



Université Internationale de Rabat

MACHINE LEARNING FOR GRAPH-STRUCTURED DATA

Graph Neural Networks

Mounir Ghogho

Outline

Introduction

- Attributed graphs
- ML tasks on graphs
- Very brief introduction to ML

Graph embedding

- Direct encoding
- Graph neural networks (GNN)

Node & graph Classification

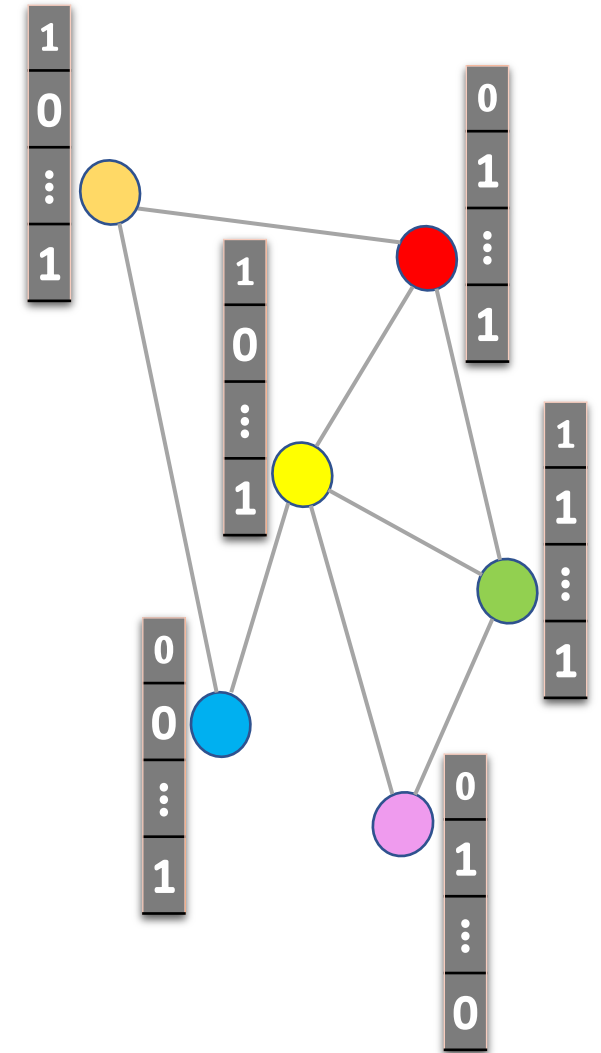
- Homophily
- Collective classification
- GNN-based classifiers
- Graph-assisted Bayesian classifiers

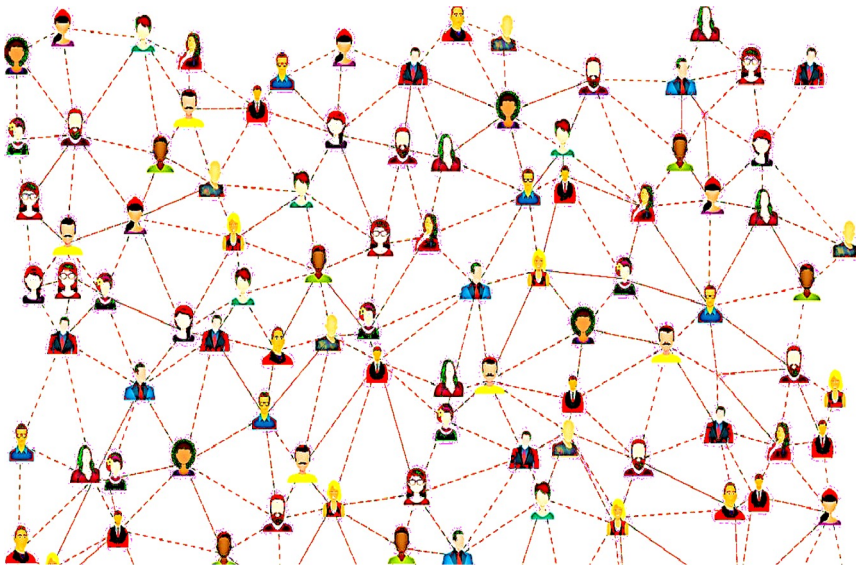
Very brief introduction to Knowledge graphs

- KG embedding
- KG completion

- Attributed graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$
 - \mathcal{V} : node set, \mathcal{E} : edge set
 - $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$: node attributes/features
- Adjacency matrix: $\mathbf{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$, i.e.:

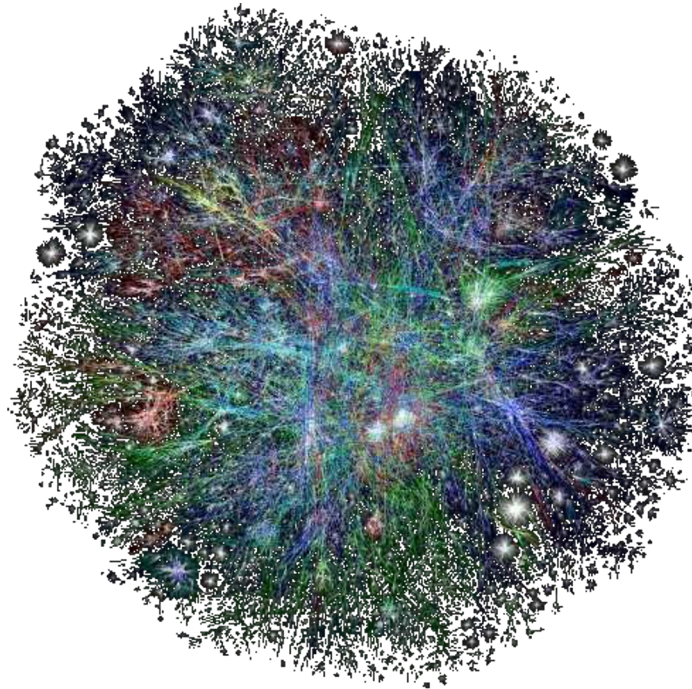
$$A_{ij} = 1 \text{ if } (i,j) \in \mathcal{E} \quad \& \quad A_{ij} = 0 \text{ if } (i,j) \notin \mathcal{E}$$
- Practical graphs are usually sparse, i.e. $|\mathcal{E}| \ll |\mathcal{V}| \times |\mathcal{V}|$
- In some graphs, the edges too have attributes.





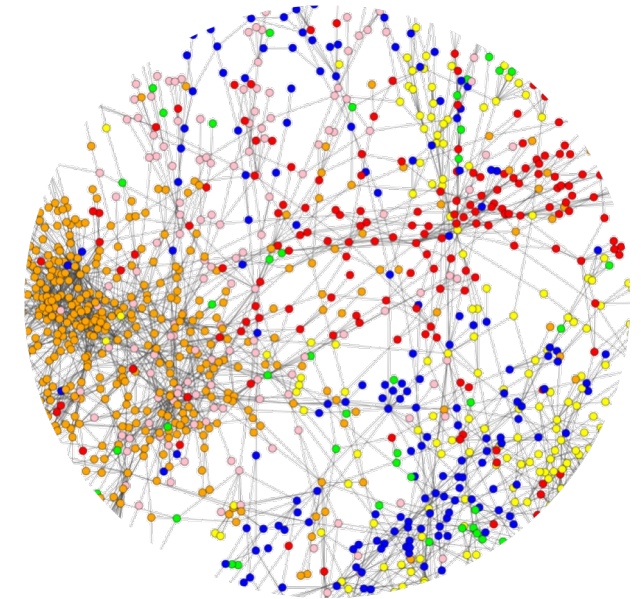
Social network

- Nodes: **Users**
- Edges: **friendship/following**
- Node features: **gender, language, country, activities...**
- Edge features: **bond type**



Internet graph

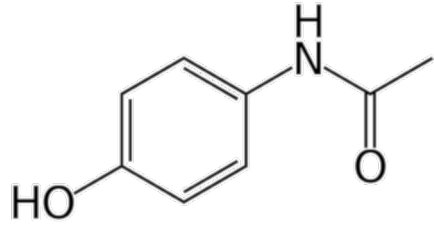
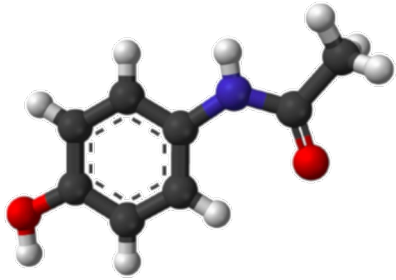
Nodes: **web pages**
Edges: **hyperlinks**
Node features: **page representation**



Citation graph

Nodes: **papers**
Edges: **citation**
Node features: **doc representation**

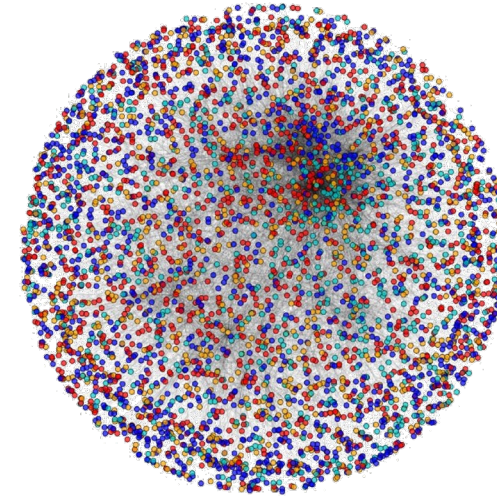
Attributed Graph



Molecule graph

- Nodes: **atoms**
- Node features: **atom type, charge**
- Edges: **bonds**
- Edge features: **bond type**

Other Examples

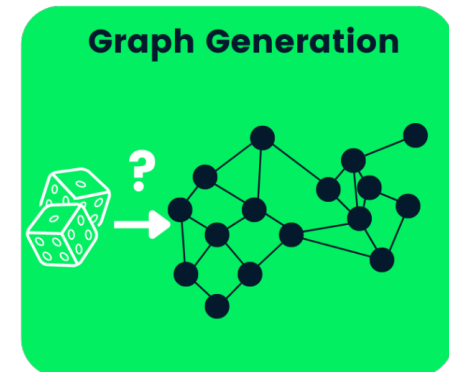
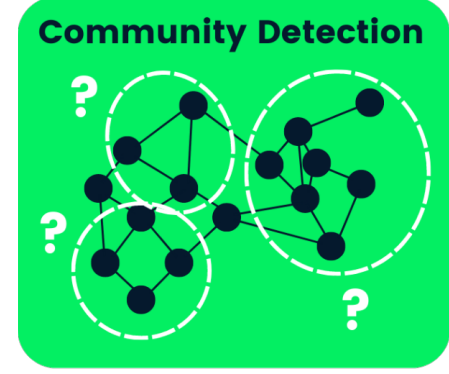
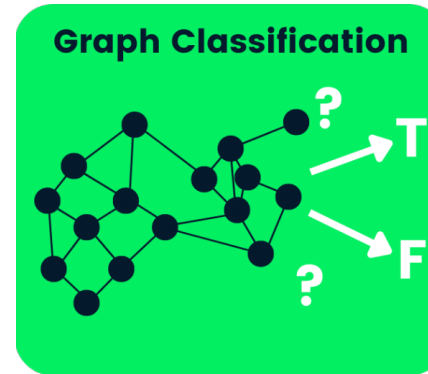
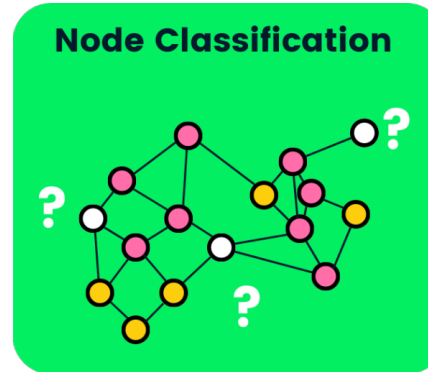


Protein-protein interactions (PPI) graph

- Nodes: **proteins**
- Edges: **ties/physical contact**
- Node features: **protein representation** (e.g. one-hot encoding of 20 standard amino acids and 7 physicochemical properties)

PPI refers to physical contacts established between two or more proteins as a result of biochemical events and/or electrostatic forces

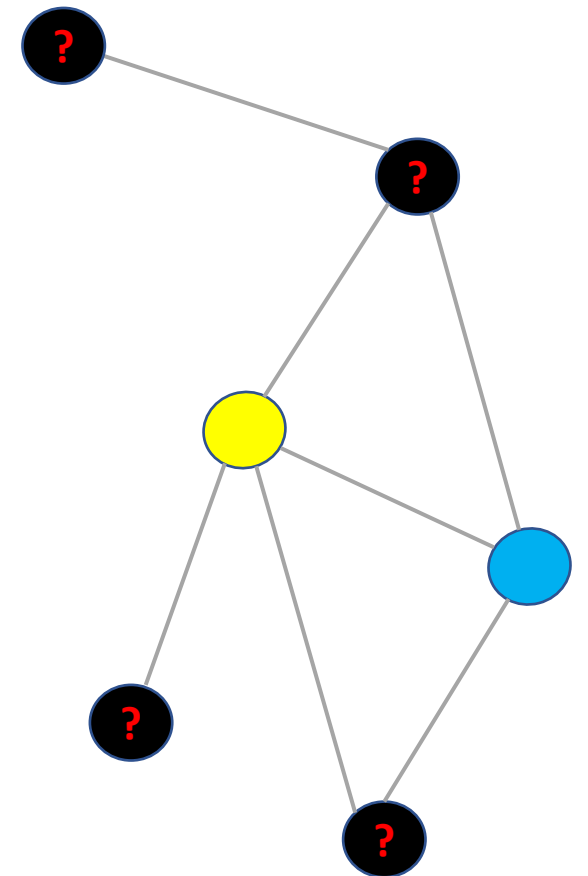
- Node-level tasks
 - Node classification
- Edge-level tasks
 - Link prediction
 - Relation prediction
 - Edge score prediction
- Graph-level tasks
 - Graph classification
 - Graph generation



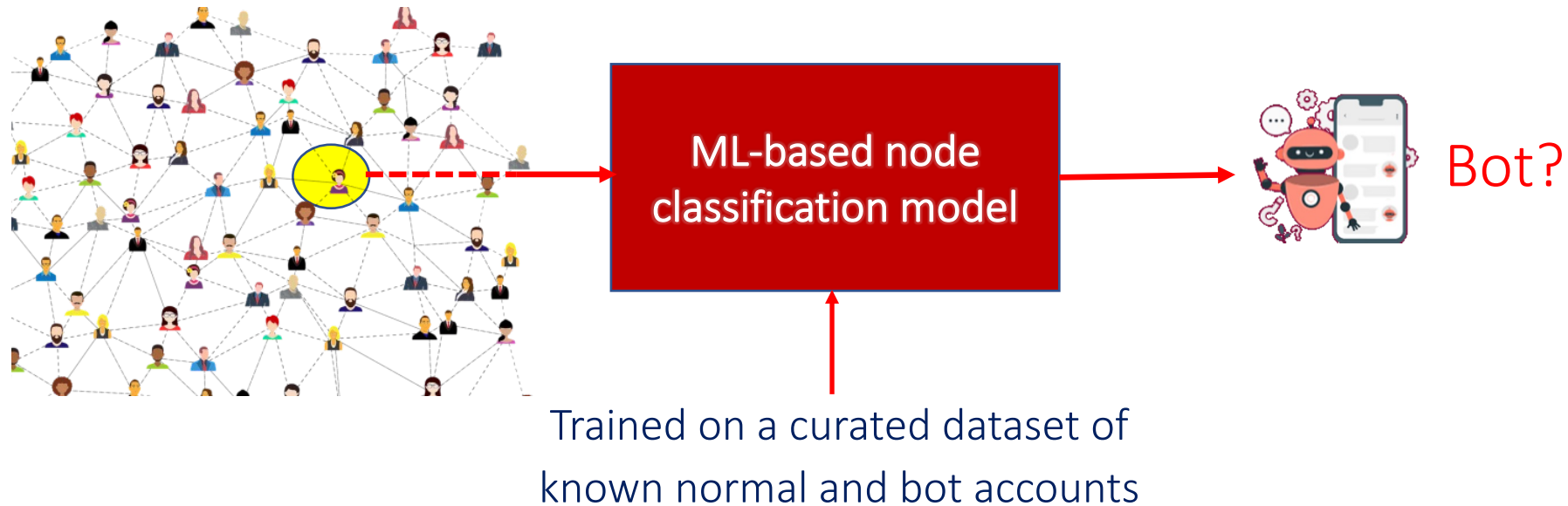
- Each node belongs to one of \mathcal{C} classes:

$$y_i \in \{1, \dots, M\}$$

- Some nodes are labeled (\mathcal{V}^L) and others not (\mathcal{V}^U),
i.e. $\{y_i, i \in \mathcal{V}^L\}$ are known, $\{y_i, i \in \mathcal{V}^U\}$ are unknown
- Often, we have $|\mathcal{V}^L| \ll |\mathcal{V}^U|$
- Goal:** predict the labels of \mathcal{V}^U using **both** node features and graph structure



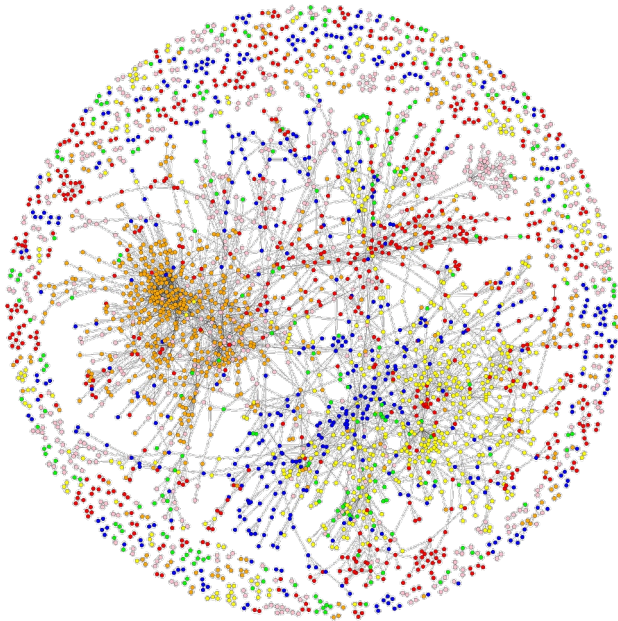
- Prediction of user's properties in Social Networks



Other user's properties: opinion on a topic, political/religious belief, interests, preferences, gendre, age, socioeconomic status, etc.

- Prediction of the category/class of a paper in a citation graph

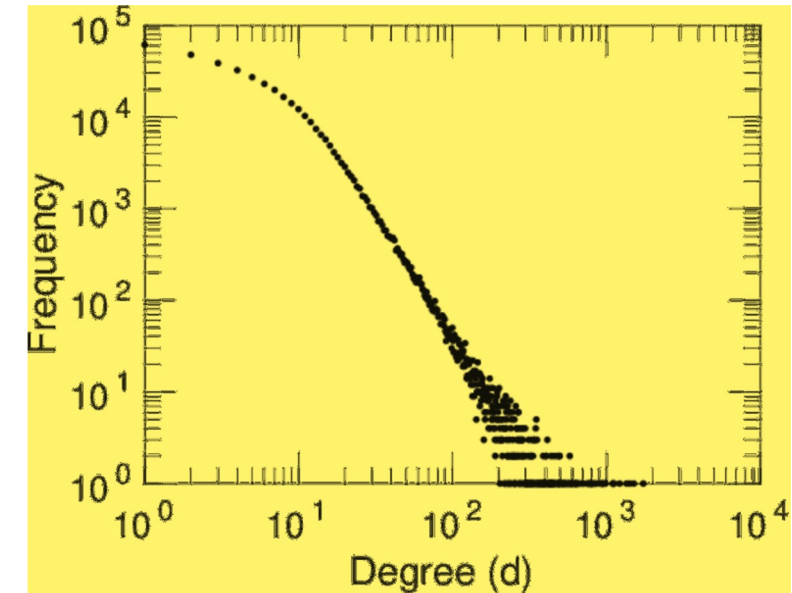
CiteSeer network



- 3312 nodes & 4732 edges
- Node feature vector:

$$x_i = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \begin{array}{l} \leftarrow \text{word 1 present} \\ \leftarrow \text{word 2 absent} \\ \leftarrow \text{word } F \text{ present} \end{array}$$

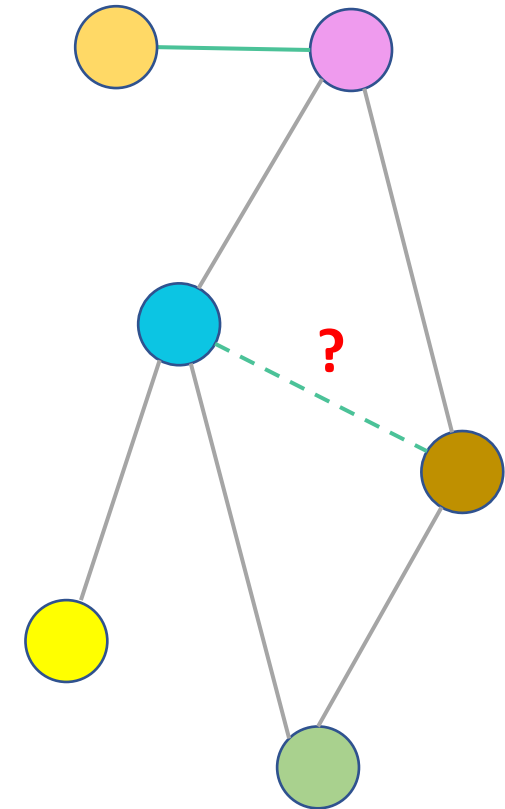
Dictionary: $F = 3703$ words



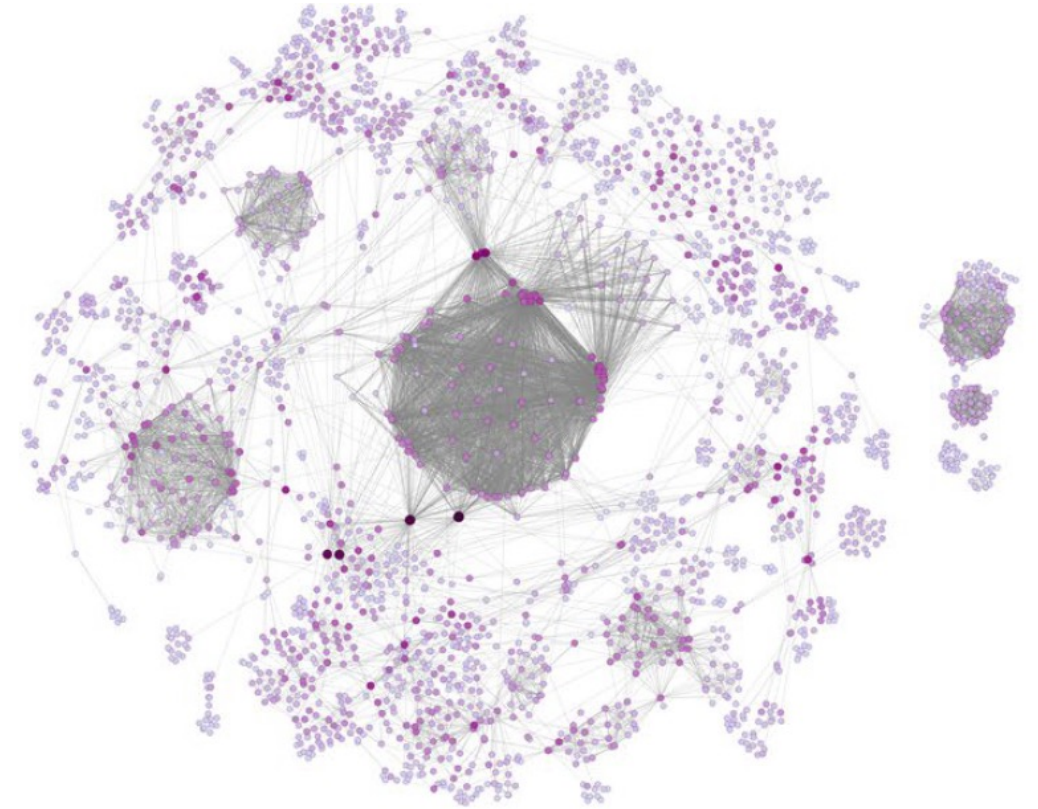
- 6 classes (Agents, AI, Database, HCI, ML, Information Retrieval)
- Graph-agnostic classifier (MLP)(10% for training): accuracy ~ 66%
- Graph-assisted classifier (SOTA) (10% for training): accuracy ~ 73%

The graph provides 7% gain

- Often, we have $|\mathcal{E}| \ll |\mathcal{V}|^2$
- Graphs are incomplete: relevant edges are missing
- **Goal**: predict new links in a graph using graph structure and node features
- The model computing the likelihood of a link is trained using part of \mathcal{E} and validated using another part.

