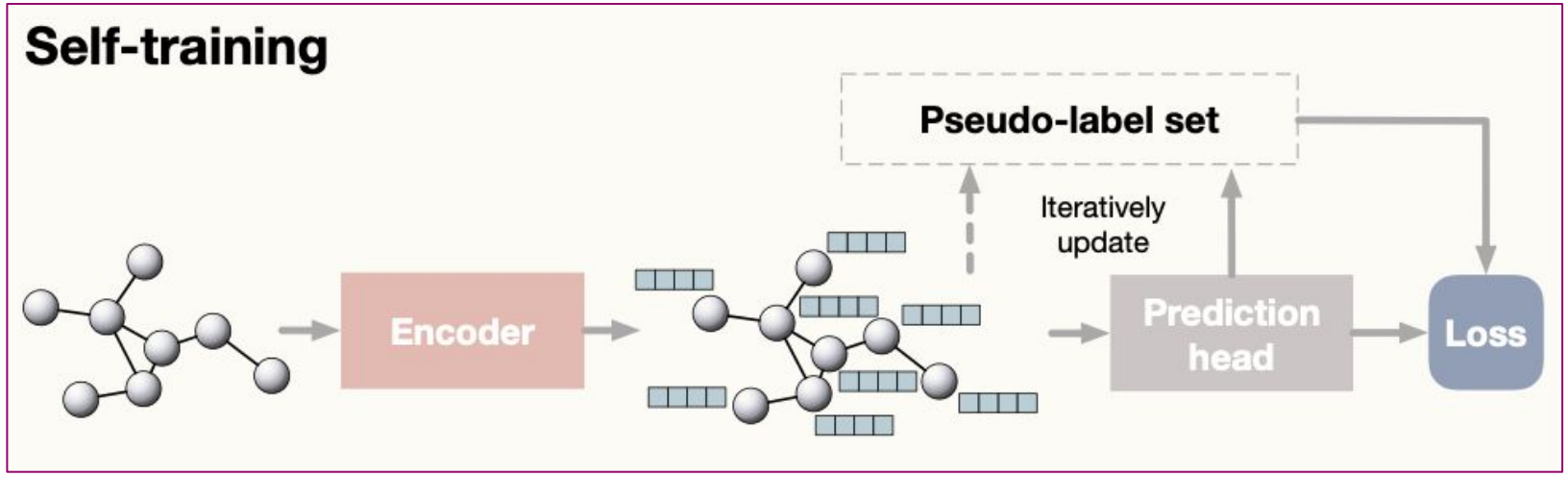


Designing the prediction tasks, based on informative graph properties that are not explicitly provided in the graph data, needs reliable and cheap labels.

Example: graph-level motif (functional group) prediction [9] based on the predefined domain knowledge.

Credit: [2]

Self-training



1. Multi-stage self-training: labeled nodes guide the training on unlabeled nodes. After the training, the predicted labels with high confidence are considered as the pseudo-labels and moved to the labeled node set.
2. Clustering: K-mean is performed on node-level representations for labels; then aligning mechanism.

Credit: [2]

Predictive Learning: Observations

1. Autoencoder-based pre-training improve the performance and show better results in comparison with other methods like node2vec [9].
2. The self-supervised pre-training based on properties prediction leads to a performance gain [10].
3. Empirically, clustering with alignment provides stronger self-supervision than multi-stage self-training [2, 12].

Credit: [2, 9, 11, 12]

Challenges

1. The optimal views generation w.r.t specific downstream tasks are still unclear for contrastive methods.
2. Richer domain knowledge can be better utilized as self-supervision.
3. Scaling-up and efficiency issues are to be addressed.
4. Explainability of SSL for GNN requires further studies.

Credit: [2]

Take-Home Messages

1. Graph Manipulation:
   a. Feature Augmentation - the input graph lacks features;
   b. Structure Augmentation - the input graph too sparse/dense/large.
2. Self-supervised learning: contrastive.
   a. Data Transformation: Feature, Structure, Sampling-based;
   b. How to contrast and Observations.
3. Self-supervised learning: predictive.
   a. Autoencoder, property prediction, self-training.
4. While existing SSL approaches have shown promising effectiveness on learning from graph data, there still exist several challenges.

Credit: [2]

BIOMEDICAL INFORMATICS

Slides & Image Credits

1. CS224W: Machine Learning with Graphs
2. Self-Supervised Learning of Graph Neural Networks: A Unified Review
3. A Survey on Contrastive Self-supervised Learning
4. Contrastive Learning Inverts the Data Generating Process
5. Graph Contrastive Learning with Augmentations
6. Contrastive Multi-View Representation Learning on Graphs
7. Depp Graph Infomax
8. GCC: Graph contrastive coding for graph neural network pre-training
9. Self-Supervised Graph Transformer on Large-Scale Molecular Data
10. node2vec: Scalable Feature Learning for Networks
11. Variational Graph Auto-Encoders
12. Multi-Stage Self-Supervised Learning for Graph Convolutional Networks on Graphs with Few Labeled Nodes
13. On Node Features for Graph Neural Networks
14. Momentum Contrast for Unsupervised Visual Representation Learning