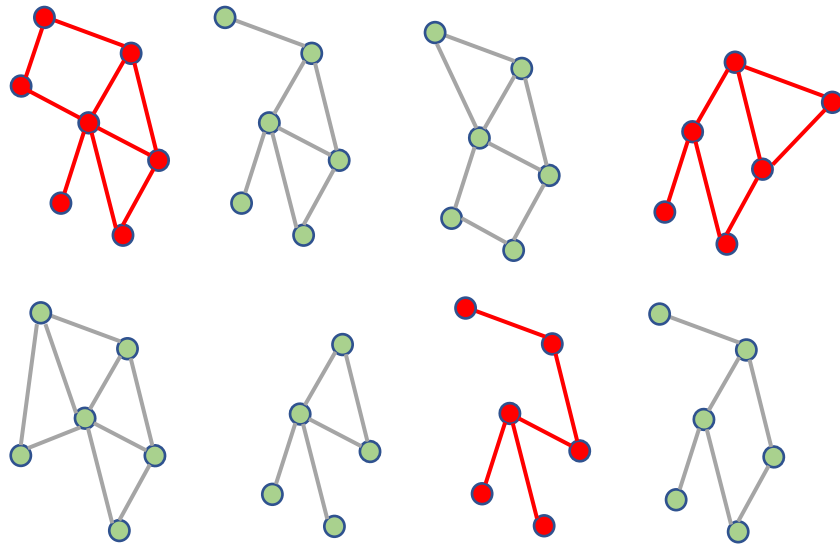


- Investigation of possible protein-protein interactions (PPI) between two proteins in intracellular signaling is an expensive and laborious procedure in the wet-lab
- An in silico approach: Graph ML used to narrow down the candidates for future experimental validations

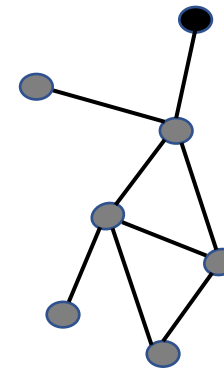


Human PPI network with green edges denoting the top predicted interactions to complete the truncated N90 network to an N100 network [Balogh et al., 2022]

- **Goal:** predict the label of the whole graph
- Often, this concerns small graphs...
- The model is trained on graphs with known property

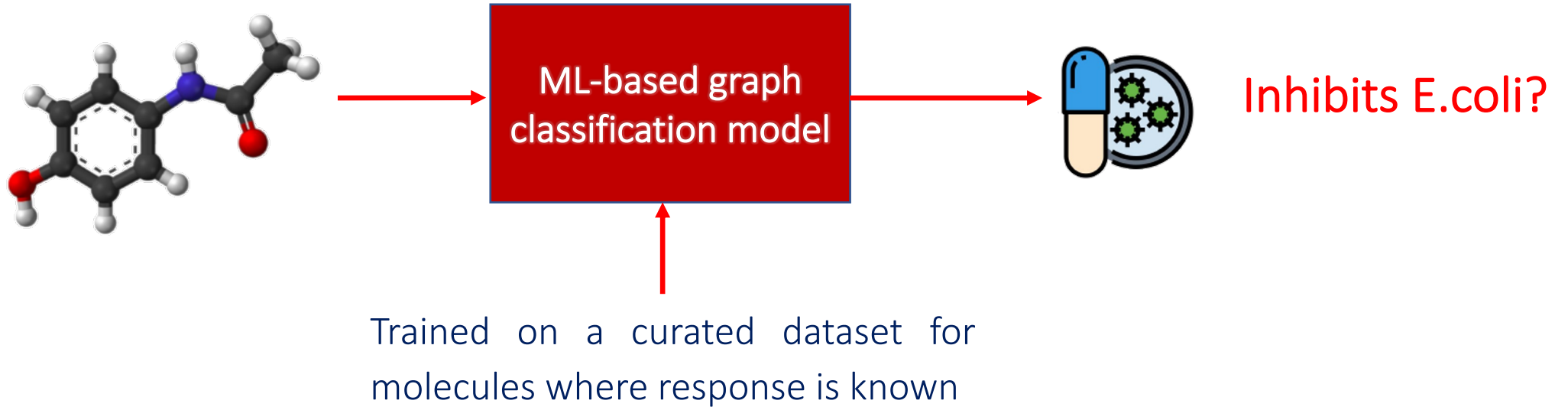


Labeled graphs



Class: Red or Green?

- Molecule classification
 - e.g. predict whether the molecule is a potent drug for some disease



Graph classification

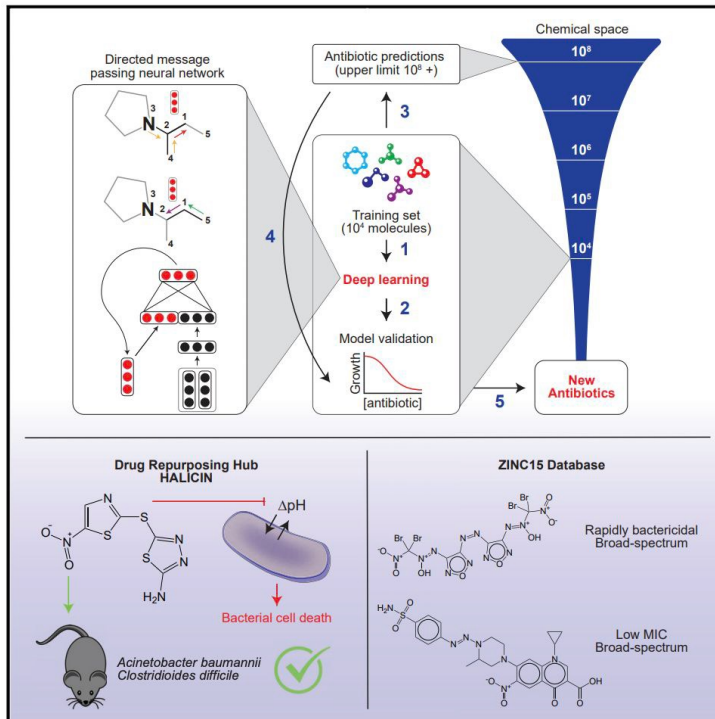
A popularized success story

- Antibiotic discovery (drug repurposing)

Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract



[Stokes et al., Cell'20]

Authors

Jonathan M. Stokes, Kevin Yang,
Kyle Swanson, ..., Tommi S. Jaakkola,
Regina Barzilay, James J. Collins

Correspondence

regina@csail.mit.edu (R.B.),
jimjc@mit.edu (J.J.C.)

In Brief

A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

nature

Subscribe

NEWS · 20 FEBRUARY 2020

Powerful antibiotics discovered using AI

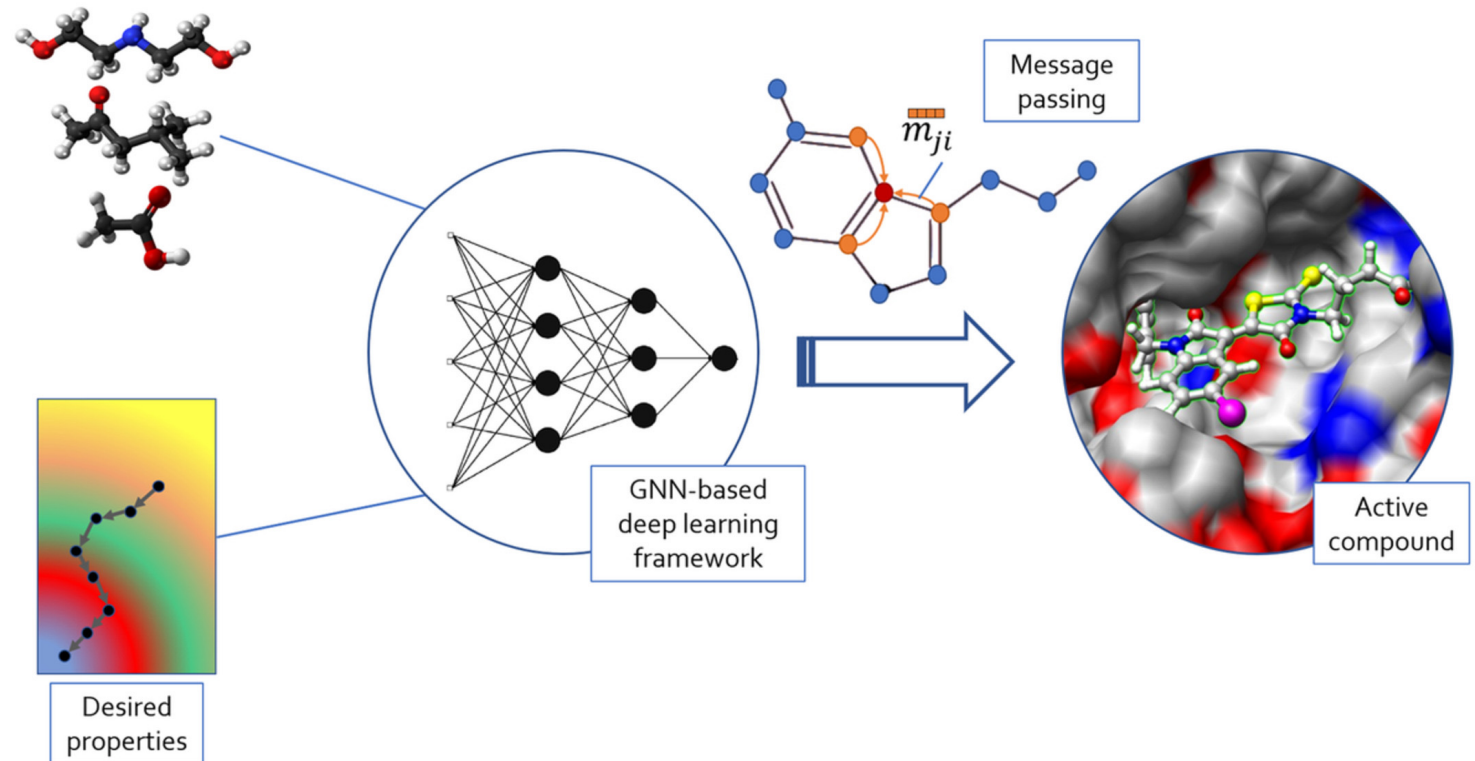
Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

The screenshot shows the Financial Times and BBC News websites. The Financial Times header includes 'FINANCIAL TIMES' and navigation links for COMPANIES, TECH, MARKETS, GRAPHICS, OPINION, WORK & CAREERS, LIFE & ARTS, and HOW TO SPEND IT. The BBC News header includes 'CORONAVIRUS BUSINESS UPDATE', 'Get 30 days' complimentary Update newsletter', and navigation links for News, Sport, Reel, Worklife, Travel, and Future. The main article title is 'Powerful antibiotics discovered using AI' with a sub-headline 'Machine learning uncovers powerful antibiotics'. The article is dated '21 February 2020'.

Halicin-structurally divergent from conventional antibiotics- displays [bactericidal activity](#) against a wide [phylogenetic](#) spectrum of pathogens including [Mycobacterium tuberculosis](#)

- **Goal:** learn the distribution generating real graphs, $p_{\text{data}}(G)$. The learnt distribution, $p_{\text{model}}(G)$ is then used to generate new graphs.
- **Examples:** drug discovery, design of new materials, etc.

- A DNN is used to probabilistically generate new molecules a single bond at a time
- GNNs often coupled with conditioning techniques to steer the generation process towards desired chemical and biological properties

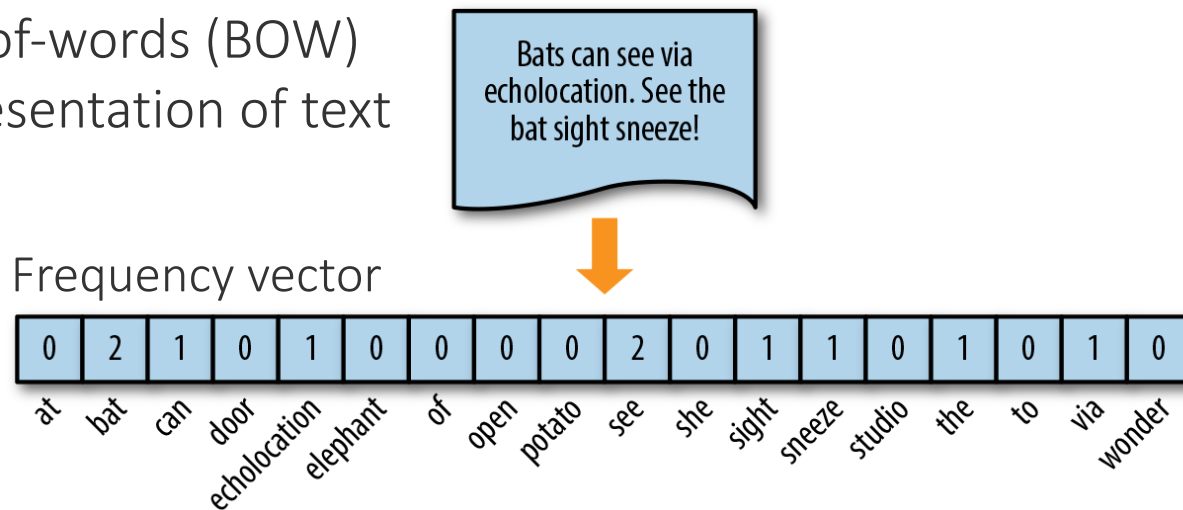


[Abate et al., 2023] Graph neural networks for conditional de novo drug design. WIREs Computational Molecular Science, 2023

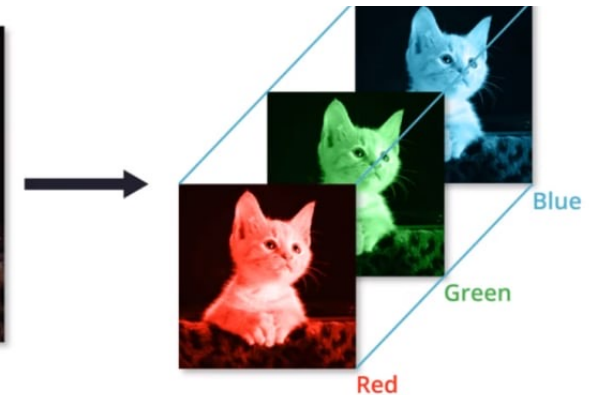
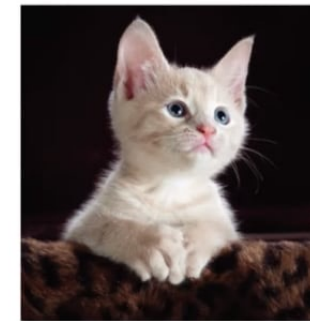
A Very Brief Introduction to Machine Learning

- **Instance**: often a feature vector $\mathbf{x} \in \mathcal{X}$ that represents a specific object
 - \mathbf{x} is an abstraction of the object. Learning task then ignores all other information about the object that is not represented in \mathbf{x} . The more **expressive** the representation, the more accurate the learning tasks.

Bag-of-words (BOW)
representation of text

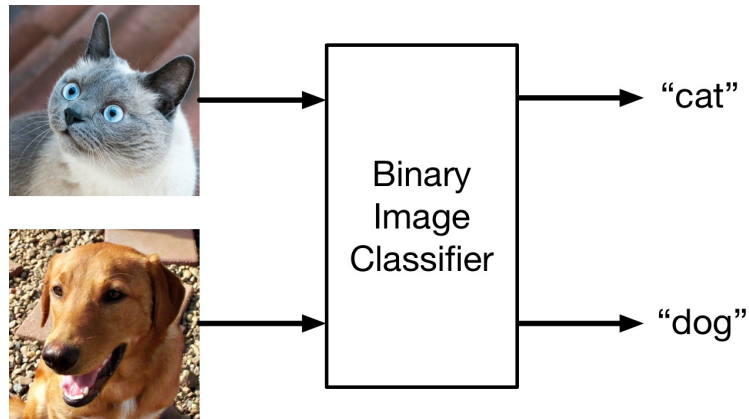


RGB image

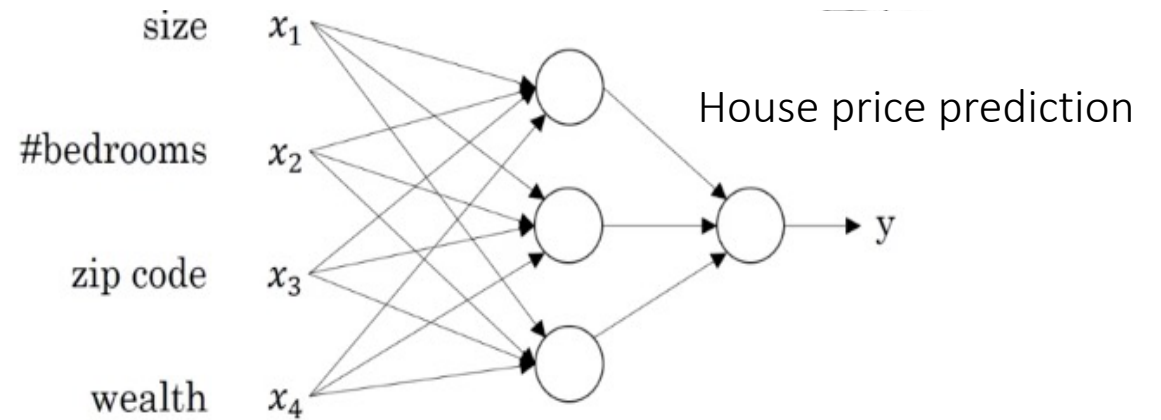


3 matrices

- **Label**: the desired prediction on an instance \mathbf{x} , denoted $y \in \mathcal{Y}$
- **Supervised learning**: Let $p_{\mathbf{x},y}(\mathbf{x}, y)$ denote the (**unknown**) joint probability on instances and labels. Given a training sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with $(\mathbf{x}_i, y_i) \sim p_{\mathbf{x},y}$, supervised learning trains a function $f: \mathcal{X} \rightarrow \mathcal{Y}$ in some function family \mathcal{F} with the goal that $f(\mathbf{x})$ predicts the true label y on future instance \mathbf{x} , where $(\mathbf{x}, y) \sim p_{\mathbf{x},y}$
- Two main supervised learning tasks:
 - Classification**: \mathcal{Y} is a discrete set
 - Regression**: \mathcal{Y} is a continuous set



ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
1000 classes; 1.2M training images, 50k validation images, and 150k testing images



- The 'best' f is:

$$f^* = \arg \min_{f \in \mathcal{F}} \mathcal{L}(f), \quad \mathcal{L}(f) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{\mathbf{x}, y}} \{c(y, f(\mathbf{x}))\}$$

- $c(y, f(\mathbf{x}))$: cost of $f(\mathbf{x}) \neq y$
- Bayes optimal predictor: same as above but no constraint on the domain of f
- **Challenge**: $p(\mathbf{x}, y)$ is **unknown** and we have to generalize the prediction from a finite training sample to any **unseen** test data (**induction problem**):

- For a regression problem, if performance is measured using MSE:

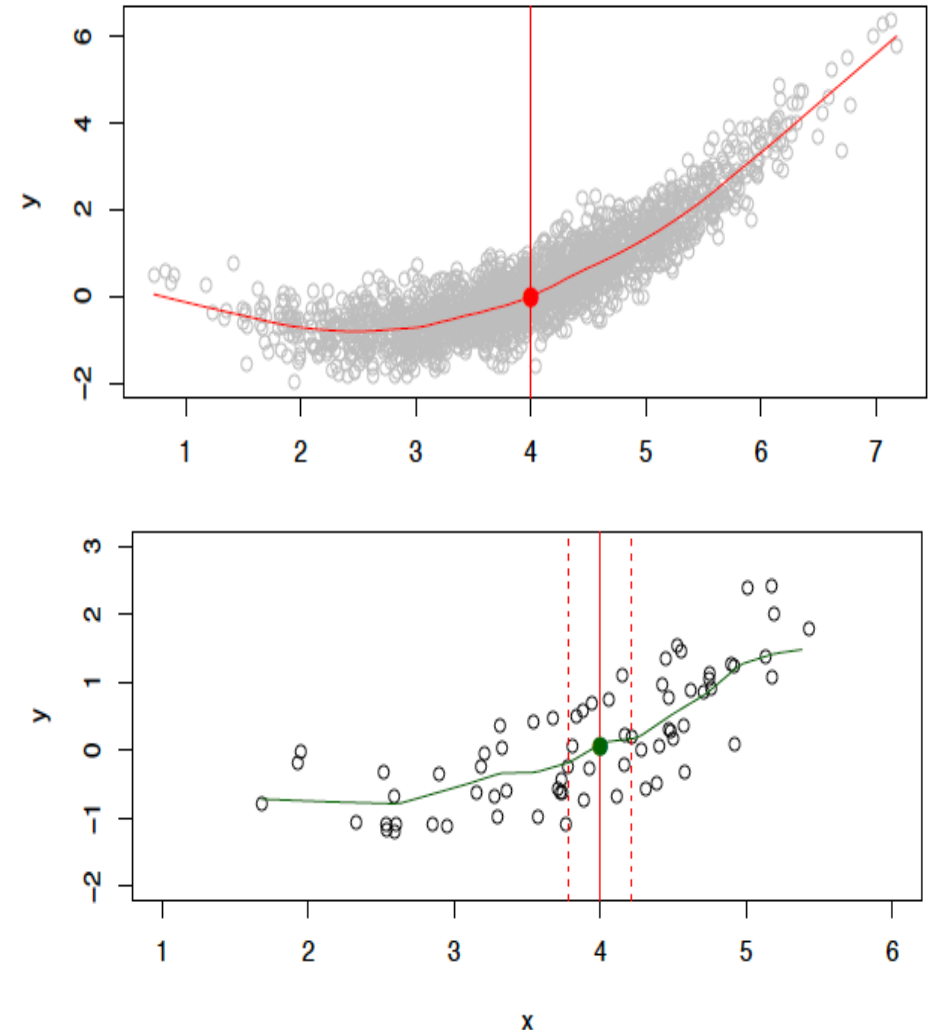
$$\mathcal{L}(f) = E[(f(\mathbf{x}) - y)^2]$$

the best regression function is:

$$f(\mathbf{x}) = E[y \mid \mathbf{x}]$$

- **Example:** If there are many y values at $x = 4$, a good estimate for $f(x)$ is the average of all these y values.
- Typically, we have few if any data points with $x = 4$ exactly. So we approximate $E[y \mid \mathbf{x}]$, e.g.

$$f(\mathbf{x}) \cong \text{Ave}(y \mid X \in \mathcal{N}(\mathbf{x}))$$



- For classification, we often seek to compute the posterior probability

$$p_m(\mathbf{x}) \triangleq \Pr[y = m \mid \mathbf{x}], \quad m = 1, \dots, C$$

- Typically, we choose the class for which the posterior probability is maximum, i.e.

$$f(\mathbf{x}) = \arg \max_m p_m(\mathbf{x})$$

- But if different errors have different costs, a different classification scheme may be chosen.

- **Discriminative methods** (parametric): directly estimate the posterior probability

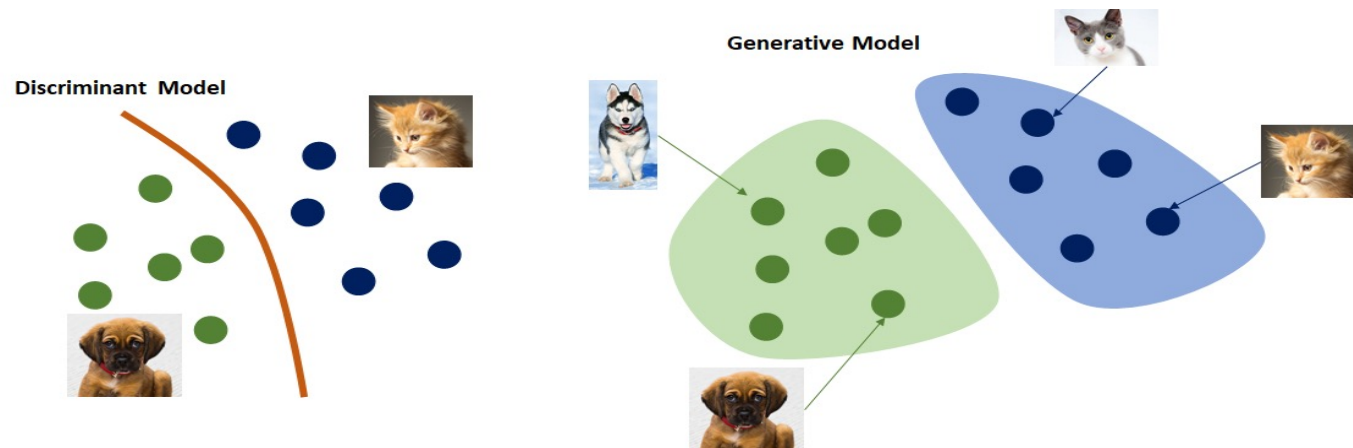
$$p_m(\mathbf{x}) \triangleq \Pr[y = m \mid \mathbf{x}] = [g_{\theta}(\mathbf{x})]_m$$

e.g. logistic regression, SVM, decision trees, artificial neural networks

- **Generative methods**: indirectly estimate the posterior probability

$$p_m(\mathbf{x}) \propto \Pr[\mathbf{x} \mid y = m] \Pr[y = m] = g_{\theta_m}(\mathbf{x}) \pi_m \quad \pi_m = \Pr[y = m]$$

e.g. linear discriminant analysis, Naive Bayes



- For regression:

$$c(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

- For classification:

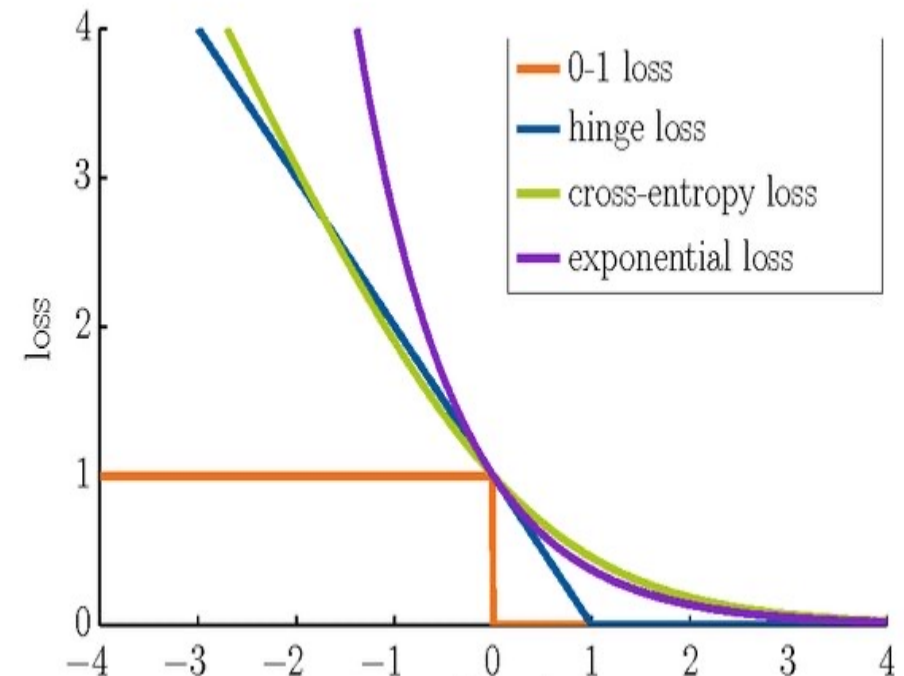
- 0-1 loss

$$c(y, f(\mathbf{x})) = \mathbb{I}(f(\mathbf{x}) \neq y)$$

- Cross-entropy

$$c(y, f(\mathbf{x})) = - \sum_{m=1}^M \beta^{(m)} \log p_m(\mathbf{x})$$

$$f(\mathbf{x}) = \arg \max_m p_m(\mathbf{x}), \beta^{(m)} = \delta(y - m)$$



- Model parameters are learnt from the training set:

$$\mathcal{L}_{\text{Tr}}(f) = \phi(\{c(y_i, f(\mathbf{x}_i)), i = 1, \dots, N\})$$

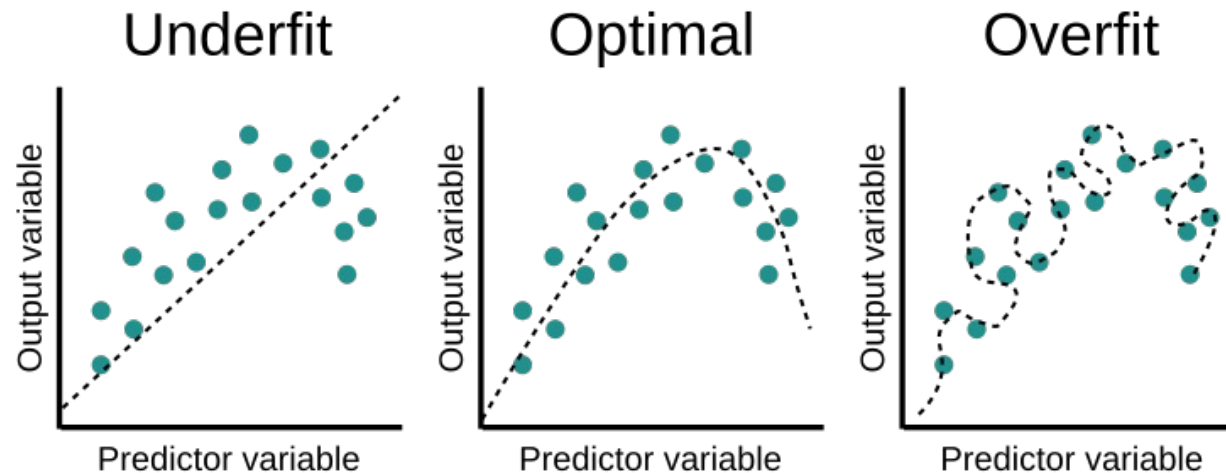
- Assuming i.i.d. training examples:

$$\mathcal{L}_{\text{Tr}}(f) = \frac{1}{N} \sum_{i=1}^N c(y_i, f(\mathbf{x}_i))$$

- If $c(y_i, f(\mathbf{x}_i))$ = cross-entropy, $\mathcal{L}_{\text{Tr}}(f)$ = - log-likelihood that the training examples are generated from model f (under i.i.d. assumption)

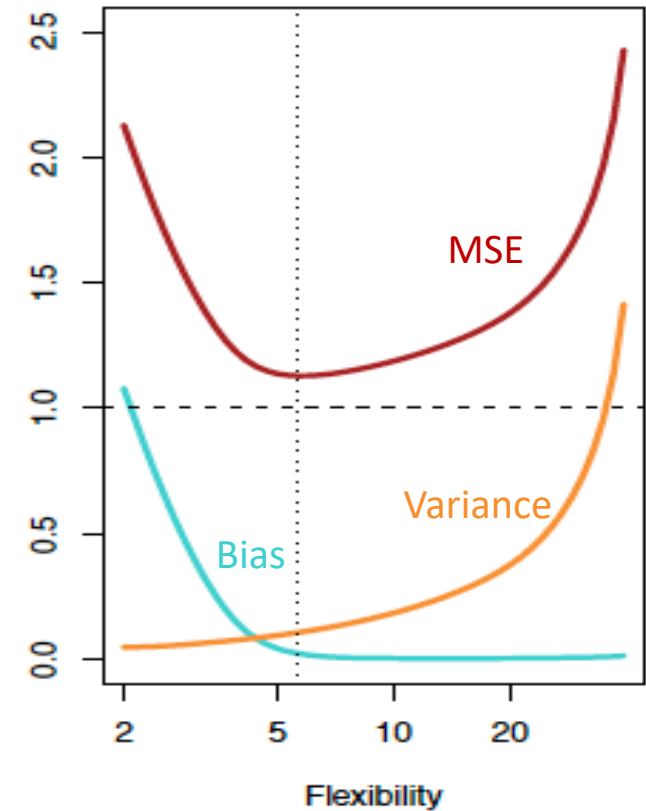
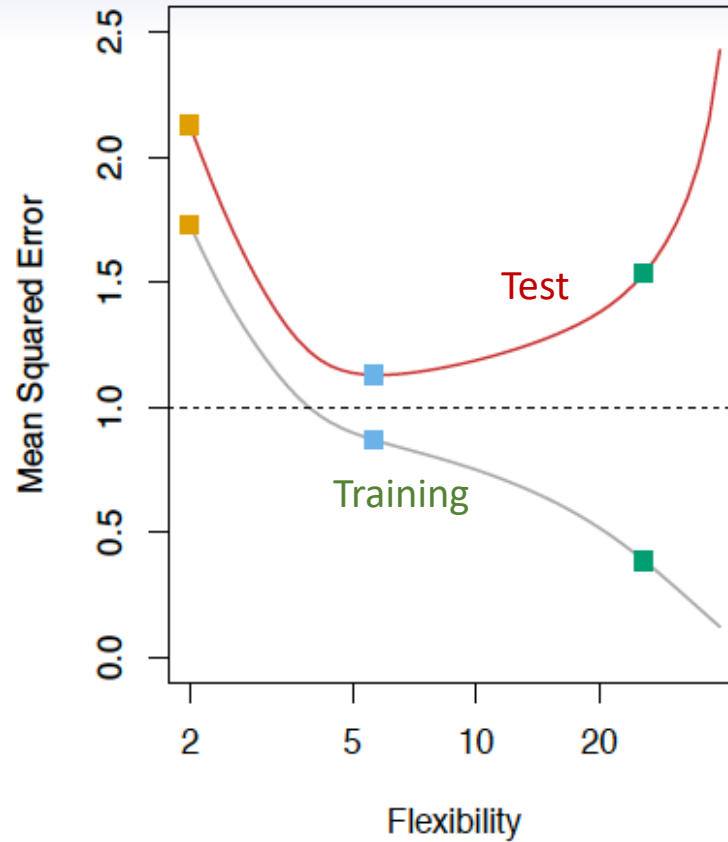
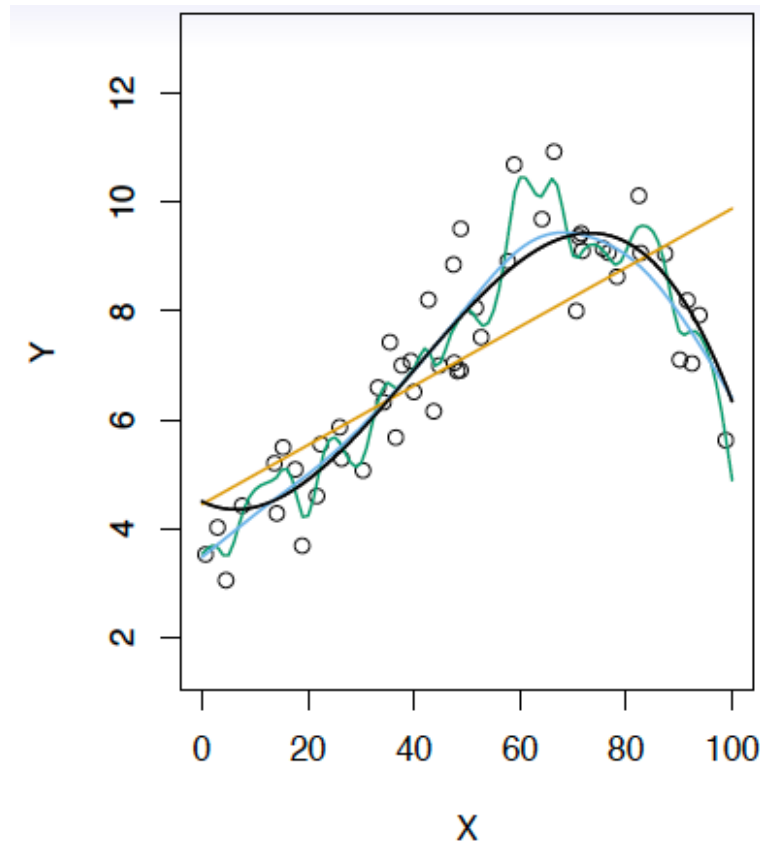
$$L(f; \{(\mathbf{x}_i, y_i)\}_{i=1}^N) = \prod_{i=1}^N \prod_{m=1}^M (p_m(\mathbf{x}_i))^{\beta_i^{(m)}} \xrightarrow{\text{Log}} \sum_{i=1}^N \sum_{m=1}^M \beta_i^{(m)} \log p_m(\mathbf{x}_i)$$

- Seeking f that minimizes training error results in overfitting: more than the true relationship between x and y is 'learnt' due to noise



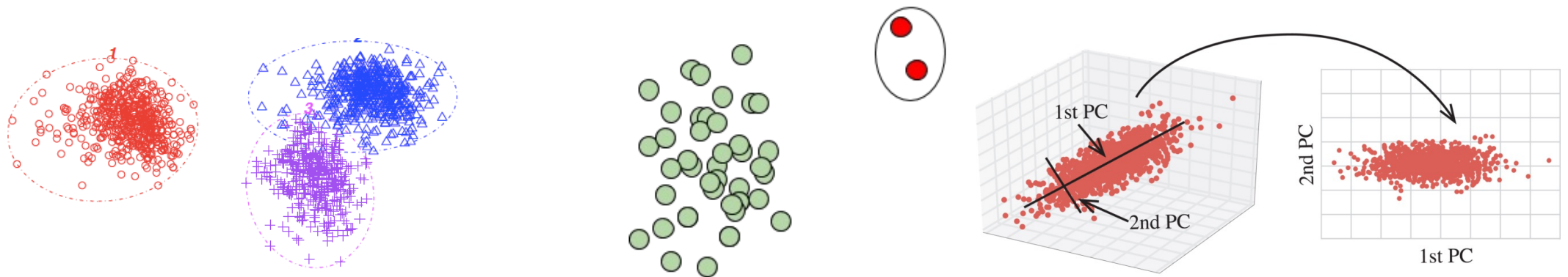
- **Regularization**: minimize the training sample error while regularizing f so that it is not too complex in a certain sense (e.g. Ridge, Lasso, dropout, pooling, etc.)
- **Validation set**: use part of the training as a validation set to optimize hyperparameters (i.e. complexity of the model)

- Training and test errors, and bias-variance treading-off



$$E \left(y_0 - \hat{f}(x_0) \right)^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

- Unsupervised learning algorithms work on a training sample $\{\mathbf{x}_i\}_{i=1}^N$ with no labels
- Common tasks:
 - **Clustering**: separates the n instances into groups
 - **Anomaly detection**: identifies unusual instances
 - **Dimensionality reduction**: reduce the dimension of the instance representation while maintaining key characteristics of the training sample



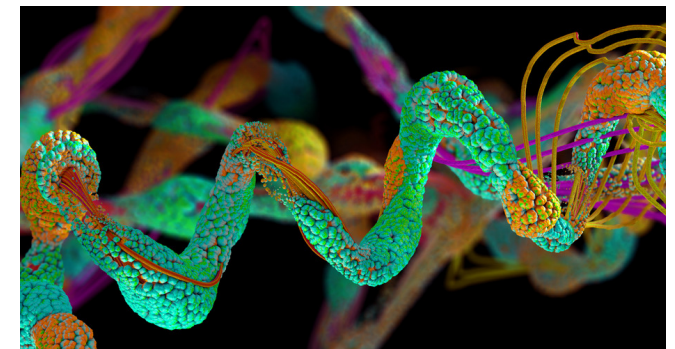
- Semi-supervised learning extends either unsupervised or supervised learning to include additional information typical of the other learning paradigm.
- **Example:** Semisupervised classification. The training data consists of both L labeled instances $\{\mathbf{x}_i, y_i\}_{i=1}^L$ and U unlabeled instances $\{\mathbf{x}_i\}_{i=L}^{L+U}$. Typically, $U \gg L$
- **Motivation:** paucity of labeled data. Labeling requires human annotators, special devices, expensive and long experiments.



Tedious task



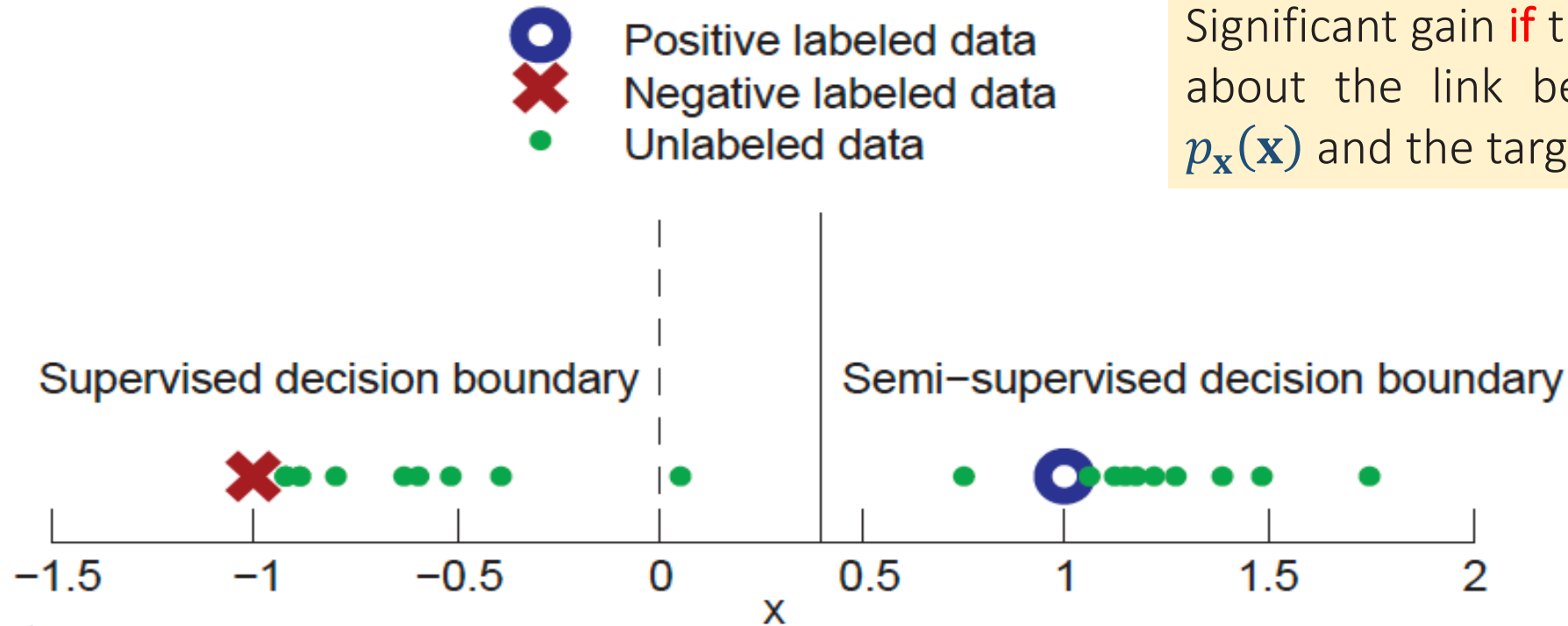
Medical expertise



Lengthy/expensive lab work
by expert crystallographers

- What makes semi-supervised learning useful for estimating f ?
- The answer lies in the **assumptions** one makes about the link between the distribution of unlabeled data $p_{\mathbf{x}}(\mathbf{x})$ and the target label
- Under the assumption that instances in each class form a **coherent group** (e.g. $p_{\mathbf{x}|y}(\mathbf{x}|y)$ is a Gaussian distribution, i.e. instances from each class center around a central mean), unlabeled data provides useful information
- If the assumptions are wrong, unlabeled data are not useful and may lead to worse performance

- Assuming one feature and Gaussian distributions

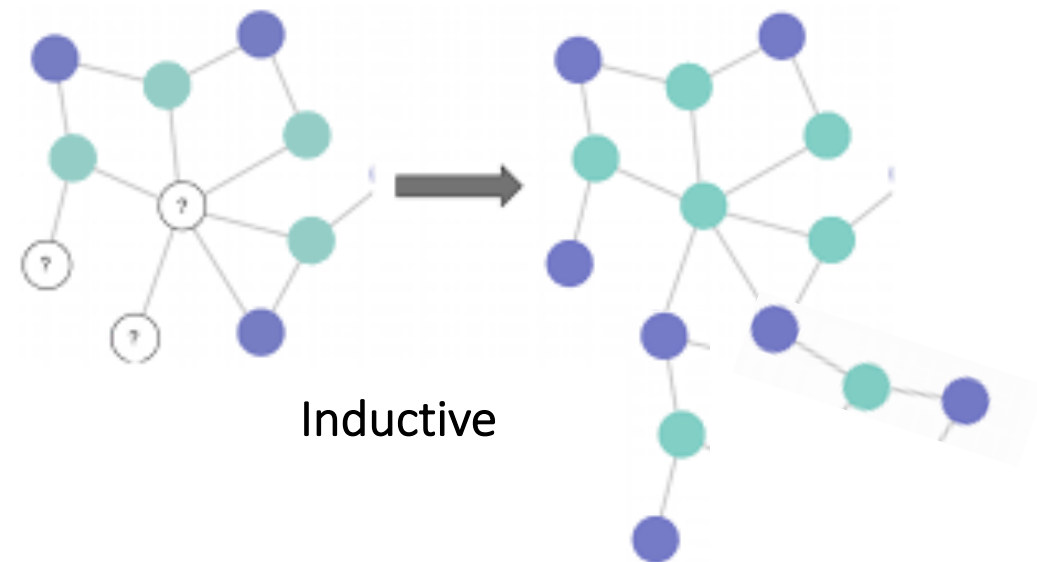
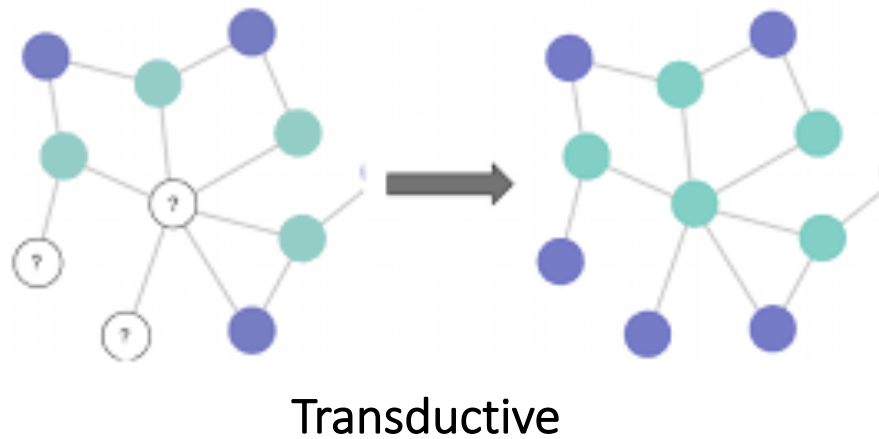


Significant gain **if** the assumptions made about the link between the marginal $p_{\mathbf{x}}(\mathbf{x})$ and the target label is relevant

[Zhu et al., 2009]

- Intuitively, the distribution of unlabeled data helps to identify regions with the same label, and the few labeled data then provide the actual labels

- **Transductive semi-supervised learning:** f is expected to be a good predictor only on the unlabeled data $\{\mathbf{x}_i\}_{i=L}^{L+U}$; predictions outside this set is not required.
- **Inductive semi-supervised learning:** the goal is to train f to predict the label beyond $\{\mathbf{x}_i\}_{i=L}^{L+U}$. f is evaluated on a test set **not used** in the training.



- Semi-supervised learning methods differ in the assumptions made on the link between the marginal distribution $p_{\mathbf{x}}(\mathbf{x})$ and $p_{y|\mathbf{x}}(y|\mathbf{x})$
- Methods include:
 - Self-training (or self-teaching)
 - Probabilistic generative models (Mixture Models and EM)
 - Self-supervised representation learning + supervised fine tuning
 - Etc.

- Learning process uses its own predictions to **teach itself**
- **Algorithm:**
 - Input: labeled data $\{\mathbf{x}_i, y_i\}_{i=1}^L$, unlabeled instances $\{\mathbf{x}_i\}_{i=L+1}^{L+U}$
 - 1. Initialization: $\mathbb{L} = \{\mathbf{x}_i, y_i\}_{i=1}^L$ and $\mathbb{U} = \{\mathbf{x}_i\}_{i=L+1}^{L+U}$
 - 2. Repeat:
 - 3. Train f from \mathbb{L} using supervised learning
 - 4. Apply f to \mathbb{U}
 - 5. Remove a subset \mathbb{S} from \mathbb{U} ; add $\{(\mathbf{x}, f(\mathbf{x})), \mathbf{x} \in \mathbb{S}\}$ to \mathbb{L}
- Typically, \mathbb{S} = unlabeled instances with the **most confident** f predictions

- Log-likelihood function ($\boldsymbol{\theta}$ is the set of all parameters):

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^L \log p_y(y_i|\boldsymbol{\theta}) p_{\mathbf{x}|y}(\mathbf{x}_i|y_i, \boldsymbol{\theta}) + \sum_{i=L+1}^{L+U} \log p_{\mathbf{x}}(\mathbf{x}_i|\boldsymbol{\theta})$$

- Marginal distribution:

$$p_{\mathbf{x}}(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{y=1}^c p_y(y_i|\boldsymbol{\theta}) p_{\mathbf{x}|y}(\mathbf{x}_i|y_i, \boldsymbol{\theta})$$

- Semi-supervised learning amounts to finding MLE of $\boldsymbol{\theta}$. Intuitively, this MLE will need to fit both labeled and unlabeled instances
- Hidden variables $\mathcal{H} = \{y_i\}_{i=L+1}^{L+U}$ can make the LLF non-concave and hard to optimize
- Efficient algorithms exist to find a local optimum, e.g. Expectation Maximization (EM)

■ EM algorithm:

Input: observed data \mathcal{D} , hidden data \mathcal{H} , initial parameter $\theta^{(0)}$

1. Initialize $t = 0$.

2. Repeat the following steps until $p(\mathcal{D}|\theta^{(t)})$ converges:

3. E-step: compute $q^{(t)}(\mathcal{H}) \equiv p(\mathcal{H}|\mathcal{D}, \theta^{(t)})$

4. M-step: find $\theta^{(t+1)}$ that maximizes $\sum_{\mathcal{H}} q^{(t)}(\mathcal{H}) \log p(\mathcal{D}, \mathcal{H}|\theta^{(t+1)})$

5. $t = t + 1$

Output: $\theta^{(t)}$

- $q^{(t)}(\mathcal{H})$: hidden label distribution, i.e. assigning ‘soft labels’ to unlabeled data according to current model θ
- EM is a special form of self-training...

1. Initialize $t = 0$ and $\theta^{(0)} = \{\pi_j^{(0)}, \mu_j^{(0)}, \Sigma_j^{(0)}\}_{j \in \{1,2\}}$ to the MLE estimated from labeled data.
2. Repeat until the log likelihood $\log p(\mathcal{D}|\theta)$ converges:
3. E-step: For all unlabeled instances $i \in \{l+1, \dots, l+u\}$, $j \in \{1, 2\}$, compute

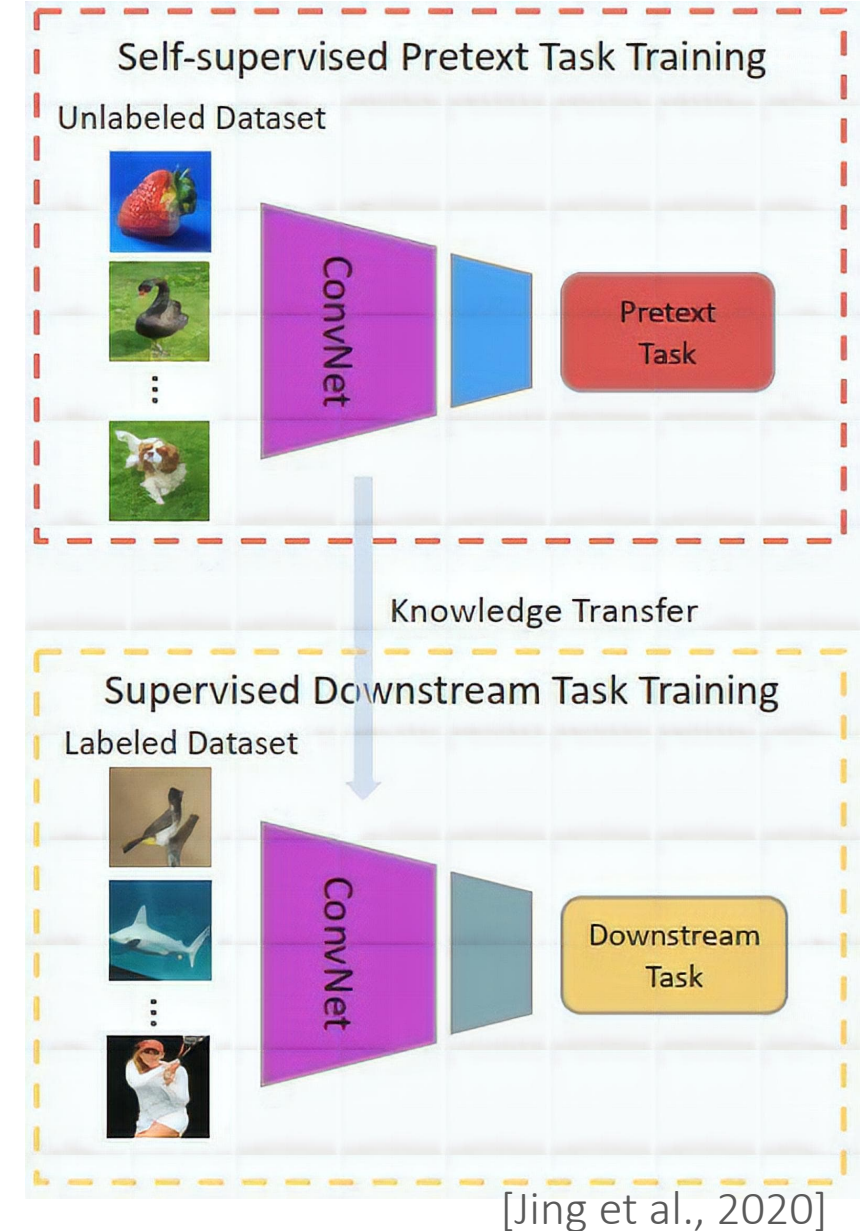
$$\gamma_{ij} \equiv p(y_j | \mathbf{x}_i, \theta^{(t)}) = \frac{\pi_j^{(t)} \mathcal{N}(\mathbf{x}_i; \mu_j^{(t)}, \Sigma_j^{(t)})}{\sum_{k=1}^2 \pi_k^{(t)} \mathcal{N}(\mathbf{x}_i; \mu_k^{(t)}, \Sigma_k^{(t)})}.$$

For labeled instances, define $\gamma_{ij} = 1$ if $y_i = j$, and 0 otherwise.

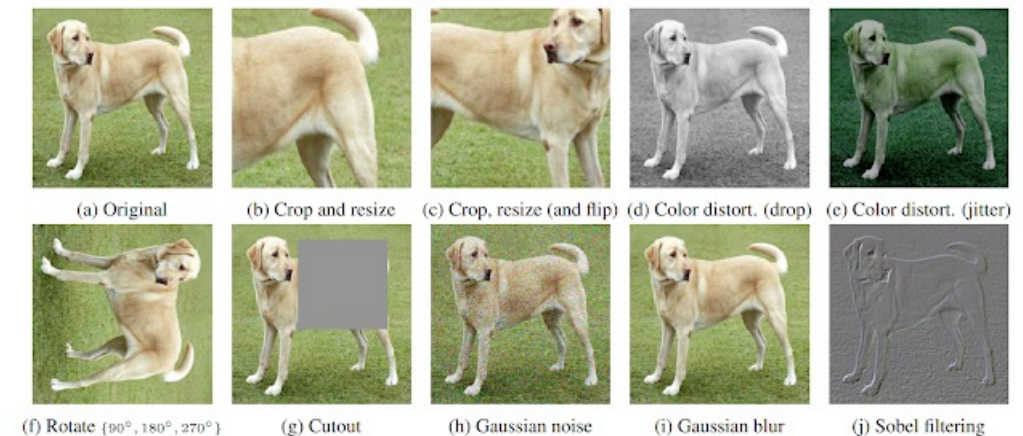
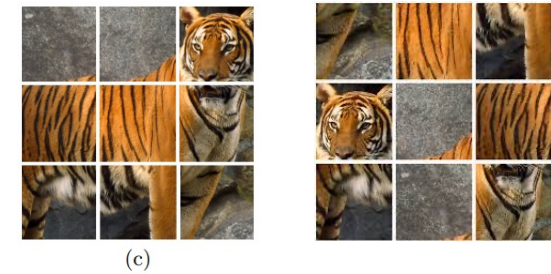
4. M-step: Find $\theta^{(t+1)}$ using the current γ_{ij} . For $j \in \{1, 2\}$,

$$\begin{aligned} l_j &= \sum_{i=1}^{l+u} \gamma_{ij} & \pi_j^{(t+1)} &= \frac{l_j}{l+u} \\ \mu_j^{(t+1)} &= \frac{1}{l_j} \sum_{i=1}^{l+u} \gamma_{ij} \mathbf{x}_i & \Sigma_j^{(t+1)} &= \frac{1}{l_j} \sum_{i=1}^{l+u} \gamma_{ij} (\mathbf{x}_i - \mu_j^{(t+1)}) (\mathbf{x}_i - \mu_j^{(t+1)})^\top \end{aligned}$$

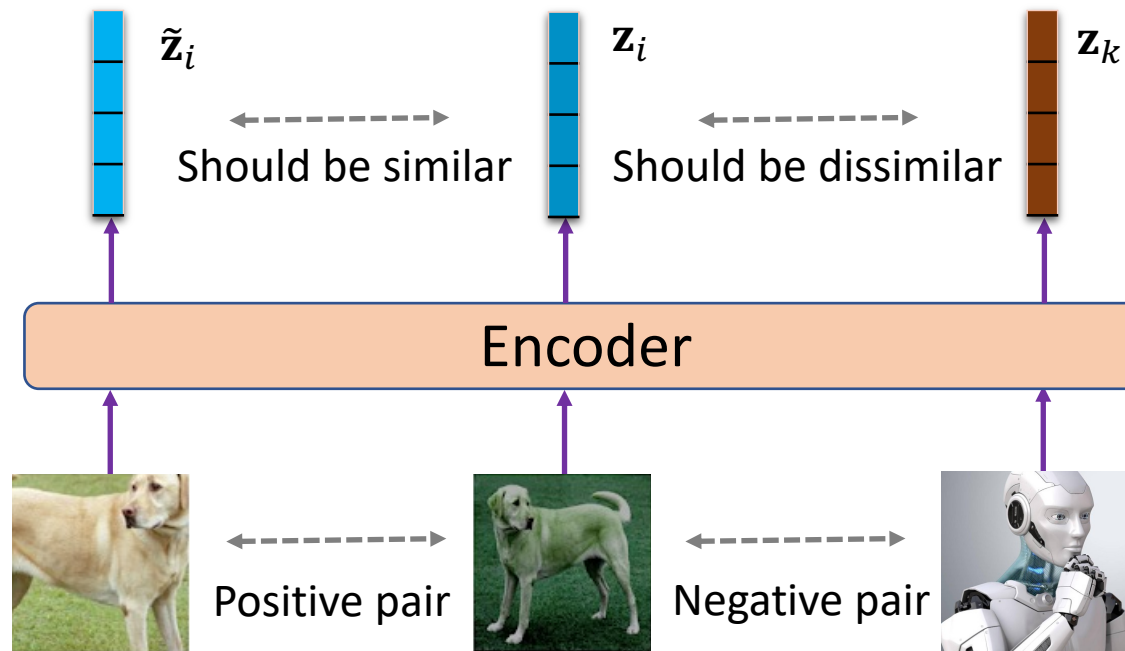
- Self-supervised learning defines a **pretext task** based on unlabeled inputs to produce descriptive and intelligible representations [Hastie et al., 2009]
- By solving pretext tasks, the network learns representations which are useful for downstream tasks



- **Generative approaches:** recover masked information (e.g. token, color, pixel)
 - Non-autoregressive or Autoregressive
- **Predictive tasks:** predict designed labels, e.g.
 - predict the context (e.g. predict locations of image patches or right order of frames)
 - Predict image rotation angle
- **Contrastive learning:** set up a binary classification problem based on positive and negative sample pairs created by augmentation



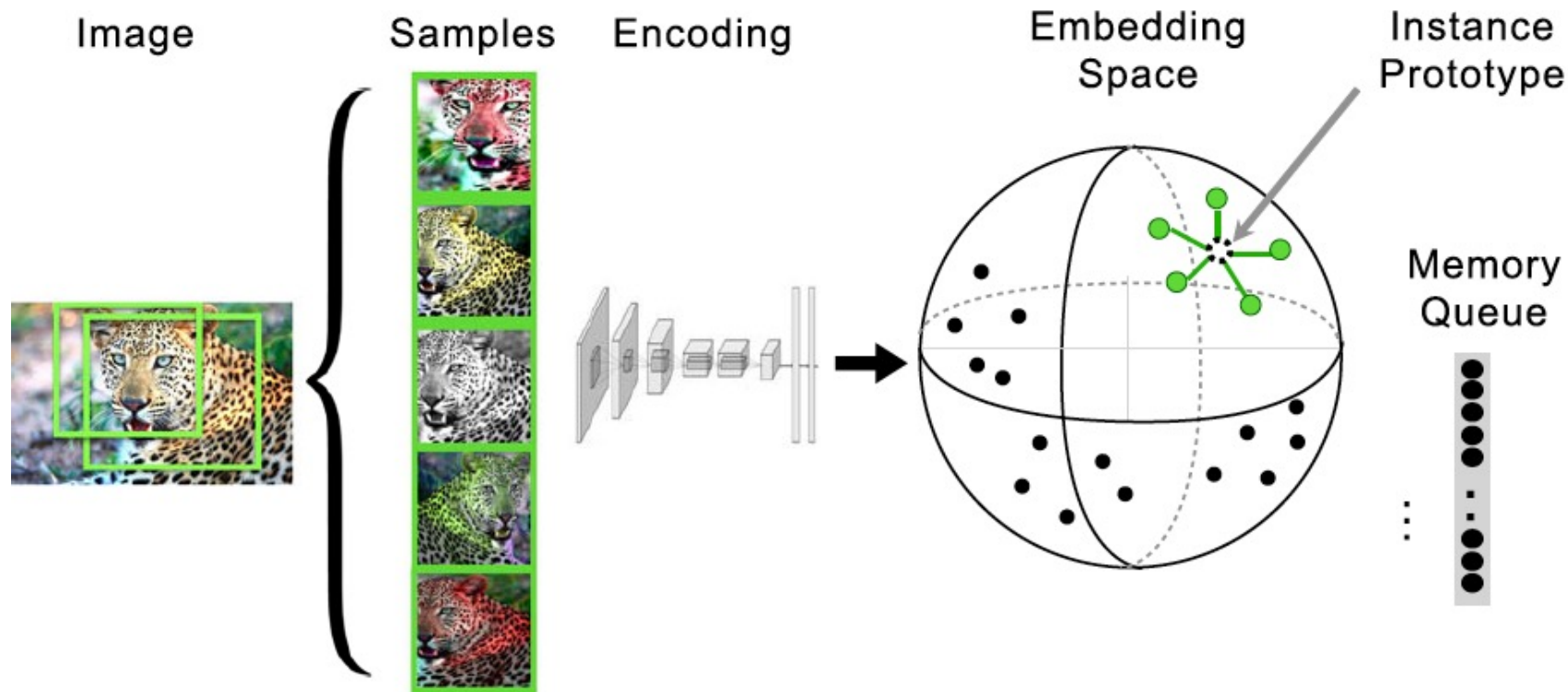
- Generate variants of a single input using known semantic preserving transformations
- Training a network to maximize the mutual information between the representations of an image under different views [Chen et al., 2020]



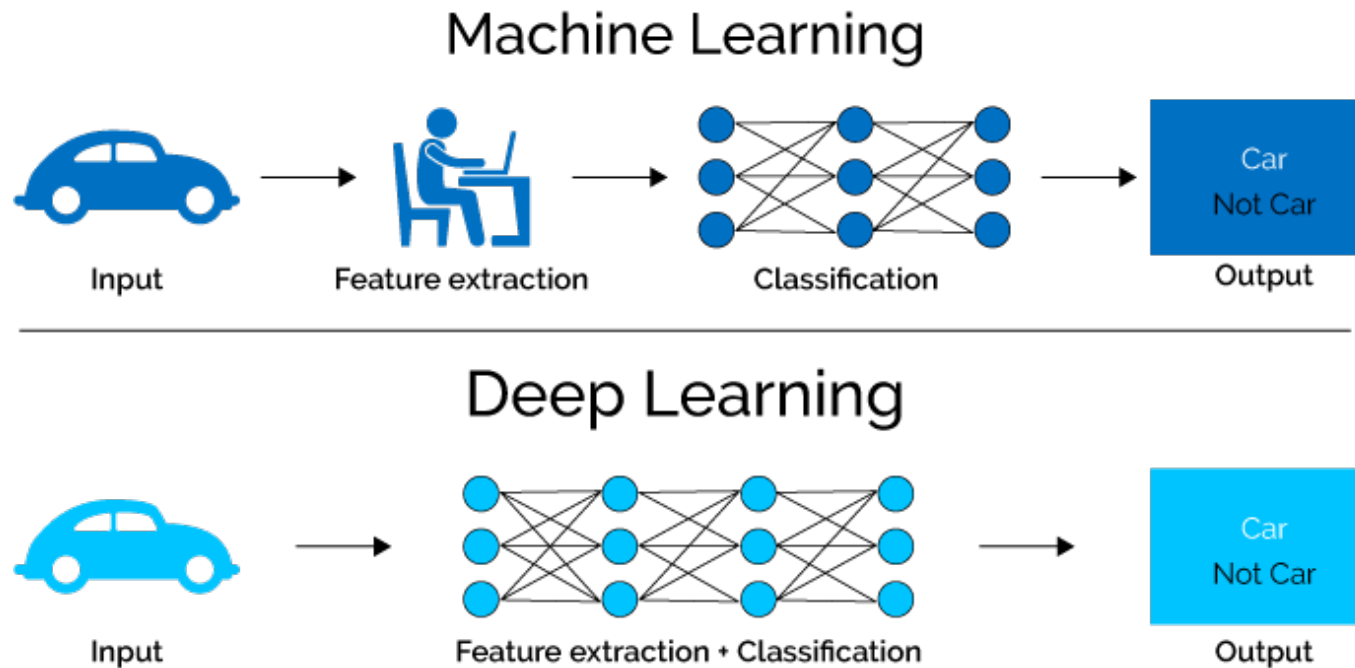
$$\text{Loss} = -\log \frac{\exp\left(\frac{\text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i)}{\tau}\right)}{\exp\left(\frac{\text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_i)}{\tau}\right) + \sum_{k=1}^K \exp\left(\frac{\text{sim}(\mathbf{z}_i, \tilde{\mathbf{z}}_k)}{\tau}\right)}$$

- Instance-level contrastive learning yields human brain-like representation without category-supervision [Konkle et al., 2022].

Instance-Prototype Contrastive Learning



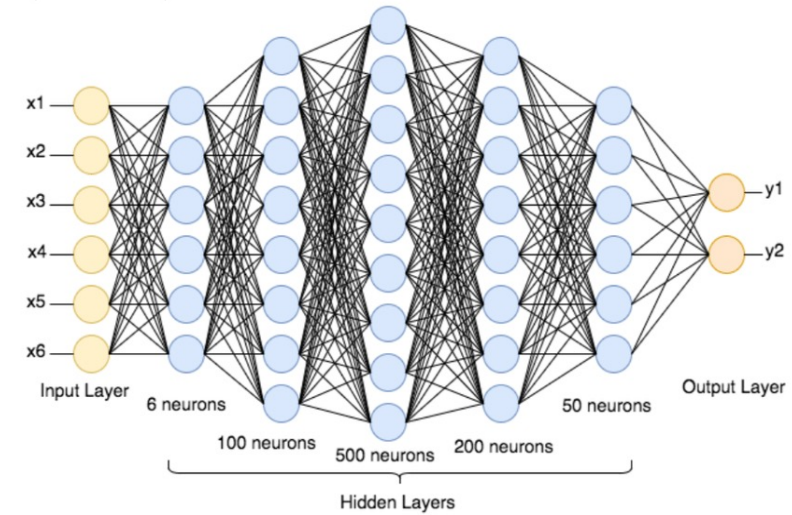
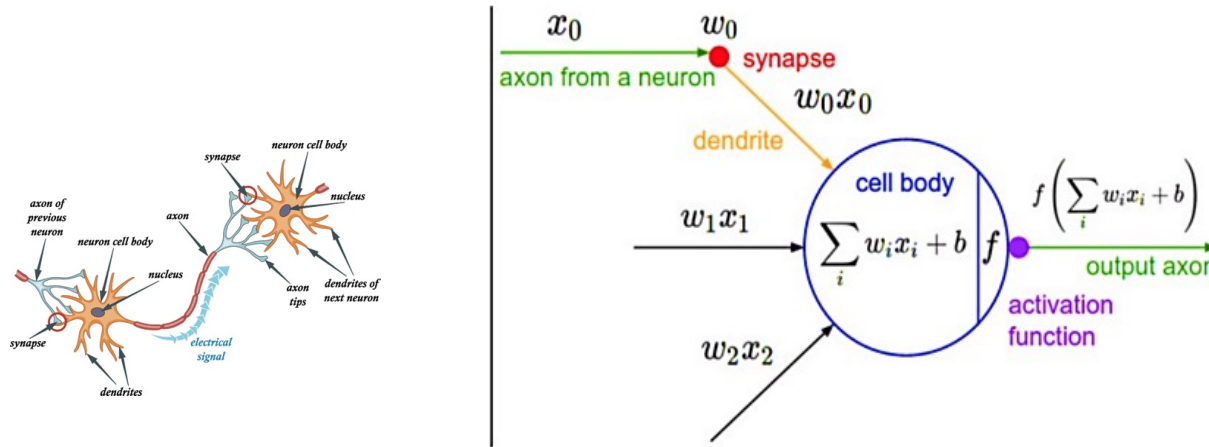
- In conventional Machine Learning, the **features are engineered by domain experts**
- In deep learning, feature engineering is part of the learning process; (multiple levels of) representations are learnt by using a **hierarchy of multiple NN layers**



Artificial neural network

Multilayer perceptron (MLP)

- MLP is a class of feedforward artificial neural network (ANN)



Nonlinear activation functions

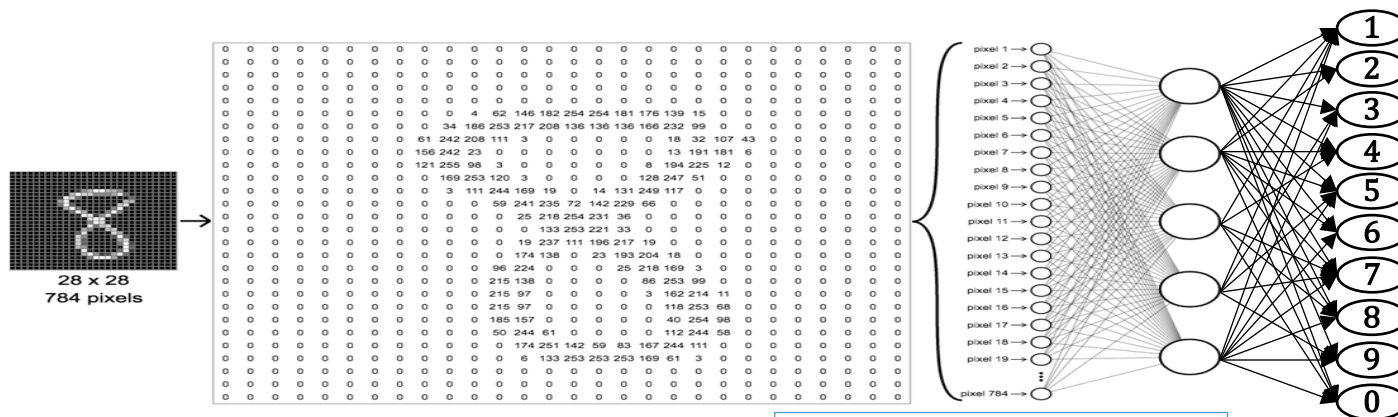
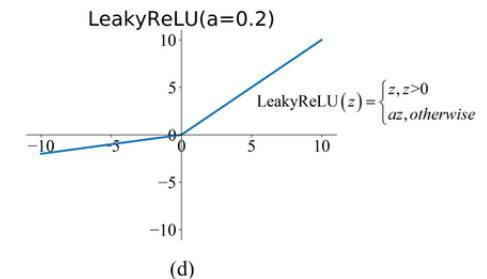
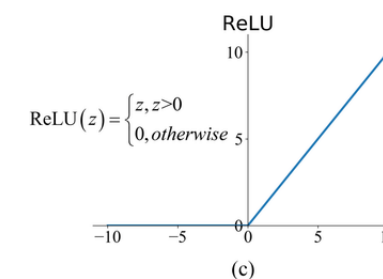
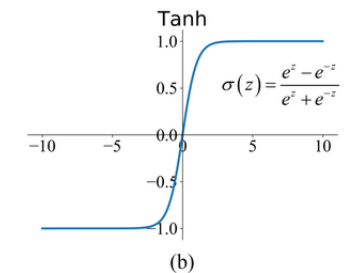
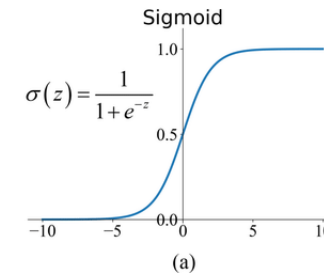


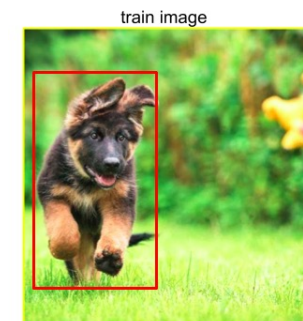
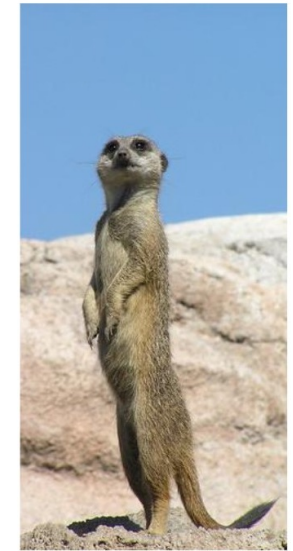
Image pixels vectorized so each neuron take a pixel



Issue #1: number of parameters increases with size of input (e.g. for 224x224 RGB images, we have 150 528 input features and the hidden layer has 1000 nodes, the MLP would have $150\,528 \times 1024 = 150+$ million weights in the first layer alone!

Issue #2: fixed input size. Different MLPs must be trained for different input sizes

Issue #3: MLP is not shift invariant, i.e. poor performance if object's position changes at inference



Convolutional Neural Networks (CNN) address these 3 limitations

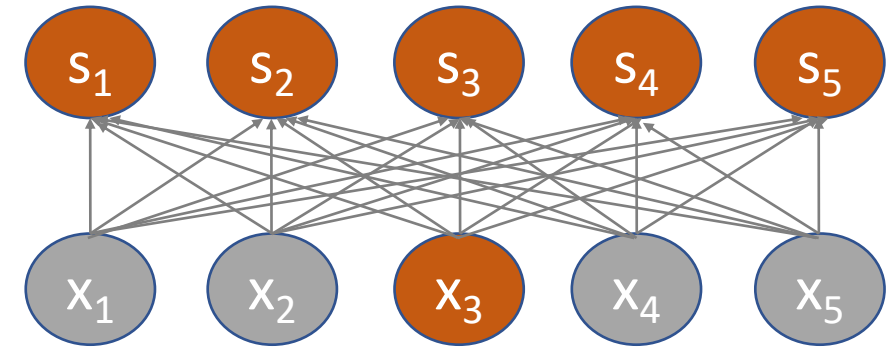
- **Obervation #1:** in natural images, objects in images are made up of small, localized features. So, it seem wasteful for every node in the first hidden layer to look at every pixel?
→ **sparse interactions**

- **Obervation #2:** Similar localized features can be processed similarly regardless of their position
→ **parameter sharing**

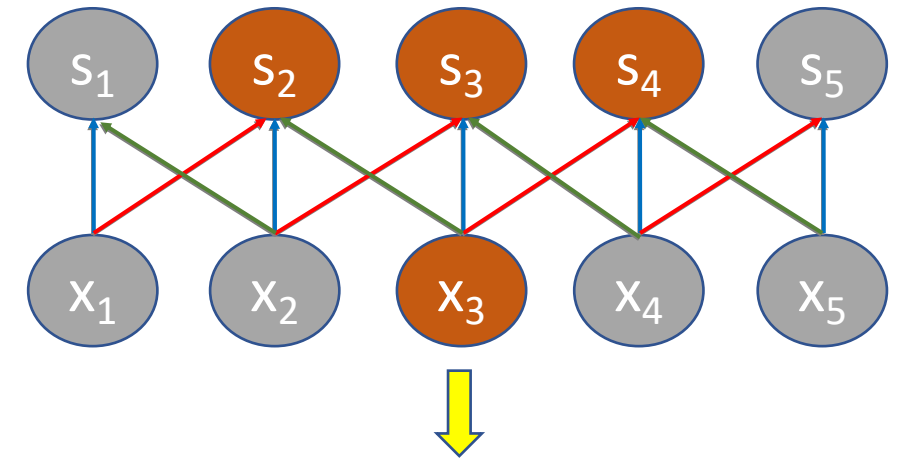
CNN \equiv

Multiple convolutional layers; each layer consists of a dozen of filters in parallel
+ Regularization techniques
+ fully connected layers

Dense interactions in MLP

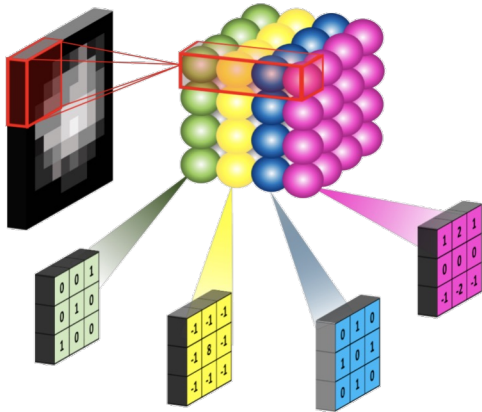


Sparse interactions + Parameter sharing

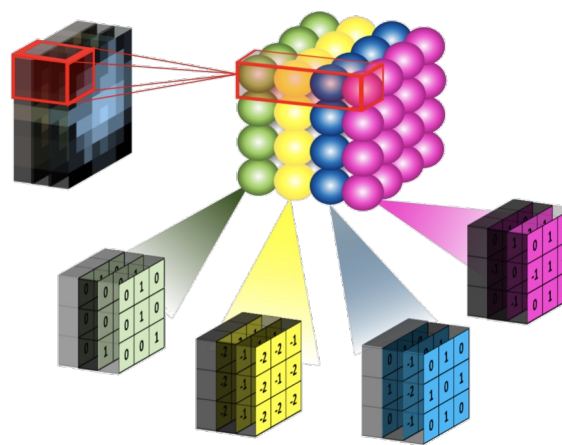


\equiv applying filters (convolution operations)

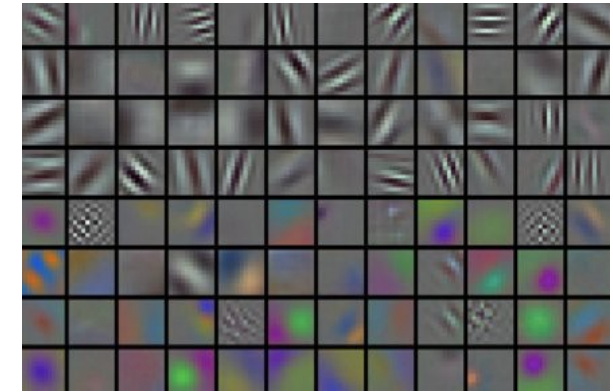
2D filters on a grayscale image



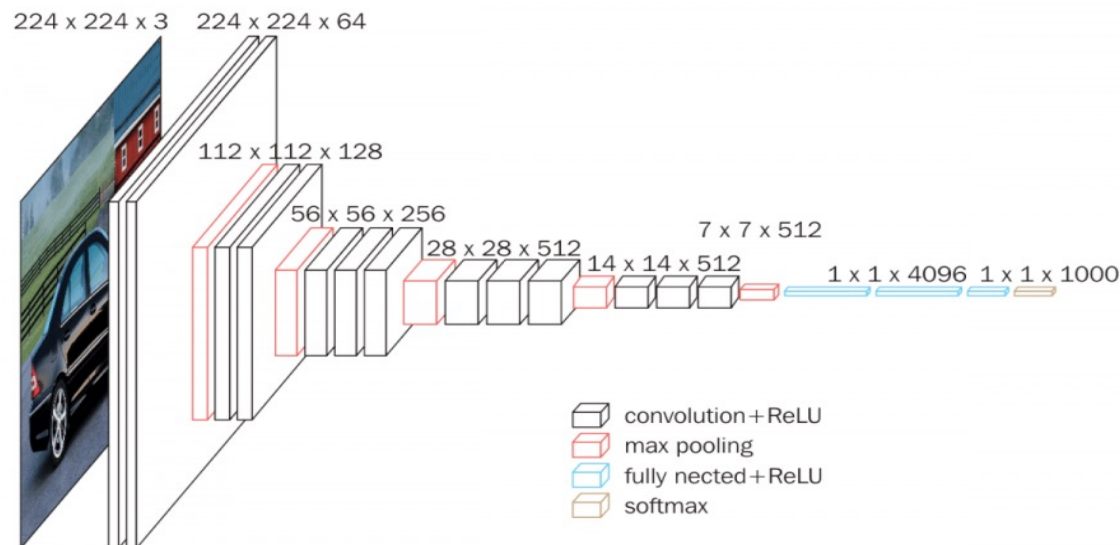
3D filters on an RGB image



Each conv layer produces a feature map



Example: **VGG16** (138M+ parameters)



- 13 conv layers, 5 pooling layers, and 3 FC layers, so 21 layers but only 16 have learnable weights
- Conv layers: same padding, Receptive field of filters= 3x3, Stride=1
- Pooling layers: max pooling 2x2 and stride =2

Outline

Introduction

- Attributed graphs
- ML tasks on graphs
- Very brief introduction to ML

Graph embedding

- Direct encoding
- Graph neural networks (GNN)

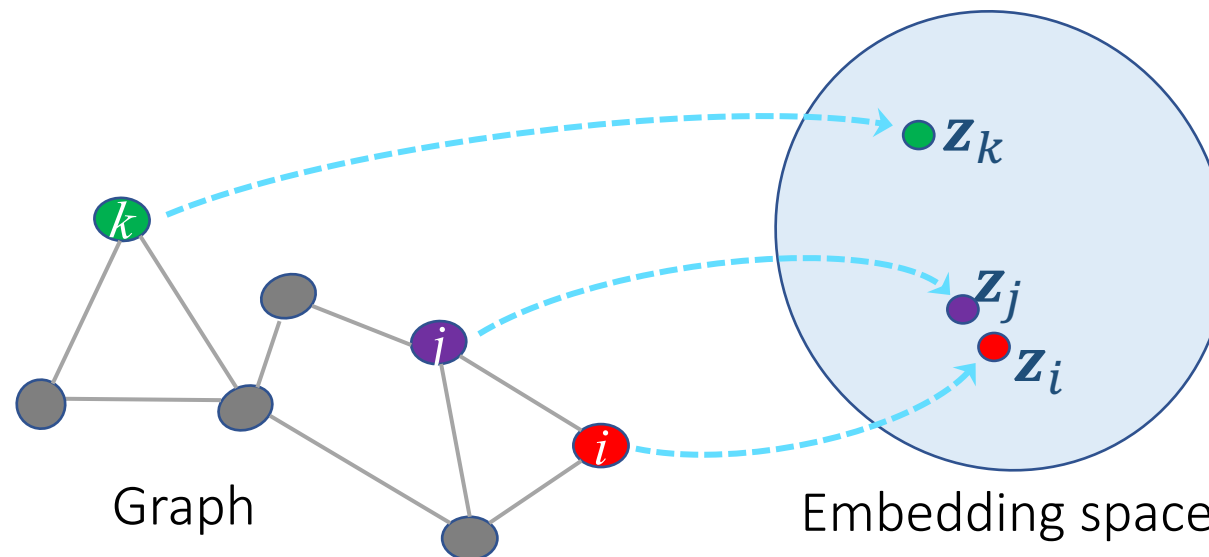
Node & graph Classification

- Collective classification
- GNN-based classifiers
- Graph-assisted Bayesian classifiers

Very brief introduction to Knowledge graphs

- KG embedding
- KG completion

- **Objective:** represent nodes in a latent space, where geometric relations correspond to relationships (e.g. edges) in the graph
- Nodes that are **similar** to each other according to some proximity measure should be close to each other in a low-dimensional embedding space
- Proximity is measured using the graph structure (edges) and also node features if any



■ Node (Edge) embedding

$$f(\text{graph}) = \begin{bmatrix} \text{vector} & \text{vector} & \dots & \text{vector} \end{bmatrix}$$

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) \in \mathbb{R}^{d \times |\mathcal{V}|} (\mathbb{R}^{d \times |\mathcal{E}|}), d \ll |\mathcal{V}|$$

- A vector for each node (edge) representing **both** node's features and graph structure
- Node embedding often preferred since $|\mathcal{V}| \ll |\mathcal{E}|$

■ Full Graph embedding

$$F(\text{features and graph structure}) = \begin{bmatrix} \text{vector} \end{bmatrix}$$

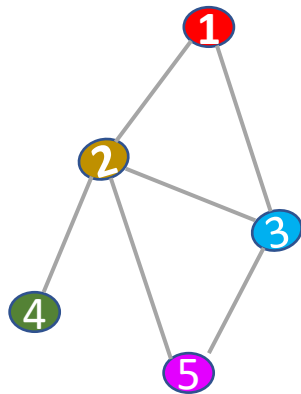
$$\mathbf{z}_G = F(\mathbf{X}, \mathbf{A}) \in \mathbb{R}^d, d \ll |\mathcal{V}|$$

Generally based on aggregation of node embeddings

The generated embedding can be used as input for downstream ML tasks

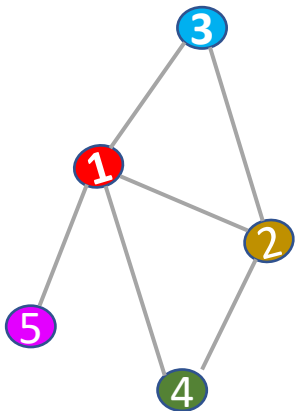
- Permutation invariance: graph embedding invariant to nodes permutation

$$F(\mathbf{PX}, \mathbf{PAP}^T) = F(\mathbf{X}, \mathbf{A})$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix};$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \\ \mathbf{x}_4^T \\ \mathbf{x}_5^T \end{bmatrix}; \quad \mathbf{z}_{\mathcal{G}} = F(\mathbf{X}, \mathbf{A})$$



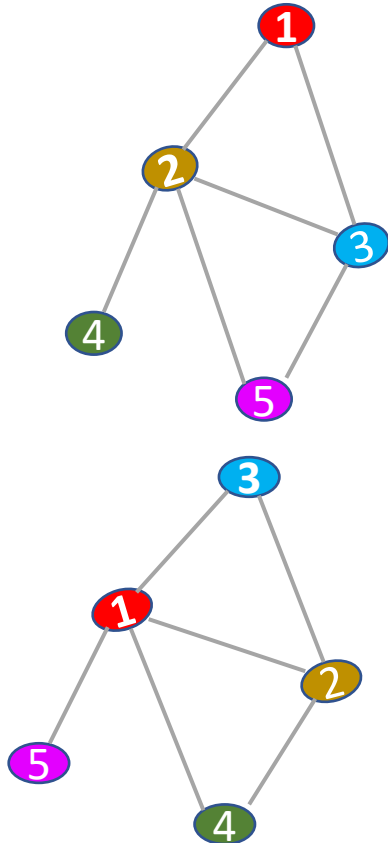
$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}; \quad \mathbf{PX} = \begin{bmatrix} \mathbf{x}_3^T \\ \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_5^T \\ \mathbf{x}_4^T \end{bmatrix};$$

$$\mathbf{PAP}^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}; \quad \mathbf{z}_{\mathcal{G}} = F(\mathbf{PX}, \mathbf{PAP}^T)$$

- f is permutation equivariant if for all permutation matrices, we have that

$$f(\mathbf{PX}, \mathbf{PAP}^T) = \mathbf{P}f(\mathbf{X}, \mathbf{A})$$

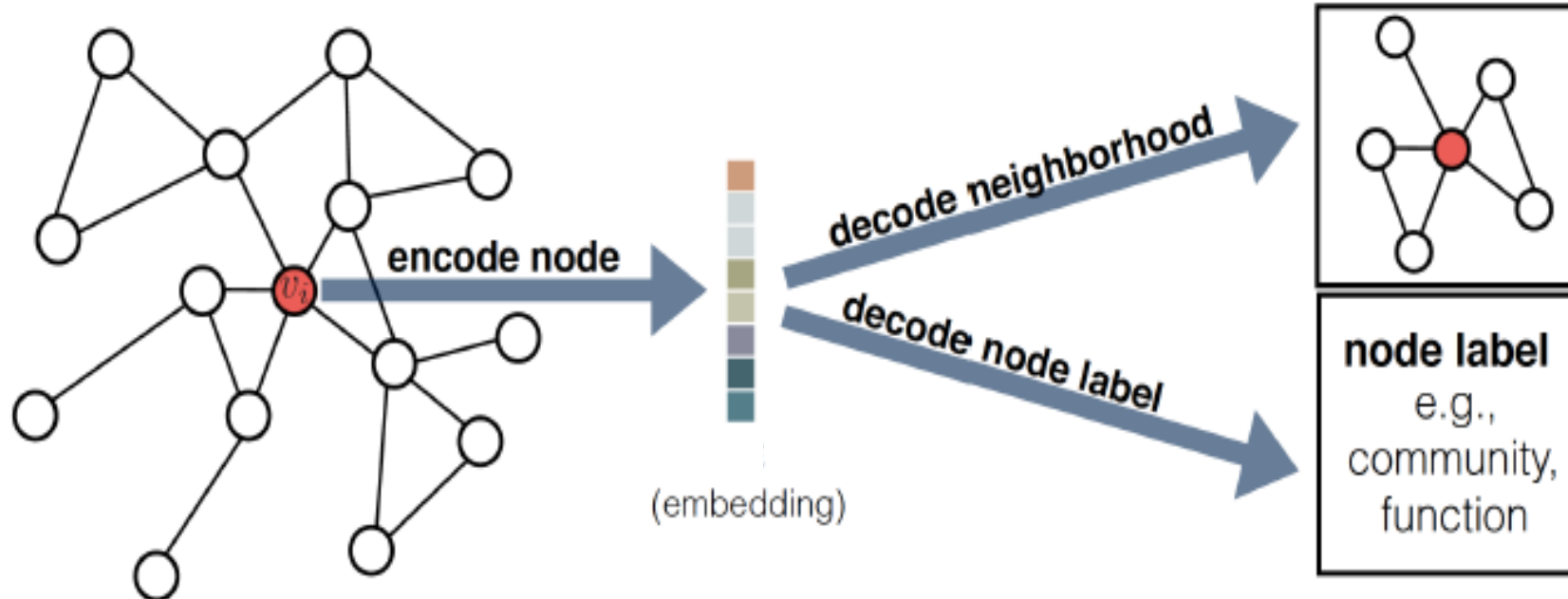
i. e. permutation of the nodes could be done before or after the function f



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}; \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \\ \mathbf{x}_4^T \\ \mathbf{x}_5^T \end{bmatrix}; \quad \mathbf{Z} = f(\mathbf{X}, \mathbf{A})$$

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}; \quad f(\mathbf{PX}, \mathbf{PAP}^T) = \mathbf{PZ}$$

- The **encoder** maps the node to a low-dimensional vector embedding, based on the node's position in the graph, its local neighborhood structure, and its attributes
- The **decoder** extracts user-specified information from the low-dimensional embedding, e.g. information about the node's local graph neighborhood or a node classification label [Hamilton et al. 2020]



- The decoder generally maps every pair of node embeddings to a real-valued graph proximity measure:

$$\text{ENC: } \mathcal{V} \rightarrow \mathbb{R}^d \quad \text{DEC: } \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$$

- **Goal:** optimize the encoder and decoder mappings to minimize the error/loss in the proximity reconstruction

$$\text{DEC}(\text{ENC}(v_i), \text{ENC}(v_j)) = \text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \approx s_{\mathcal{G}}(v_i, v_j)$$

- Examples of the proximity measure:

$$s_{\mathcal{G}}(v_i, v_j) = A_{i,j}$$

$$s_{\mathcal{G}}(v_i, v_j) = \text{Prob. of } v_i \text{ and } v_j \text{ co-occurring on} \\ \text{a fixed length random walk over } \mathcal{G}$$

- Optimizing the reconstruction objective is generally done by minimizing an empirical loss over the encoder's parameters using a set of training node pairs, \mathcal{D} :

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \ell \left(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j), s_{\mathcal{G}}(v_i, v_j) \right)$$

- ℓ : user-specified loss function which measures the discrepancy between the decoded/estimated proximity values, $\text{DEC}(\mathbf{z}_i, \mathbf{z}_j)$ and the true values, $s_{\mathcal{G}}(v_i, v_j)$
- The different embedding methods differ in the way, the encoding function, decoding function, proximity function and loss function are defined

- Lookup table

$$\text{ENC}(v_i) = \mathbf{Z}\delta_i$$

$$\delta_i = [0, \dots, 0, \underset{\uparrow}{1}, 0, \dots, 0]^T$$

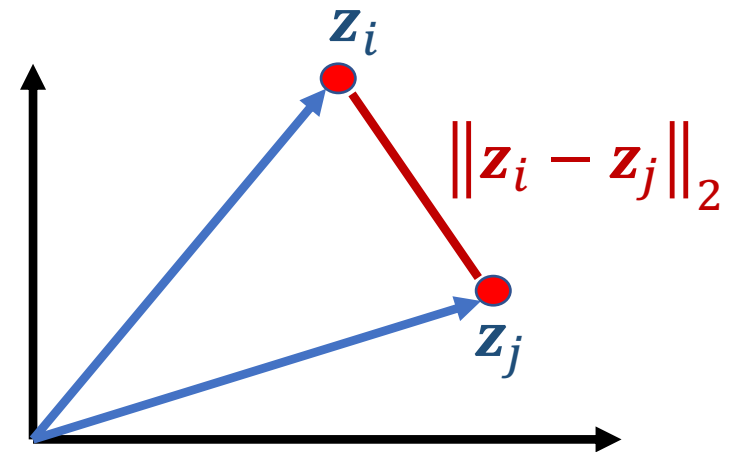
*i*th position

- The set of trainable parameters for the encoding is simply the embedding matrix \mathbf{Z}
- Direct encoding is transductive learning, so, the training set \mathcal{D} should involve all nodes (not necessarily all possible pairs). If a new node is added, the encoding must be optimized again for all nodes
- Many of these approaches were originally motivated as factorization algorithms

- Decoding and loss in LE method are [Belkin et al., 2002]:

$$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$$

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_g(v_i, v_j)$$

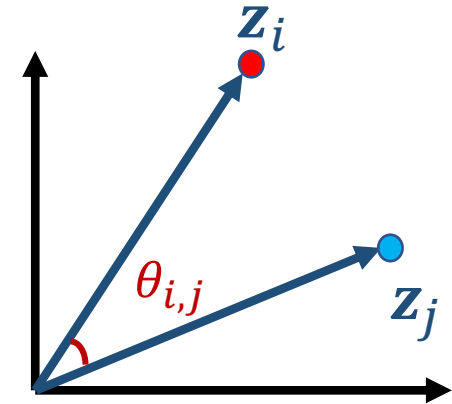


- Decoding and loss in IP methods:

$$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^T \mathbf{z}_j$$

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \left(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j) \right)^2$$

$$\mathbf{z}_i^T \mathbf{z}_j = \cos(\theta_{i,j}) \|\mathbf{z}_i\|_2 \|\mathbf{z}_j\|_2$$



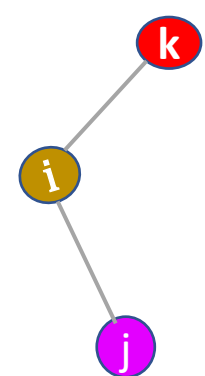
- If vectors are normalized (often the case):

$$\|\mathbf{z}_i - \mathbf{z}_j\|_2^2 = 2(1 - \mathbf{z}_i^T \mathbf{z}_j) = 2(1 - \cos(\theta_{i,j}))$$

- In Graph factorization (GF) method (IP type) [Ahmed et al., 2013], $s_g(v_i, v_j) = A_{i,j}$

$$\mathcal{L}_{\text{GF}} = \sum_{\substack{(v_i, v_j) \in \mathcal{D} \\ (i,j) \in \mathcal{E}}} (\mathbf{z}_i^T \mathbf{z}_j - 1)^2 + \sum_{\substack{(v_i, v_j) \in \mathcal{D} \\ (i,j) \notin \mathcal{E}}} (\mathbf{z}_i^T \mathbf{z}_j)^2$$

↑ Alignment ↑ Orthogonality



$\mathbf{z}_i^T \mathbf{z}_j \rightarrow 1$
 $\mathbf{z}_i^T \mathbf{z}_k \rightarrow 1$
 $\mathbf{z}_j^T \mathbf{z}_k \rightarrow 0$

- If $\mathcal{D} = \{(v_i, v_j)\}_{(i,j) \in \mathcal{E}}$ i.e. no constraints on embeddings of nonconnected nodes:

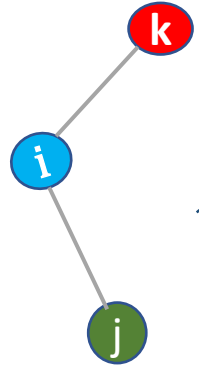
Trivial solution: all nodes have the same embedding (Collapse)!

Direct encoding

Factorization Approach

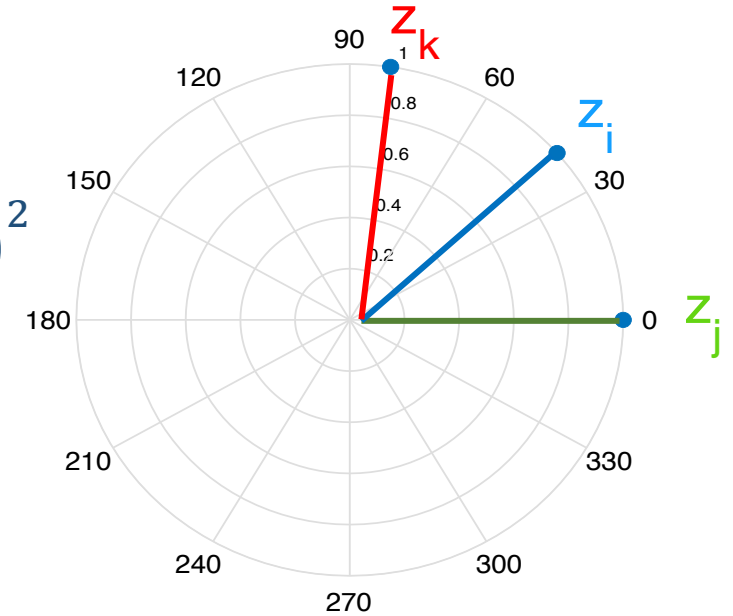
Graph Factorization method

Example 1:



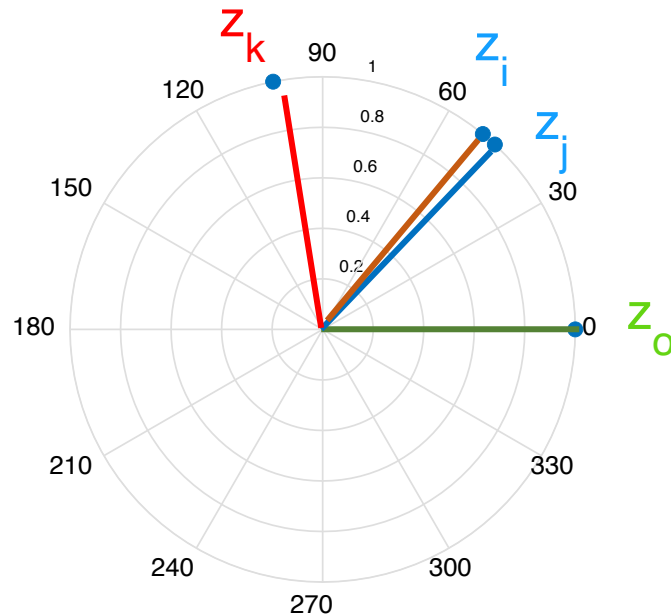
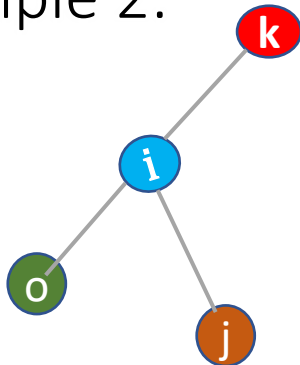
$$\mathcal{L}_{\text{GF}} = (\mathbf{z}_i^T \mathbf{z}_j - 1)^2 + (\mathbf{z}_i^T \mathbf{z}_k - 1)^2 + (\mathbf{z}_j^T \mathbf{z}_k)^2$$

$$\mathcal{L}_{\text{GF}} = (\cos(\theta_{ij}) - 1)^2 + (\cos(\theta_{ik}) - 1)^2 + (\cos(\theta_{jk}))^2$$



$$\theta_{ij} = \theta_{ki} = 0.7104$$

Example 2:

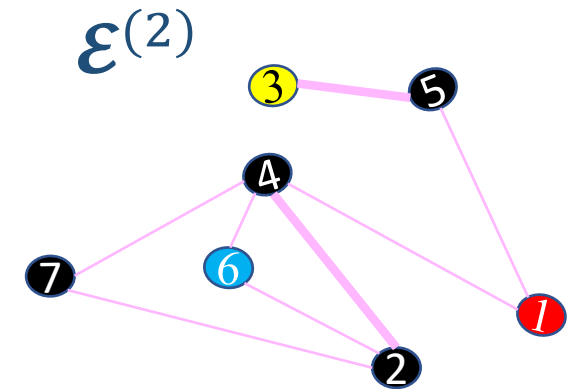
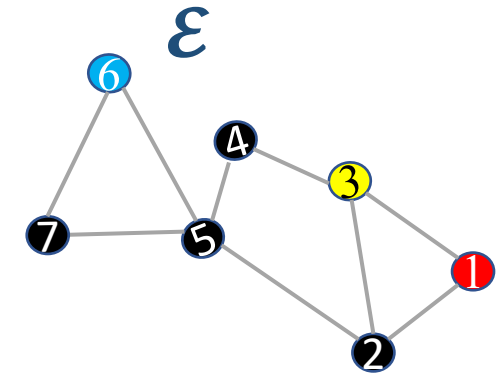


$$\theta_{io} = 0.8841; \theta_{ko} = 1.7681; \theta_{jo} = 0.8209$$

- $s_{\mathcal{G}}(v_i, v_j) = [\tilde{\mathbf{A}}^n]_{i,j}$, with $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$, $n \geq 1$:

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \left(\mathbf{z}_i^T \mathbf{z}_j - [\tilde{\mathbf{A}}^n]_{i,j} \right)^2$$

- [Cao et al., 2015] concatenated embeddings for different values of n



Direct encoding

Factorization Approach

Other methods

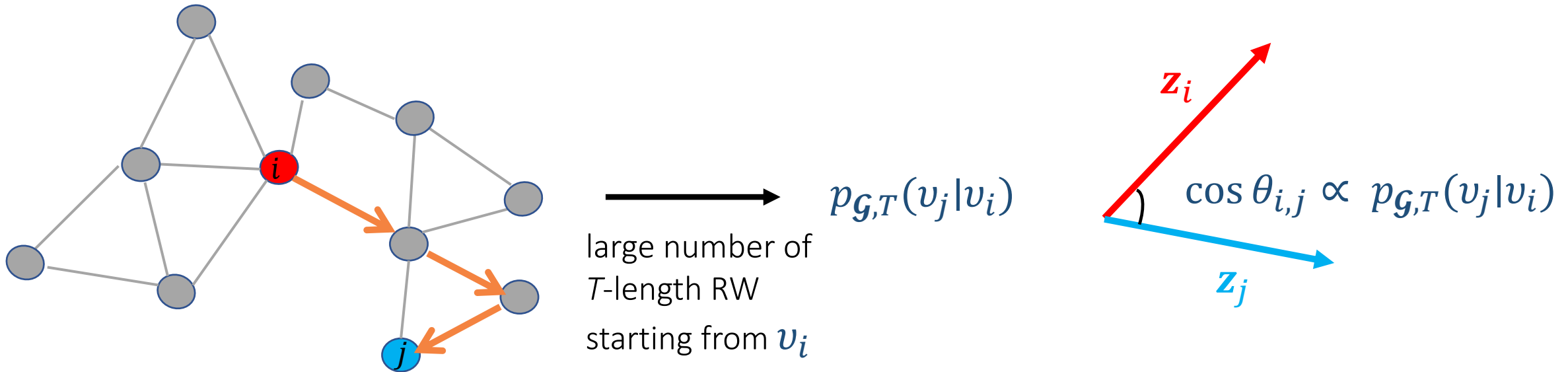
- HOPE algorithm [Ou et al., 2016]: $s_{\mathcal{G}}(v_i, v_j) = \text{Jaccard}$
- Generally, $s_{\mathcal{G}}(v_i, v_j)$ is designed to trade-off 1st-order and higher-order proximities
- These methods are referred to as matrix-factorization approaches because:

$$\mathcal{L} \approx \|\mathbf{Z}^T \mathbf{Z} - \mathbf{S}\|_F^2$$

$$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}, \mathbf{S} \in \mathbb{R}_+^{|\mathcal{V}| \times |\mathcal{V}|}$$

- Problem solved using SVD or stochastic gradient descent
- But these methods use a **deterministic** measure of graph proximity.

- **Idea**: similar embeddings for nodes that likely co-occur on short random walks over \mathcal{G}
- Use stochastic measure of graph proximity



Typical length of RW: $T = \{2, 3, \dots, 10\}$

- **Goal:** learn the encoding function so that [B. Perozzi, 2014]

$$\text{DEC}(\mathbf{z}_i, \mathbf{z}_j) \triangleq \frac{e^{\mathbf{z}_i^T \mathbf{z}_j}}{\sum_{v_k \in \mathcal{V}} e^{\mathbf{z}_i^T \mathbf{z}_k}} \approx p_{\mathcal{G}, T}(v_j | v_i)$$

Similarity measure is both stochastic and asymmetric

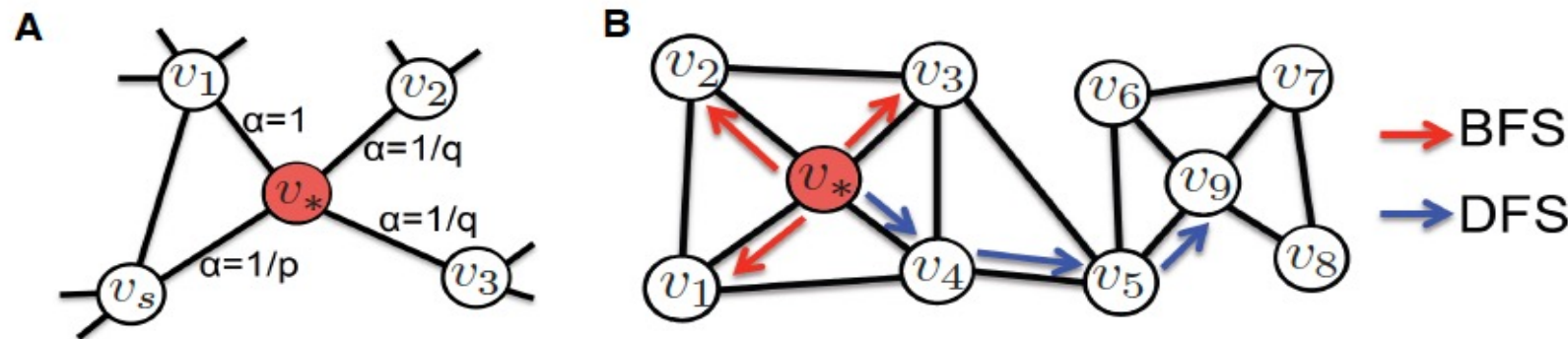
- Loss function:

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} -p_{\mathcal{G}, T}(v_j | v_i) \log(\text{DEC}(\mathbf{z}_i, \mathbf{z}_j))$$

\mathcal{D} : training set of random walks, generated by sampling random walks starting from each node.

- Reduced complexity versions have been proposed

- Same decoding as in DeepWalk, but random walks are designed differently
- Node2Vec introduces two hyperparameters:
 - p controls the likelihood of the walk immediately revisiting a node
 - q controls the likelihood of the walk revisiting a node's one-hop neighborhood



[Grover et al., 2016]

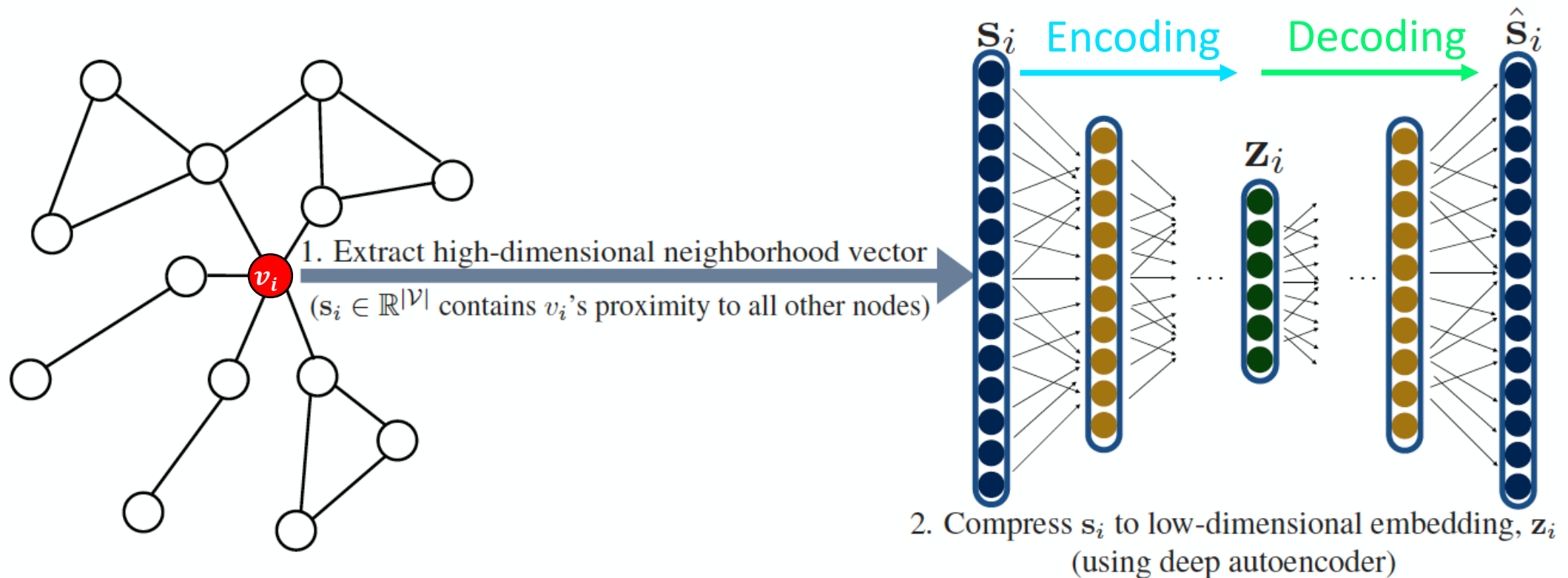
- Loss function approximated using negative samples

- Summary of some well-known direct (shallow) embedding algorithms

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\ \mathbf{z}_u - \mathbf{z}_v\ _2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], \dots, \mathbf{A}^k[u, v]$	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\ \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\ _2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v u)$	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v u)$ (biased)	$-\mathbf{S}[u, v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

- No parameter sharing between nodes, so:
 - statistical inefficiency
 - computational inefficiency as number of parameters = $O(|\mathcal{V}|)$
- Direct encoding methods fail to leverage node attributes during encoding
- Direct encoding methods are inherently transductive

- **Idea**: compress a high-dimensional proximity vector for each node, \mathbf{s}_i (i th row of \mathbf{S}), and reconstruct the proximity vector using a decoder



- Encoder and decoder functions consist of multiple stacked neural network layers:

$$\text{ENC}(\mathbf{s}_i) = f(\mathbf{s}_i, \boldsymbol{\theta}_{\text{ENC}}); \quad \text{DEC}(\mathbf{z}_i) = g(\mathbf{z}_i, \boldsymbol{\theta}_{\text{DEC}});$$

- Encoder and decoder optimized such that:

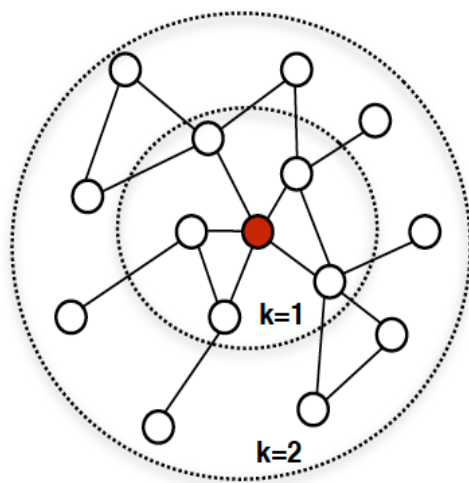
$$\text{DEC}(\text{ENC}(\mathbf{s}_i)) = \text{DEC}(\mathbf{z}_i) \approx \mathbf{s}_i$$

- Loss function:

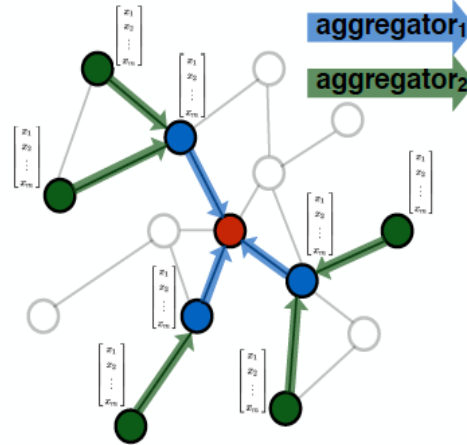
$$\mathcal{L} = \sum_{v_i \in \mathcal{V}} \|\text{DEC}(\mathbf{z}_i) - \mathbf{s}_i\|_2^2$$

- **Deep Neural Graph Representations** (DNGR) [Cao et al., 2016]: \mathbf{s}_i defined using pointwise mutual information of two nodes co-occurring on random walks (as in DeepWalk and Node2vec)
- **Structural Deep Network Embeddings** (SDNE) [Wang et al., 2016]: $\mathbf{s}_i = \mathbf{A}_i$, i.e., v_i 's adjacency vector. Encoder optimized by combining the above loss with the Laplacian eigenmaps objective
- Drawbacks of autoencoders:
 - Input dimension is fixed at $|\mathcal{V}|$, so intractability for very large graphs
 - Strictly transductive and cannot generalize to evolving or new graphs
 - Cannot leverage node features

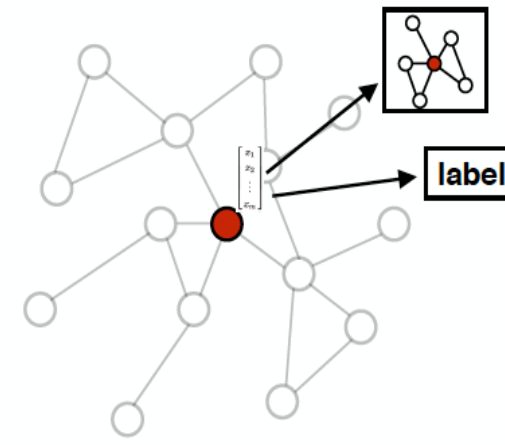
- **Idea:** generate node embeddings by aggregating information from local neighborhood
- Unlike direct encoding and autoencoder methods, this approach relies on node features/attributes



1. Collect neighbors



2. Aggregate feature information from neighbors



3. Decode graph context and/or label using aggregated information

[Hamilton et al., 2017]

- In the absence of node features, graph statistics may be used as features (e.g. node degrees), or simply use one-hot indicator vectors

- For node u , the (1-hop) neighborhood is defined as:

$$\mathcal{N}_u = \{v: (u, v) \in \mathcal{E} \vee (v, u) \in \mathcal{E}\}$$

- General formulation of neighborhood aggregation is based on the message passing model, i.e. hidden embedding for each node, $\mathbf{h}_u^{(k)}$, is updated as ($\mathbf{h}_u^{(0)} = \mathbf{x}_u$):

$$\mathbf{m}_{\mathcal{N}_u}^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ \mathbf{h}_v^{(k)} : v \in \mathcal{N}_u \right\} \right)$$

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}_u}^{(k)} \right)$$

- Final embedding is the representation at the last iteration, K , i.e. $\mathbf{z}_u = \mathbf{h}_u^{(K)}$
- Higher K involves higher-order neighbors

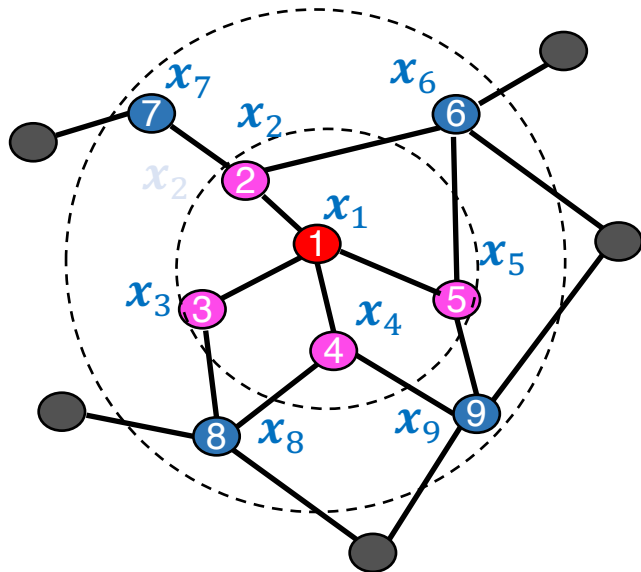
- Neighborhood aggregation is a **set function**, i.e. it maps the set of embeddings $\{\mathbf{h}_v^{(k)} : v \in \mathcal{N}_u\}$ to a single vector $\mathbf{m}_{\mathcal{N}_u}$. So, no natural ordering of the neighbors. The aggregation function must thus be **permutation invariant**
- Alternatives to the sum aggregator are average and pooling aggregators e.g. element-wise max or min as in [Qi et al., 2017]
- Aggregation of $\{\mathbf{h}_v^{(k)}, v \in \mathcal{N}_u\}$ may/may not depend on $\mathbf{h}_u^{(k)}$...
- Aggregation of $\{\mathbf{h}_v^{(k)}, v \in \mathcal{N}_u\}$ may/may not depend on local graph statistics (node degrees)...

Graph Neural Networks

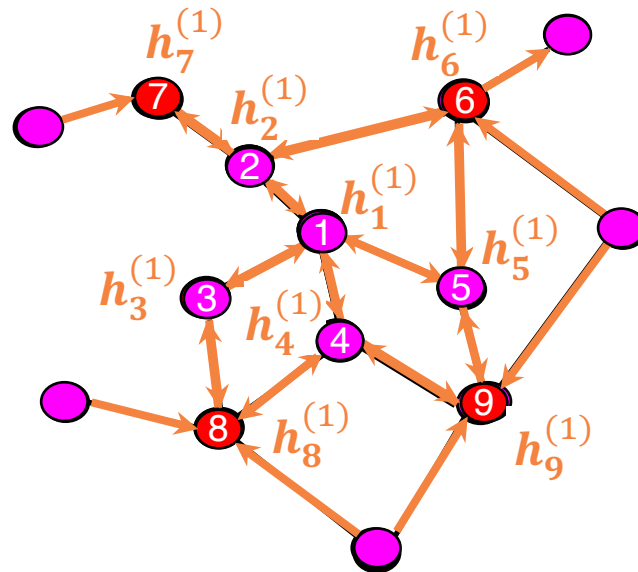
Neighborhood Aggregation

Message passing

■ Illustration

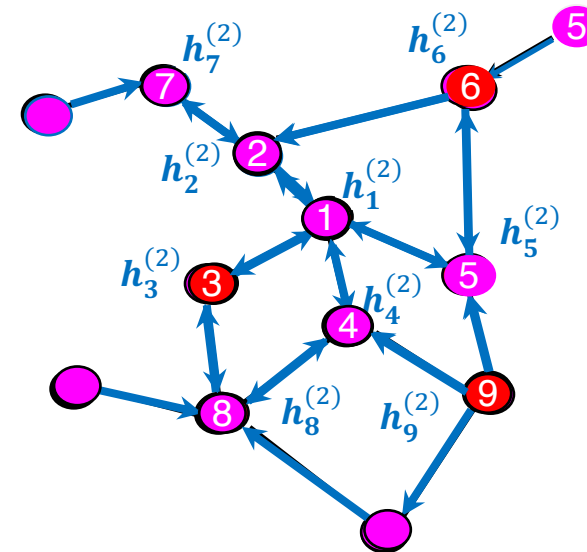


$$\mathbf{H}^{(0)} = \mathbf{X}$$



$$\mathbf{H}^{(1)}$$

Iteration $k = 1$



$$\mathbf{H}^{(2)}$$

Iteration $k = 2$

$$\mathbf{Z} = \mathbf{H}^{(K)}$$

Last iteration

- Higher K involves higher-order neighbors in node embedding. Typically $K \in \{1, 2, 3\}$.

- GNN node embedding captures both graph structure and node features information
- After k iterations, node embedding encodes information about all the features in their node's k -hop neighborhood.
- GNN's neighborhood aggregation behaviour is analogous to that of convolutional kernels in convolutional neural networks (CNNs)

- A simplified version of the neural message passing approach: add a self-loop and omit the explicit update step

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE} \left(\mathbf{h}_v^{(k-1)}, v \in \mathcal{N}_u \cup \{u\} \right)$$

- For example:

$$\mathbf{h}_u^{(k)} = \text{MLP}^{(k)} \left(\mathbf{h}_u^{(k-1)} + \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k-1)} \right)$$

- MLP (multilayer perceptron) often consists of a single hidden layer, i.e.

$$\text{MLP}^{(k)}(\mathbf{x}) = \sigma(\mathbf{W}^{(k)}\mathbf{x} + \mathbf{b}^{(k)})$$

$$\longrightarrow \mathbf{H}^{(k)} = \sigma(\mathbf{W}^{(k)}\mathbf{H}^{(k-1)}(\mathbf{A} + \mathbf{I}) + \mathbf{b}^{(k)}\mathbf{1}^T)$$

\mathbf{I} : $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix

- Proposed in [Xu et al., 2019]

$$\mathbf{h}_u^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) \mathbf{h}_u^{(k-1)} + \underbrace{\sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k-1)}}_{\mathbf{m}_{\mathcal{N}_u}^{(k-1)}} \right)$$

- Learnable parameters: MLP parameters and $\{\epsilon^{(k)}\}$
- GIN shown to be as powerful as *Weisfeiler-Lehman test* of isomorphism in distinguishing graph structures!

- Two graphs with adjacency matrices \mathbf{A}_1 and \mathbf{A}_2 , as well as node features \mathbf{X}_1 and \mathbf{X}_2 , are said to be isomorphic if and only if there exists a permutation matrix \mathbf{P} such that

$$\mathbf{P}\mathbf{A}_1\mathbf{P} = \mathbf{A}_2 \quad \text{and} \quad \mathbf{P}\mathbf{X}_1 = \mathbf{X}_2$$

- **Naive approach**: searching over the full set of permutations; complexity is $O(|\mathcal{V}|!)$
- No polynomial time is known for testing graph isomorphism
- *Weisfeiler-Lehman* approach is one of the most successful and well understood frameworks for approximate isomorphism testing.

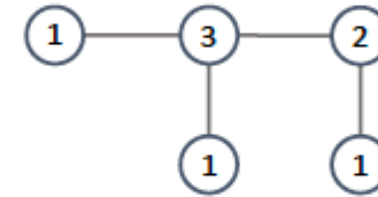
■ Algorithm 1-WL (color refinement)

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

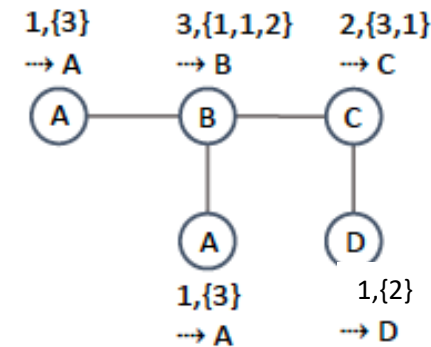
1. $c_v^0 \leftarrow |\mathcal{N}_v|, \forall v \in \mathcal{V}$
2. **repeat**
3. $c_v^\ell \leftarrow \text{hash} \left(c_v^{\ell-1}, \left\{ \{c_u^{\ell-1} : u \in \mathcal{N}_v\} \right\} \right), \forall v \in \mathcal{V}$
4. **Until** $(c_v^\ell)_{v \in \mathcal{V}} = (c_v^{\ell-1})_{v \in \mathcal{V}}$
5. **Return** $\{ \{c_v^\ell : v \in \mathcal{V}\} \}$

- The WL algorithm converges in at most $|\mathcal{V}|$ iterations
- The algorithm successfully distinguishes most pairs of graphs, but fails in some basic cases, e.g. regular graphs on n nodes and degree d

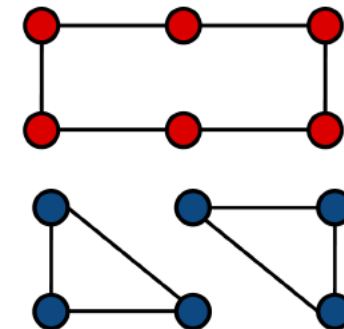
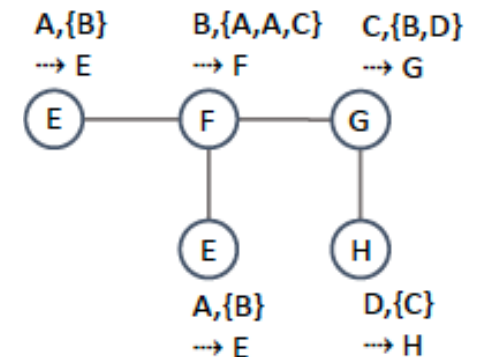
Iteration 0



Iteration 1



Iteration 2



- In both WL algorithm and neural message passing GNN, we iteratively aggregate information from local node neighborhoods and use this aggregated information to update the representation of each node
- **Key distinction:** WL algorithm aggregates and updates discrete labels (using a hash function) while GNN models aggregate and update node embeddings using neural networks
- GNNs have been motivated and derived as a continuous and differentiable analog of the WL algorithm

- **Theorem** ([Morris et al., 2019, Xu et al., 2019]). *Define a message-passing GNN (MP-GNN) to be any GNN that consists of K message-passing layers of the following form:*

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)} \left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \right), \quad (7.61)$$

where AGGREGATE is a differentiable permutation invariant function and UPDATE is a differentiable function. Further, suppose that we have only discrete feature inputs at the initial layer, i.e., $\mathbf{h}_u^{(0)} = \mathbf{x}_u \in \mathbb{Z}^d, \forall u \in \mathcal{V}$. Then we have that $\mathbf{h}_u^{(K)} \neq \mathbf{h}_v^{(K)}$ only if the nodes u and v have different labels after K iterations of the WL algorithm.

→ GNNs are no more powerful than WL algorithm when node features are discrete

Theorem ([Morris et al., 2019, Xu et al., 2019]). *There exists a MP-GNN such that $\mathbf{h}_u^{(K)} = \mathbf{h}_v^{(K)}$ if and only if the two nodes u and v have the same label after K iterations of the WL algorithm.*

→ There exist message-passing GNNs that are as powerful as the WL test

- **Question:** which GNN are as powerful as WL?

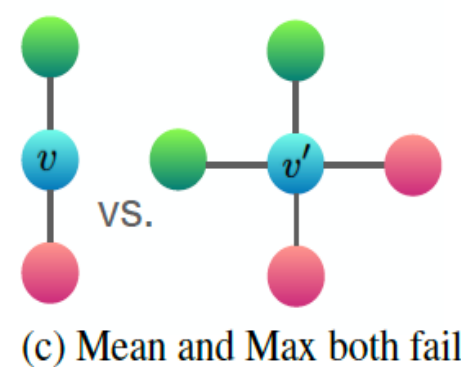
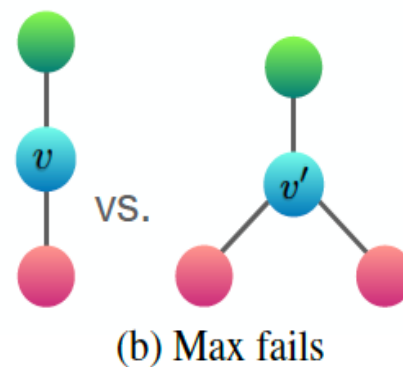
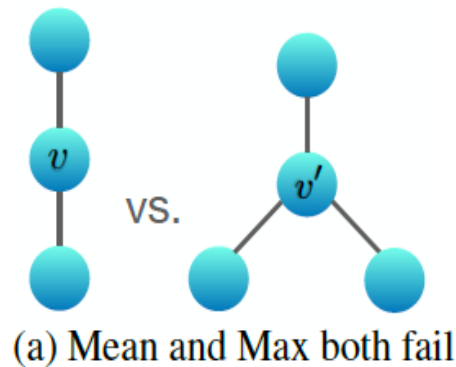
Theorem Let $\mathcal{A} : \mathcal{G} \rightarrow \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, \mathcal{A} maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:

a) \mathcal{A} aggregates and updates node features iteratively with

$$h_v^{(k)} = \phi \left(h_v^{(k-1)}, f \left(\left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right) \right),$$

where the functions f , which operates on multisets, and ϕ are injective.

b) \mathcal{A} 's graph-level readout, which operates on the multiset of node features $\{h_v^{(k)}\}$, is injective.



Examples of graph structures that mean and max aggregators fail to distinguish.

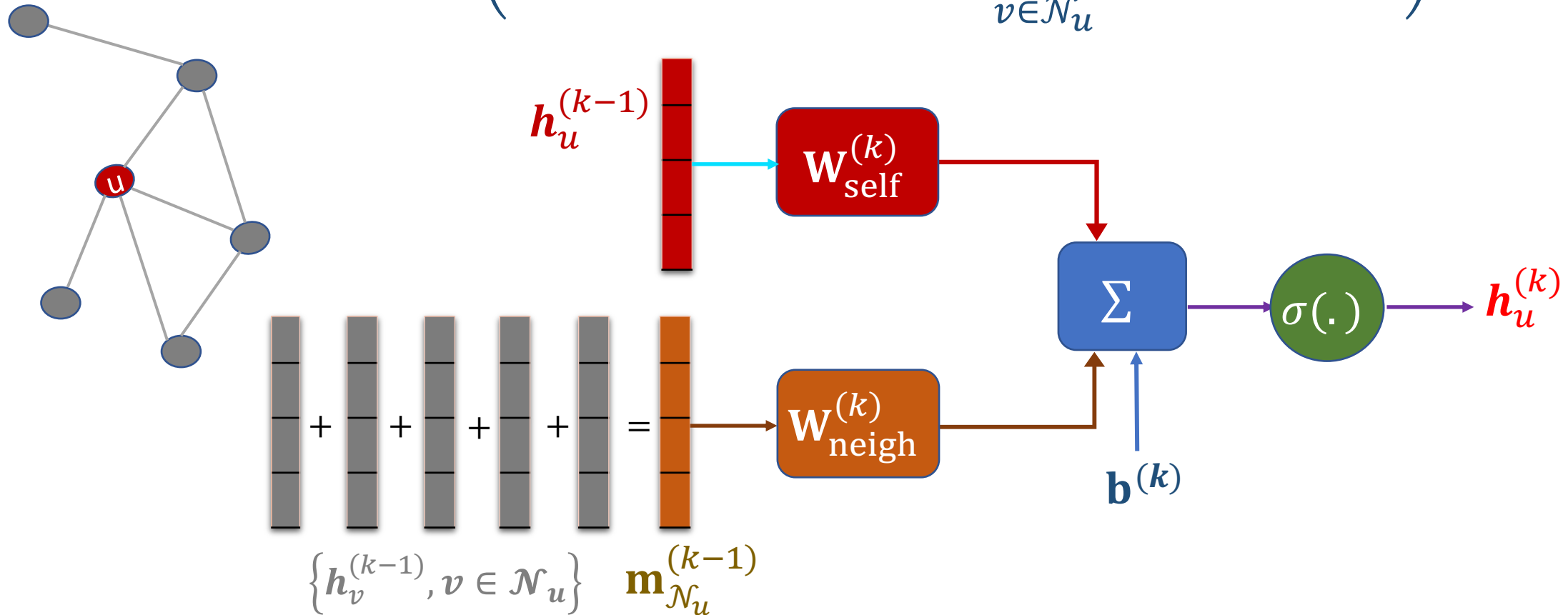
- GIN was shown to be the ‘minimal’ GNN that is as powerful as WL

$$\mathbf{h}_v^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) \mathbf{h}_u^{(k-1)} + \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k-1)} \right)$$

- Some other GNN may not be as expressive but this does not mean that they are not useful.
- For example, using the mean aggregator may be sufficient if what matters is the distribution of labels in the neighborhood...
- For node classification, the aim is to have similar embeddings for nodes belonging to the same class regardless of the structure of their subgraph

- Another MP-GNN which is as powerful as WL:

$$\mathbf{h}_u^{(k)} = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right)$$



- Message passing components (superscript omitted):

$$\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \mathbf{h}_v$$

$$\text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}_u}) = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u + \mathbf{W}_{\text{neigh}}^{(k)} \mathbf{m}_{\mathcal{N}_u} + \mathbf{b}^{(k)} \right)$$

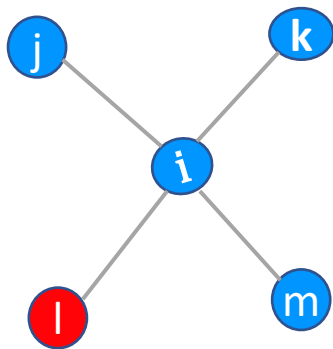
- Graph-level representation (bias omitted to simplify notation):

$$\mathbf{H}^{(k)} = \sigma \left(\mathbf{W}_{\text{self}}^{(k)} \mathbf{H}^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \mathbf{H}^{(k-1)} \mathbf{A} \right) \quad \mathbf{H}^{(k)} \in \mathbb{R}^{d_k \times |\mathcal{V}|}$$

- May be useful for heterophilic graphs...

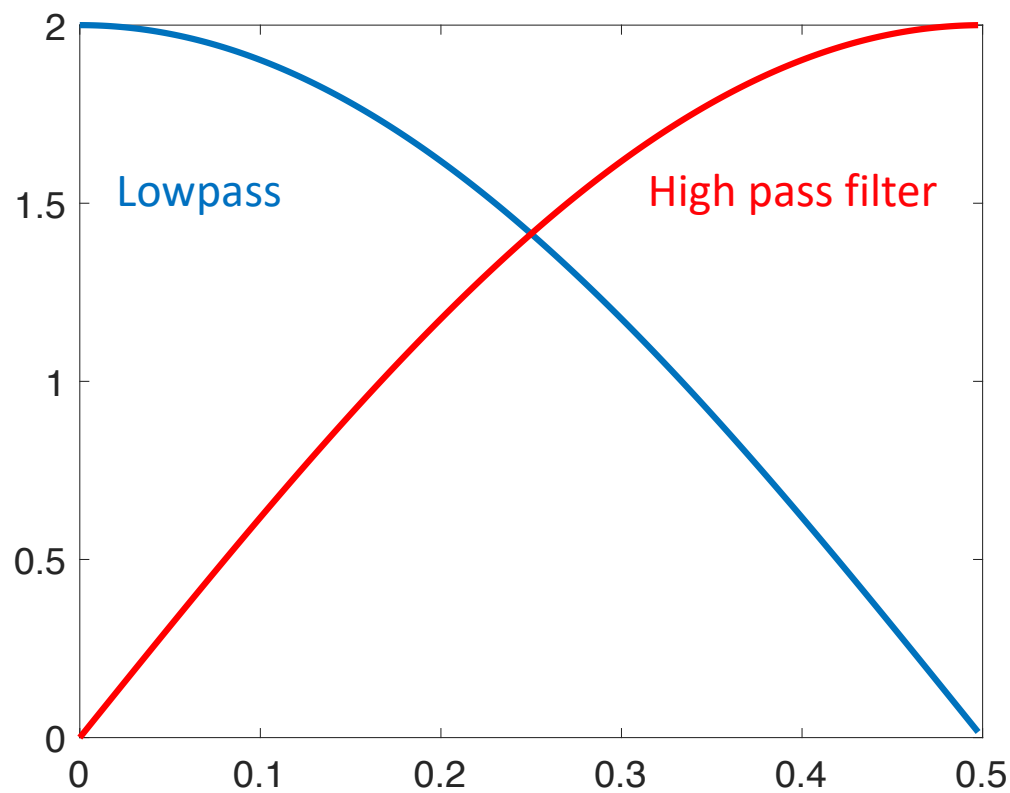
$$\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \mathbf{h}_v \quad \text{Lowpass filtering/smoothing}$$

$$\mathbf{W}_{\text{neigh}}^{(k)} = \mathbf{W}_{\text{self}}^{(k)}$$

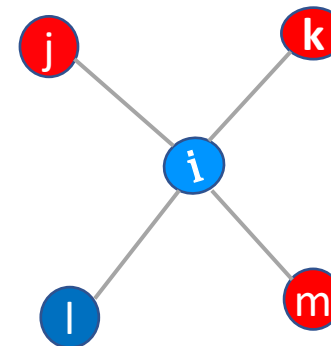


Good when $\mathbf{h}_v \approx \mathbf{h}_u$

Homophily



$$\mathbf{W}_{\text{neigh}}^{(k)} = -\mathbf{W}_{\text{self}}^{(k)}$$



Good when $\mathbf{h}_v \approx -\mathbf{h}_u$

Heterophily

- The aggregation may lead to instability and impeded learning when node degree variations are high; so it might be useful to normalize the aggregation

- Common normalizations:

$$\mathbf{m}_{\mathcal{N}_u} = \frac{\sum_{v \in \mathcal{N}_u} \mathbf{h}_v}{|\mathcal{N}_u|}$$

$$\mathbf{m}_{\mathcal{N}_u} = \sum_{v \in \mathcal{N}_u} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}_u| |\mathcal{N}_v|}}$$

Symmetric normalization [Kipf and Welling, 2016]

- However, normalization may lead to a loss of information: it can obscure structural features (e.g. node degree); may make aggregation non-injective
- Normalize or not normalize thus depends on the graph being investigated and the ML task

- CGN employs self-loop and symmetric normalization (popular baseline) (not as powerful as WL !) [Kipf and Welling, 2016]:

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

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

- CGN is a first-order approximation of a spectral graph convolution.

- CGN employs

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

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

- Neighbors of lower degrees than node u have relative weights larger than one!
- This is problematic for node classification, particularly when node features are more important than graph structure for node classification

- The way the aggregate and update functions should be designed has been a very active area of research.
- Universal set function approximator [Zaheer et al., 2017]

$$\mathbf{h}_u^{(k+1)} = \phi \left(\mathbf{h}_u^{(k)}, \sum_{v \in \mathcal{N}_u} \psi(\mathbf{h}_u^{(k)}, \mathbf{h}_v^{(k)}) \right)$$

- ϕ and ψ are arbitrarily deep multi-layer perceptrons (though typically a single layer NN)

- **Janossy pooling** [Murphy et al., 2018]: instead of using a permutation invariant reduction, apply a permutation-sensitive function (g_{ϕ}) and average the result over many possible permutations

$$\mathbf{m}_{\mathcal{N}(u)} = \text{MLP}_{\theta} \left(\frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} g_{\phi} \left(\left(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \dots, \mathbf{h}_{v_{|\mathcal{N}(u)|}} \right)_p \right) \right)$$

- In practice, g_{ϕ} is often an LSTM NN, and a random subset of permutations, or a canonical ordering of the nodes (e.g. node degree-based) is used.

- **Idea**: assign an attention weight/importance to each neighbor, before the aggregation step. The attention weights depend on the node features

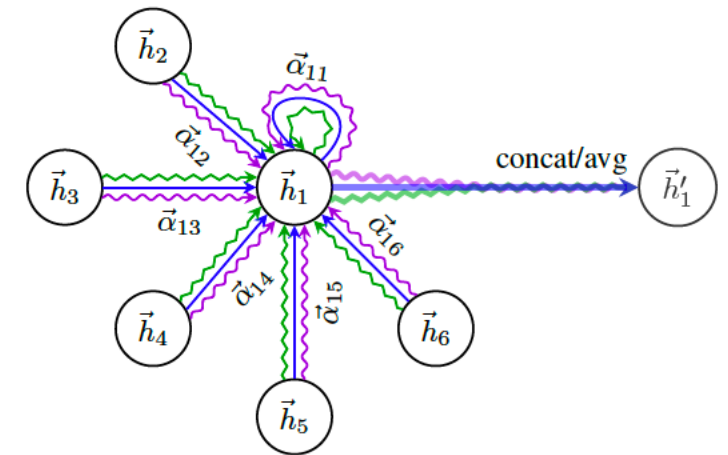
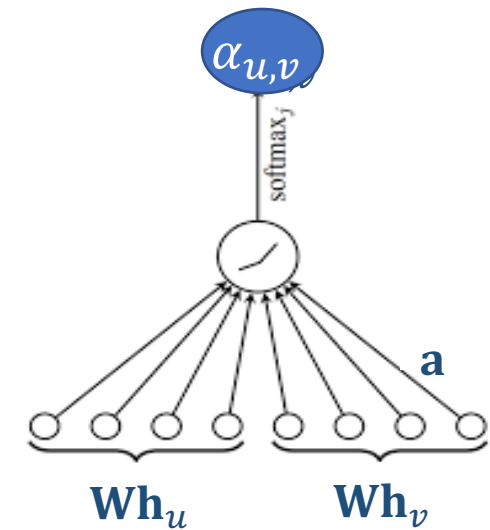
$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}_u} \alpha_{u,v} \mathbf{h}_v$$

- **Graph Attention Network (GAT)** [Veličković et al. 2018]:

$$\alpha_{u,v} = \frac{\exp(g(\mathbf{a}^T [\mathbf{W}\mathbf{h}_u \parallel \mathbf{W}\mathbf{h}_v]))}{\sum_{v' \in \mathcal{N}_u \cup \{u\}} \exp(g(\mathbf{a}^T [\mathbf{W}\mathbf{h}_u \parallel \mathbf{W}\mathbf{h}_{v'}]))}$$

$$\mathbf{h}_u^{(k)} = \sigma \left(\sum_{v \in \mathcal{N}_u \cup \{u\}} \alpha_{u,v}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right)$$

- **a**: trainable attention vector; **W**: trainable matrix;



- Assuming one layer:

$$\mathbf{z}_u = \sigma \left(\frac{1}{\sum_v \varphi(\tilde{\mathbf{h}}_u, \tilde{\mathbf{h}}_v)} \sum_{v \in \mathcal{N}_u \cup \{u\}} \varphi(\tilde{\mathbf{h}}_u, \tilde{\mathbf{h}}_v) \tilde{\mathbf{h}}_v + \mathbf{b} \right) \quad \tilde{\mathbf{h}}_v = \mathbf{W} \mathbf{h}_v$$

$$\varphi(\tilde{\mathbf{h}}_u, \tilde{\mathbf{h}}_v) = \exp(g(\mathbf{a}^T [\tilde{\mathbf{h}}_u \parallel \tilde{\mathbf{h}}_v]))$$

- GAT can be interpreted as an instantiation of **nonlinear aggregation**, where the **attention is additive**, i.e. if one feature

$$\varphi(\tilde{h}_u, \tilde{h}_v) = \exp(g(a_1 \tilde{h}_u + a_2 \tilde{h}_v))$$

- Assuming one layer and one feature:

$$\sum_{v \in \mathcal{N}_u \cup \{u\}} \varphi(\tilde{h}_u, \tilde{h}_v) \tilde{h}_v = \varphi(\tilde{h}_u, \tilde{h}_u) \left(\tilde{h}_u + \sum_{v \in \mathcal{N}_u \cup \{u\}} \lambda_{u,v} \tilde{h}_v \right)$$

$$\lambda_{u,v} = \frac{\varphi(\tilde{h}_u, \tilde{h}_v)}{\varphi(\tilde{h}_u, \tilde{h}_u)} = \frac{\exp(g(a_1 \tilde{h}_u + a_2 \tilde{h}_v))}{\exp(g(a_1 \tilde{h}_u + a_2 \tilde{h}_u))}$$

Relative weight
assigned to neighbor v

- The closer \tilde{h}_v is to \tilde{h}_u , the closer $\lambda_{u,v}$ to one, which is good
- But $\lambda_{u,v}$ is larger than one when $a_2 \tilde{h}_v > a_2 \tilde{h}_u$
- Other attention mechanisms may be better, e.g. similarity based attention (assumes homophily):

$$\varphi(\tilde{h}_u, \tilde{h}_v) = \beta \exp(\text{sim}(\tilde{h}_u, \tilde{h}_v) / \tau)$$

- **Over-smoothing**: more message passing iterations makes node representations more alike. This hurts the building of deeper GNN models (needed for longer-term dependencies in the graph)
- Increasing K is similar to increasing the receptive field of kernels in convolutional neural networks (CNN)
- The influence of node u on node v in the GNN can be quantified using the magnitude of the corresponding Jacobian matrix [Xu et al., 2018]:

$$I_K(u, v) = \mathbf{1}^T \left(\frac{\partial \mathbf{h}_v^{(K)}}{\partial \mathbf{h}_u^{(0)}} \right) \mathbf{1} \quad \mathbf{h}_u^{(0)} = \mathbf{x}_u \text{ (input feature vector)}$$

- [Xu et al., 2018]

Theorem *For any GNN model using a self-loop update approach and an aggregation function of the form*

$$\text{AGGREGATE}(\{\mathbf{h}_v, \forall v \in \mathcal{N}(u) \cup \{u\}\}) = \frac{1}{f_n(|\mathcal{N}(u) \cup \{u\}|)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \mathbf{h}_v,$$

where $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is an arbitrary differentiable normalization function, we have that

$$I_K(u, v) \propto p_{\mathcal{G}, K}(u|v),$$

where $p_{\mathcal{G}, K}(u|v)$ denotes the probability of visiting node v on a length- K random walk starting from node u .

[Hamilton et al. 2020]

As K increases, $p_{\mathcal{G}, K}(u|v)$ gets closer to uniform distribution, i.e. local neighborhood information is lost

- Here, the output of the aggregation function is viewed as an observation to update the hidden state of each node
- **Concatenation-base skip connection** [Hamilton et al., 2017a]: concatenate the output of a base update function $\text{UPDATE}_{\text{base}}$ with node's previous representation

$$\text{UPDATE}_{\text{concat}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = [\text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) \parallel \mathbf{h}_u]$$

- **Linear interpolation-skip connections** [Pham et al. 2017]

$$\text{UPDATE}_{\text{concat}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \lambda \circ \text{UPDATE}_{\text{base}}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) + (1 - \lambda) \circ \mathbf{h}_u$$

$$\lambda = [0,1]^d$$

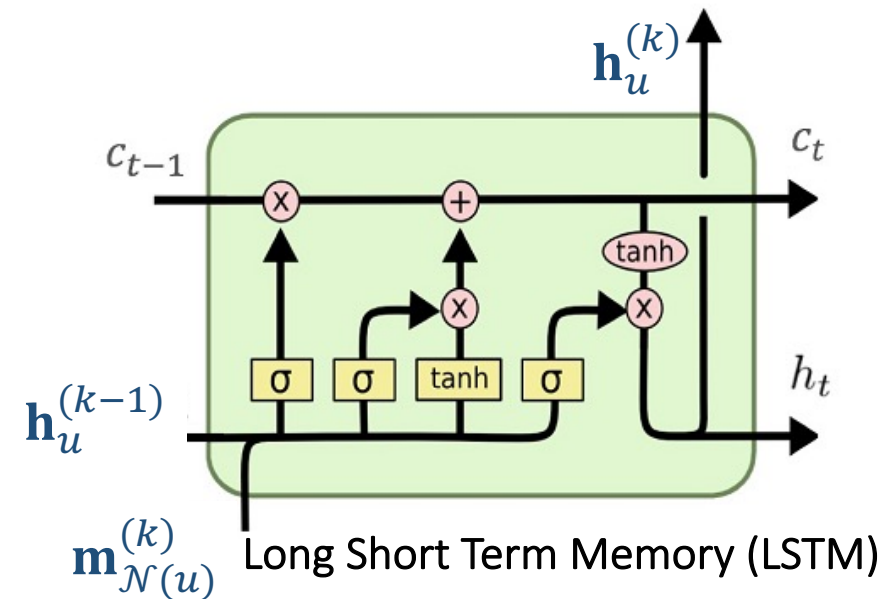
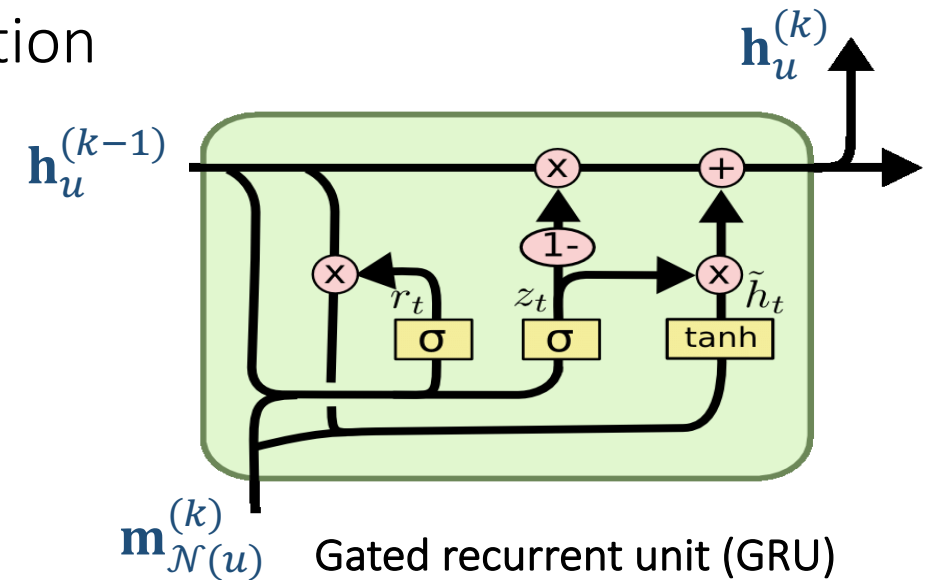
- The output of the aggregation is viewed as observation used to update the hidden state as in RNN
- Update with GRU [Li et al., 2015]

$$\mathbf{h}_u^{(k)} = \text{GRU} \left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)} \right)$$

- Update with LSTM [Selsam et al., 2019]

Effective at facilitating deep GNN architectures (e.g., more than 10 layers) while preventing over-smoothing

Useful when prediction requires complex reasoning over the global graph, e.g. program analysis



- Instead of using $\mathbf{z}_u = \mathbf{h}_u^{(K)}$, i.e. embedding= output of the final layer of GNN, [Xu et al., 2018] proposed Jumping Knowledge (JK) connections

$$\mathbf{z}_u = f_{\text{JK}} \left(\mathbf{h}_u^{(0)} \parallel \mathbf{h}_u^{(1)} \parallel \cdots \parallel \mathbf{h}_u^{(K)} \right)$$

- f_{JK} may be the identify function, max-pooling, LSTM...

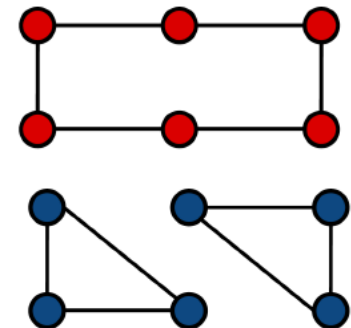
- Message-passing GNNs (MP-GNNs) are no more powerful than the WL algorithm
- Designing GNN that are more powerful than WL is an active area of research
- Some attempts:
 - Relational pooling [Murphy et al., 2019]
 - K-WL [Maron et al., 2019]

- Message-passing GNNs (MP-GNNs) are no more powerful than the WL algorithm
- Designing GNN that are more powerful than WL is an active area of research
- Relational pooling [Murphy et al., 2019]:
 - **Idea**: augmenting MP-GNNs with unique node ID features

$$\mathbf{Z} = \text{MP-GNN}(\mathbf{A}, \mathbf{X} \oplus \mathbf{I})$$

\mathbf{I} : $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix; \oplus : concatenation horizontale

But this breaks the permutation equivariance!



- **Solution**:

$$\text{RP-GNN}(\mathbf{A}, \mathbf{X}) = \sum_{\mathbf{P} \in \mathcal{P}} \text{MP-GNN}(\mathbf{PAP}^T, (\mathbf{PX}) \oplus \mathbf{I})$$

- **Intuition**: interleaving trainable neural networks with graph convolution layers may not be necessary. So, simply use ANN to learn feature transformations at the beginning and end, and apply a deterministic convolution layer to leverage the graph structure,

$$\mathbf{Z} = \text{MLP}_{\theta}(f(\mathbf{A})\text{MLP}_{\phi}(\mathbf{X}))$$

- Example 1 [Wu et al. 2019]:

$$f(\mathbf{A}) = \tilde{\mathbf{A}}^k$$
$$\tilde{\mathbf{A}} = (\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-\frac{1}{2}}$$

- Example 2 [Klicpera et al. 2019] (by analogy to PageRank algorithm):

$$f(\mathbf{A}) = \alpha(\mathbf{I} - (1 - \alpha)\tilde{\mathbf{A}})^{-1} = \alpha \sum_{k=0}^{\infty} (\mathbf{I} - \alpha\tilde{\mathbf{A}})^k$$

- **Motivation**: connections to variational inference in probabilistic graphical models
- View the embeddings \mathbf{z}_u as latent variables to infer from \mathbf{A} and \mathbf{X}
- MP in GNNs can then be viewed as a neural network analogue of certain MP algorithms that are used for variational inference to infer distributions over latent variables [Dai et al., 2016] [Hamilton, 2020].
- Markov random field model:

$$p(\{\mathbf{x}_v\}, \{\mathbf{z}_v\}) \propto \prod_{v \in V} \Phi(\mathbf{x}_v, \mathbf{z}_v) \prod_{(u,v) \in \mathcal{E}} \Psi(\mathbf{z}_u, \mathbf{z}_v)$$

Model dependency between nodes



- Our goal is to infer $p(\mathbf{z}_u)$ for all nodes, while also implicitly learning Φ and Ψ

- **Approximation:** employ mean-field variational inference, i.e.

$$p(\{\mathbf{z}_v\}|\{\mathbf{x}_v\}) \approx q(\{\mathbf{z}_v\}) = \prod_{v \in \mathcal{V}} q_v(\mathbf{z}_v)$$

- Standard approach: minimize the Kullback-Leibler (KL) divergence between the approximate posterior and the true posterior

$$\text{KL}(q(\{\mathbf{z}_v\})|p(\{\mathbf{z}_v\}|\{\mathbf{x}_v\})) = \int_{(\mathbb{R}^d)^{\mathcal{V}}} \prod_{v \in \mathcal{V}} q(\{\mathbf{z}_v\}) \log \left(\frac{\prod_{v \in \mathcal{V}} q(\{\mathbf{z}_v\})}{p(\{\mathbf{z}_v\}|\{\mathbf{x}_v\})} \right) \prod_{v \in \mathcal{V}} d\mathbf{z}_v.$$

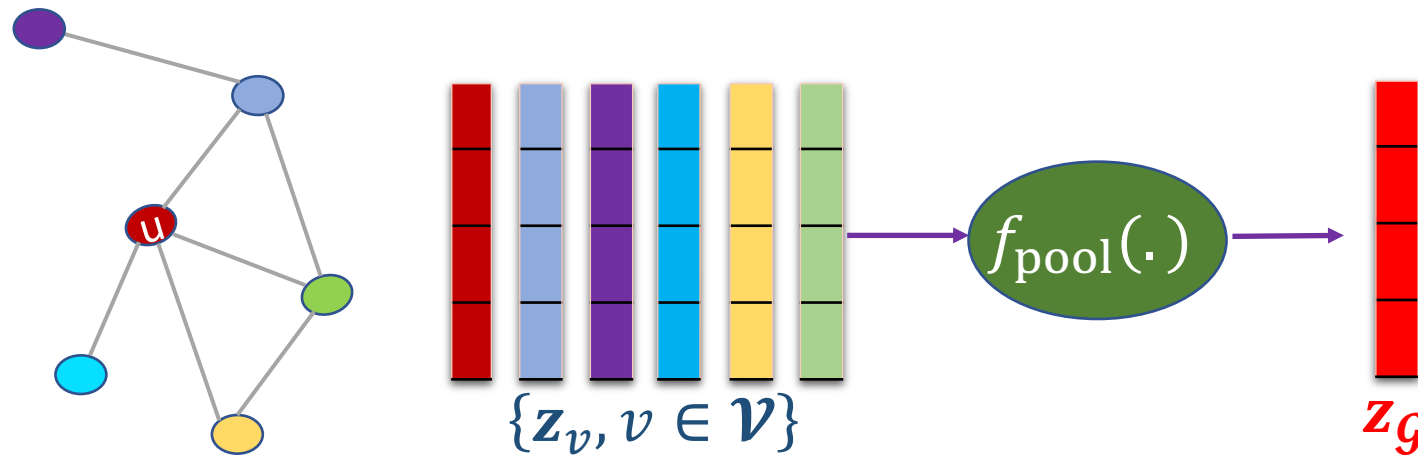
- Solve fixed-point equation:

$$\log(q(\mathbf{z}_v)) = c_v + \log(\Phi(\mathbf{x}_v, \mathbf{z}_v)) + \sum_{u \in \mathcal{N}(v)} \int_{\mathbb{R}^d} q_u(\mathbf{z}_u) \log(\Psi(\mathbf{z}_u, \mathbf{z}_v)) d\mathbf{z}_u,$$

- Key distinction: mean-field MP equations operate over distributions rather than embeddings as in standard GNN MP

- **Goal:** learn an embedding $\mathbf{z}_{\mathcal{G}}$ for the entire graph \mathcal{G}
- Downstream tasks: graph classification, isomorphism test, etc.
- **Set pooling:**

$$\mathbf{z}_{\mathcal{G}} = f_{\text{pool}}(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{|\mathcal{V}|})$$



■ Set pooling:

• Approach 1:

$$\mathbf{z}_G = \frac{\sum_{v \in \mathcal{V}} \mathbf{z}_v}{f_n(|\mathcal{V}|)} \quad f_n \text{ is a normalization function}$$

- Approach 2: combine LSTMs and attention mechanism; iterate a series of attention based aggregations ($i = 1, \dots, T$) [Vinyals et al., 2015]

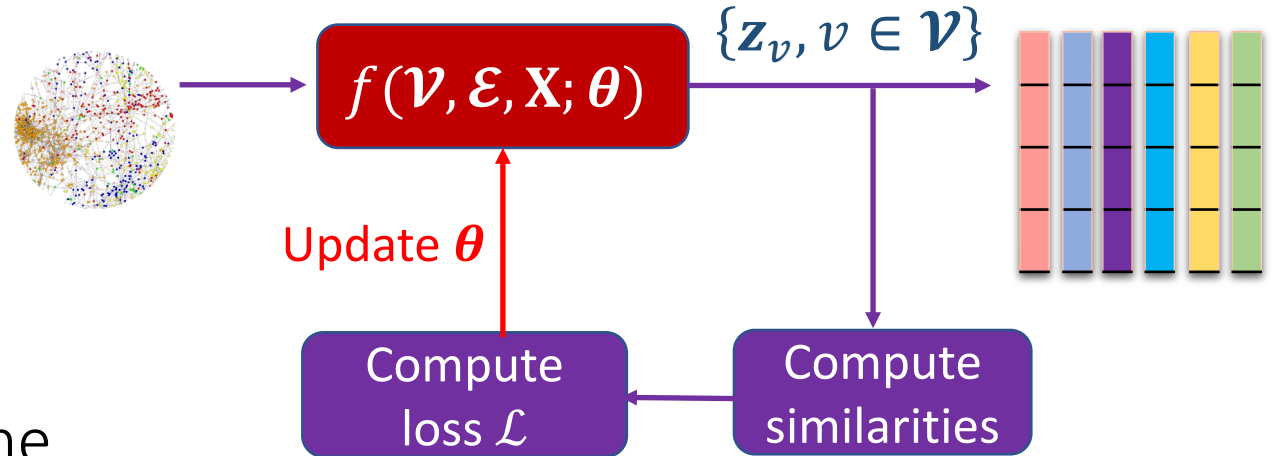
$$\mathbf{q}_i = \text{LSTM}(\mathbf{o}_{i-1}, \mathbf{q}_{i-1})$$

$$a_{v,i} = \frac{\exp(\mathbf{z}_v^T \mathbf{q}_i)}{\sum_{u \in \mathcal{V}} \exp(\mathbf{z}_u^T \mathbf{q}_i)}$$

$$\mathbf{o}_i = \sum_{v \in \mathcal{V}} a_{v,i} \mathbf{z}_v \quad \longrightarrow \quad \mathbf{z}_G = \mathbf{o}_1 \parallel \mathbf{o}_2 \parallel \dots \parallel \mathbf{o}_T$$

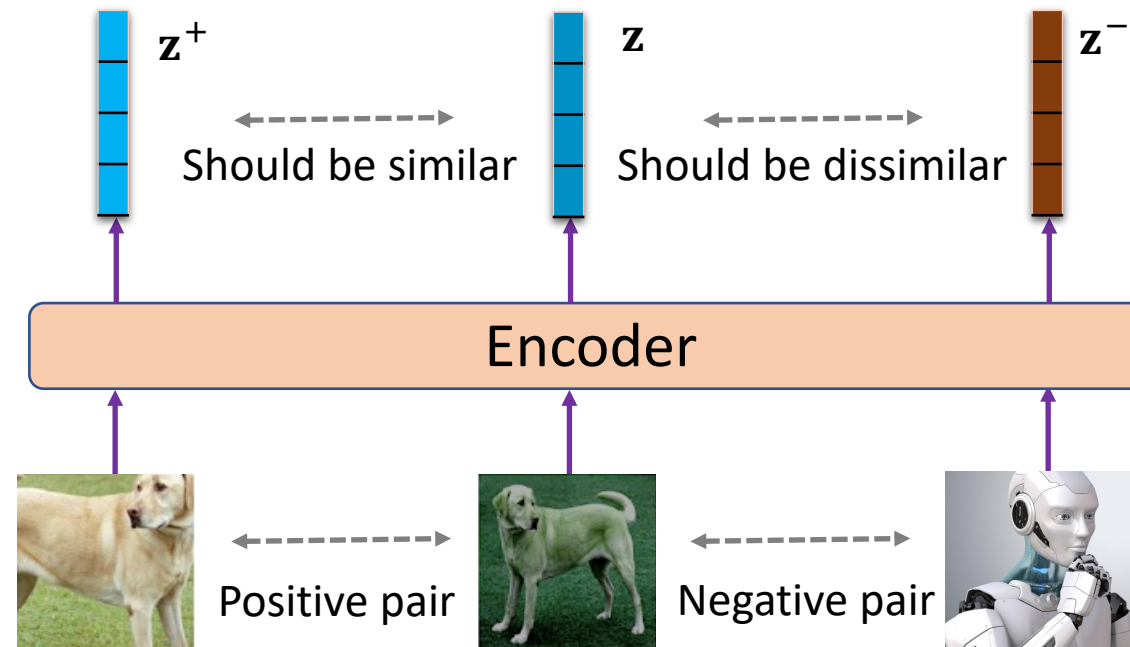
Initialization: $\mathbf{o}_0 = \mathbf{q}_0 = \mathbf{0}$

- **Goal:** Optimize the GNN parameters in an unsupervised manner to generate node embeddings independently of the downstream task
- Methods are generally based on:
 - Reconstruction objective
 - Similarity metric
 - Negative sampling
- Methods differ in the way the above components are designed...

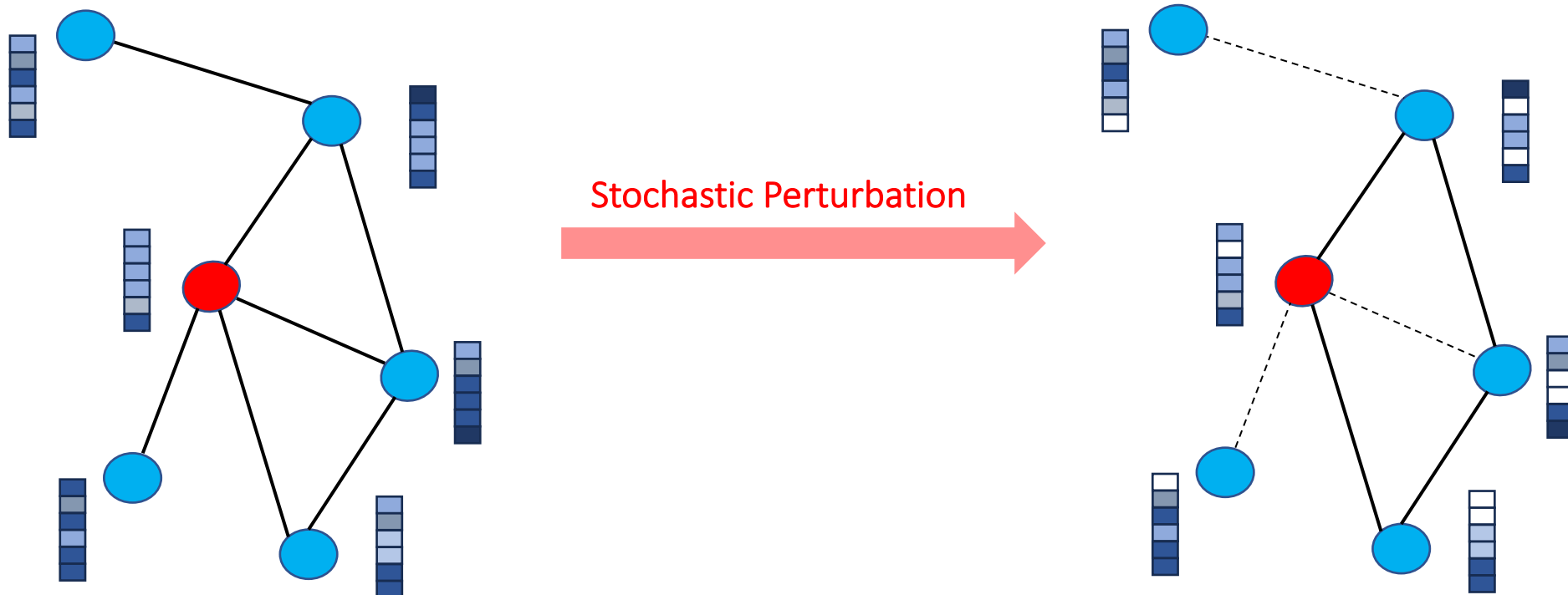


Loss \mathcal{L} is generally computed using negative sampling

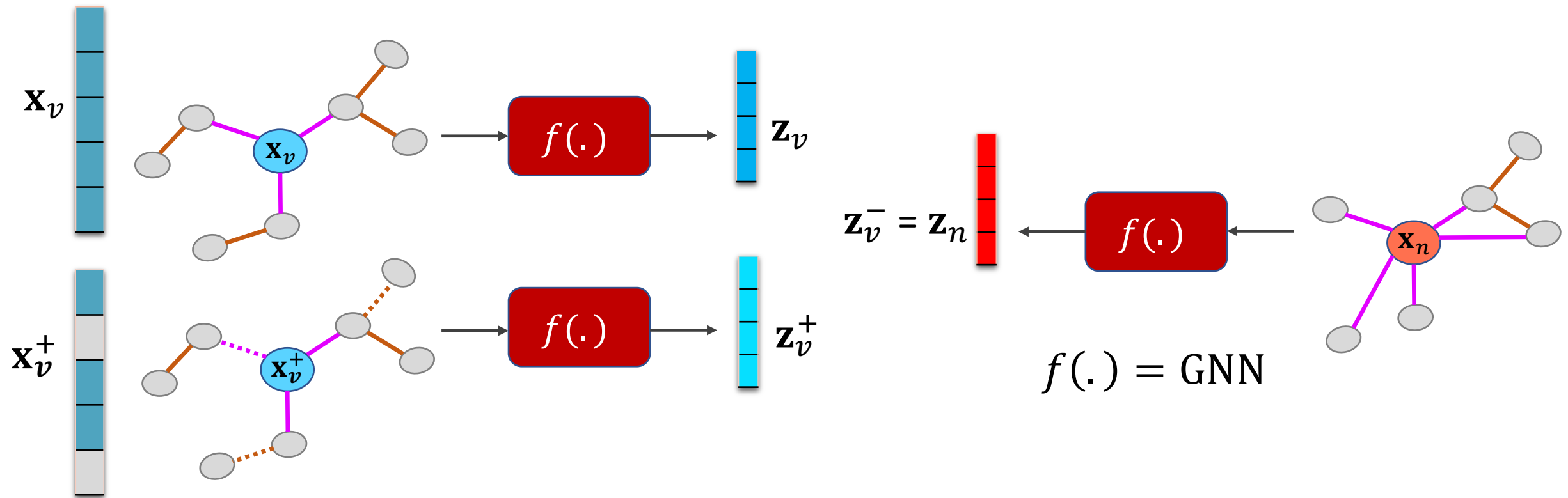
- We have extended the contrastive learning method to graphs [Hafidi et al., 2021a]
- **Reminder:** the goal is to bring representations of the same object under different views (resp. different objects) close to (resp. far from) each other in the embedding space [Chen et al., 2020]



- **Aim**: learn node embeddings by maximizing similarity between representations of **perturbed versions of the same node's** local subgraph.
- Perturbation involves **randomly dropping** a subset of edges and nodes' intrinsic features from the node's L-hop subgraph.



- To avoid the representation collapse, negative sampling is required



Maximize similarity between \mathbf{z}_v and \mathbf{z}_v^+ and minimizing similarity between \mathbf{z}_v and \mathbf{z}_v^-

- Individual contrastive loss

$$\mathcal{L}(\mathbf{z}_v, \mathbf{z}_v^+, Q_v^-) = -\log \frac{\exp(\mathbf{z}_v^\top \mathbf{z}_v^+ / \tau)}{\exp(\mathbf{z}_v^\top \mathbf{z}_v^+ / \tau) + \sum_{\mathbf{z}_n \in Q_v^-} \exp(\mathbf{z}_v^\top \mathbf{z}_n / \tau)}$$

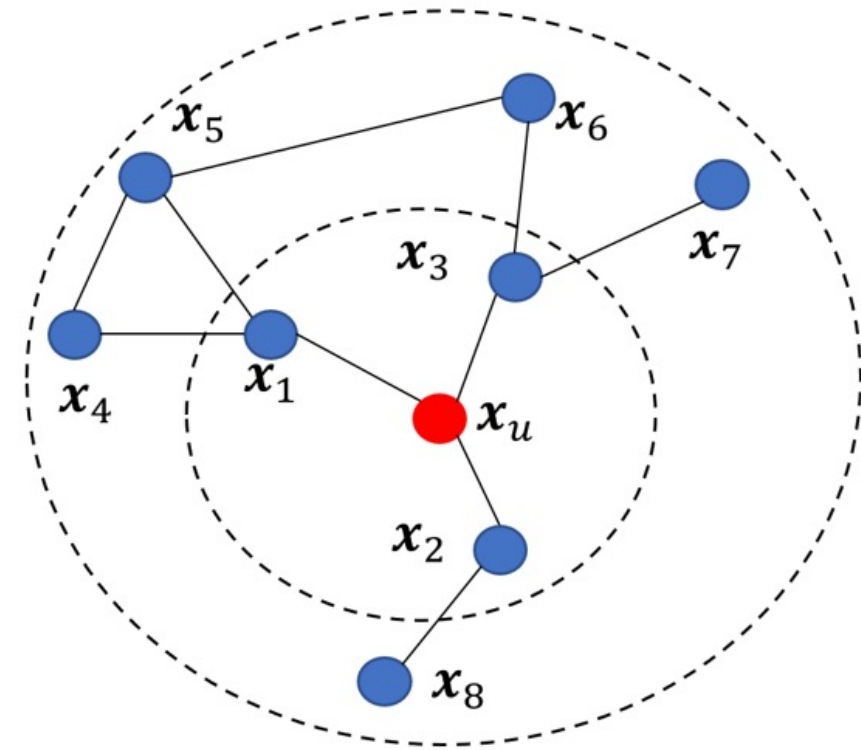
- Q_v^- : a set of negative examples for node v

- Total contrastive loss:

$$\mathcal{L} = -\frac{1}{\tau|\mathcal{D}|} \sum_{v \in \mathcal{D}} \mathbf{z}_v^\top \mathbf{z}_v^+ + \frac{1}{|\mathcal{D}|} \log \sum_{v \in \mathcal{D}} \left(\exp(\mathbf{z}_v^\top \mathbf{z}_v^+ / \tau) + \sum_{\mathbf{z}_n \in Q_v^-} \exp(\mathbf{z}_v^\top \mathbf{z}_n / \tau) \right)$$

- Different negative sampling strategies may be used for constructing Q_v^- : random sampling, feature-based sampling, and graph-based sampling [Hafidi et al., 2021b]

- Leverages graph structure information to select negatives.
- Avoids computing similarities between each pair of nodes in the graph.
- Uses the distance between nodes on the graph to sample negatives.
- For each node v , sample negatives from its ℓ -th order neighbors.



[Hafidi et al., 2021b]

- Optimize the GNN parameters using the available labels and a specific downstream task or multiple downstream tasks (multitask learning)

See next section (node & graph classification)

Outline

Introduction

- Attributed graphs
- ML tasks on graphs
- Very brief introduction to ML

Graph embedding

- Direct encoding
- Graph neural networks (GNN)

Node & graph Classification

- Homophily
- Collective classification
- GNN-based classifiers
- Graph-assisted Bayesian classifiers

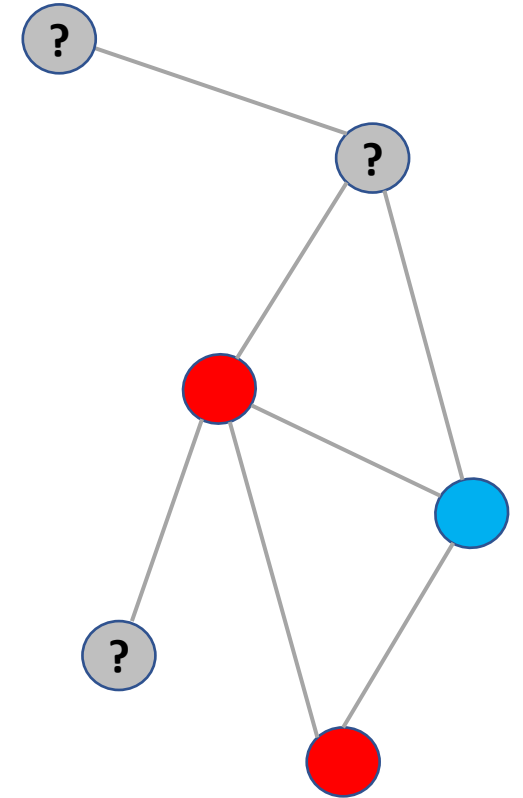
Very brief introduction to Knowledge graphs

- KG embedding
- KG completion

- Each node belongs to one of C classes:

$$y_v \in \{1, \dots, C\}$$

- Some nodes are labeled (\mathcal{V}^L) and others not (\mathcal{V}^U),
i.e. $\{y_v, v \in \mathcal{V}^L\}$ are known, $\{y_v, v \in \mathcal{V}^U\}$ are unknown
- Generally $|\mathcal{V}^L| \ll |\mathcal{V}^U|$
- Goal**: predict the labels of \mathcal{V}^U
- Example**: in a network, some nodes are known to be fraudsters and some others are known to be trusted, how to label the other nodes



- Ignoring structural information of \mathcal{G} :

$$\hat{y}_v = \arg \max_k p_k(\mathbf{x}_v); \quad p_k(\mathbf{x}_v) = \Pr(y_v = k | \mathbf{x}_v)$$

- Discriminative (parametric) methods:

$$p_k(\mathbf{x}_v) = f_{\boldsymbol{\theta}_k}(\mathbf{x}_v)$$

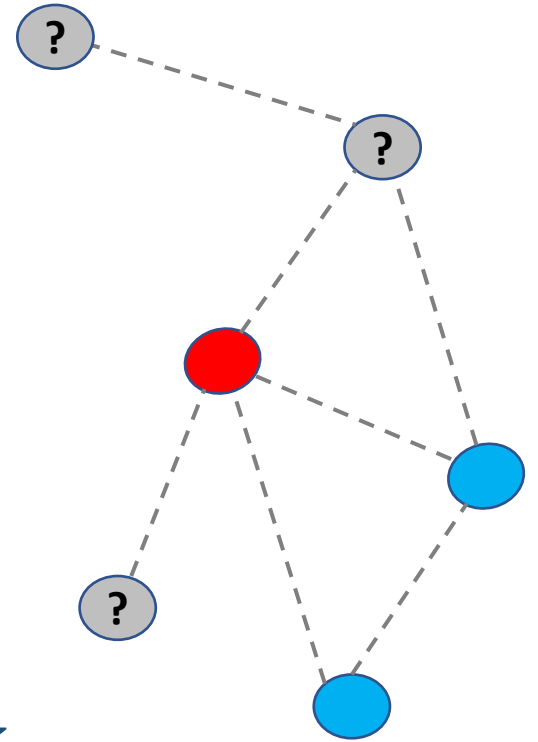
- Generative methods:

$$p_k(\mathbf{x}_v) \propto p(\mathbf{x}_v | y_v = k) \Pr(y_v = k) = D(\mathbf{x}_v; \boldsymbol{\theta}_k) \pi_k$$

$D(\mathbf{x}_v; \boldsymbol{\theta}_k)$: assumed distribution for \mathbf{x}_v , parameterized by $\boldsymbol{\theta}_k$

- Learning approaches:

- Supervised: $\{\boldsymbol{\theta}_k\}$ estimated using $\{(\mathbf{x}_v, y_v), u \in \mathcal{V}^L\}$
- Semi-supervised: $\{\boldsymbol{\theta}_k\}$ estimated using $\{(\mathbf{x}_v, y_v), v \in \mathcal{V}^L\}$ and $\{\mathbf{x}_v, v \in \mathcal{V}^U\}$



- The graph-agnostic classifier leverages only the correlations between the label of v and the observed attributes of v . The instances are considered i.i.d.
- The graph introduces correlations between node labels, which may be leveraged to improve node classification
- Indeed, in real-world graphs, similar nodes tend to connect...
- **Guilt-by-association**: if node v is connected to a node of label k , then node v is likely to have label k as well
- Key concept is **collective classification**: leveraging correlations to classify nodes together

- « Birds of a feather flock together » « Those who resemble each other assemble together » « People sharing the same tastes come together » « Similarity begets friendship [Plato] » « People love those who are like themselves [Aristotle] ».
- Homophily shown to play a key role in structuring social networks [Talaga et al., 2020]:
 - Value/choice homophily (similarity due entirely to 'conscious' preferences)
 - Induced homophily (similarity follows from structural constraints e.g. geographical distance, racial and ethnic segregation)
 - Similarity may also result from social influence. So similarity breeds connection and connection breeds similarity



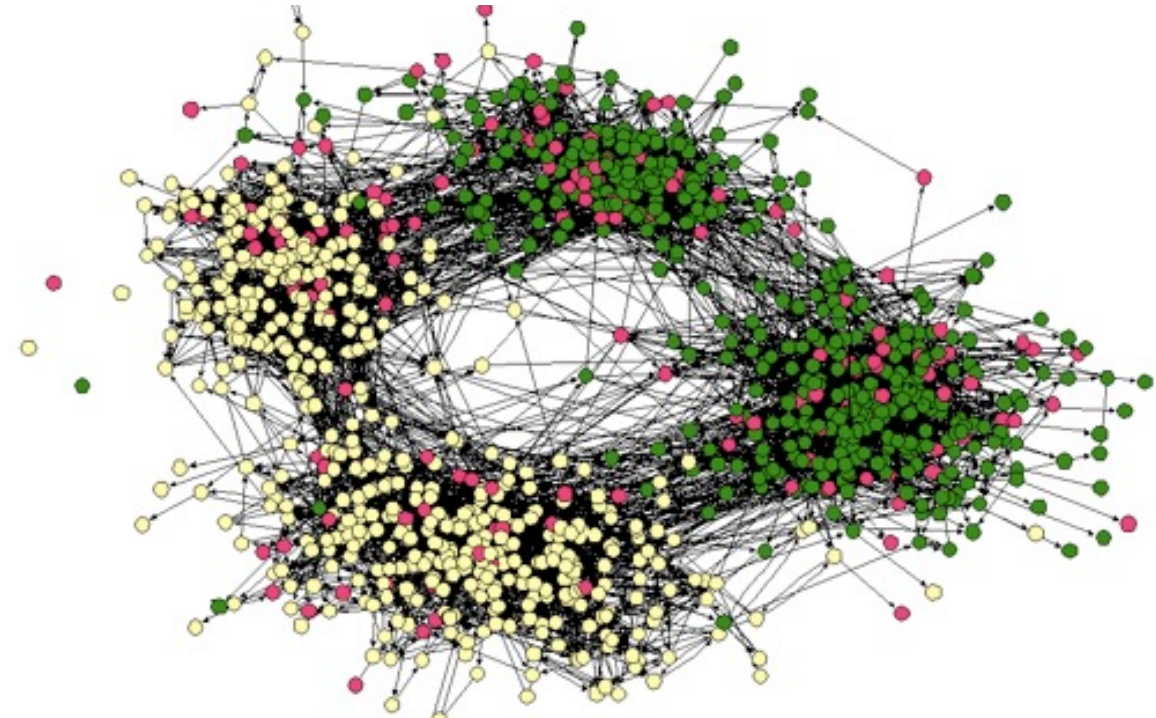
Homophily

Examples

- Function prediction of unknown proteins from the PPI networks

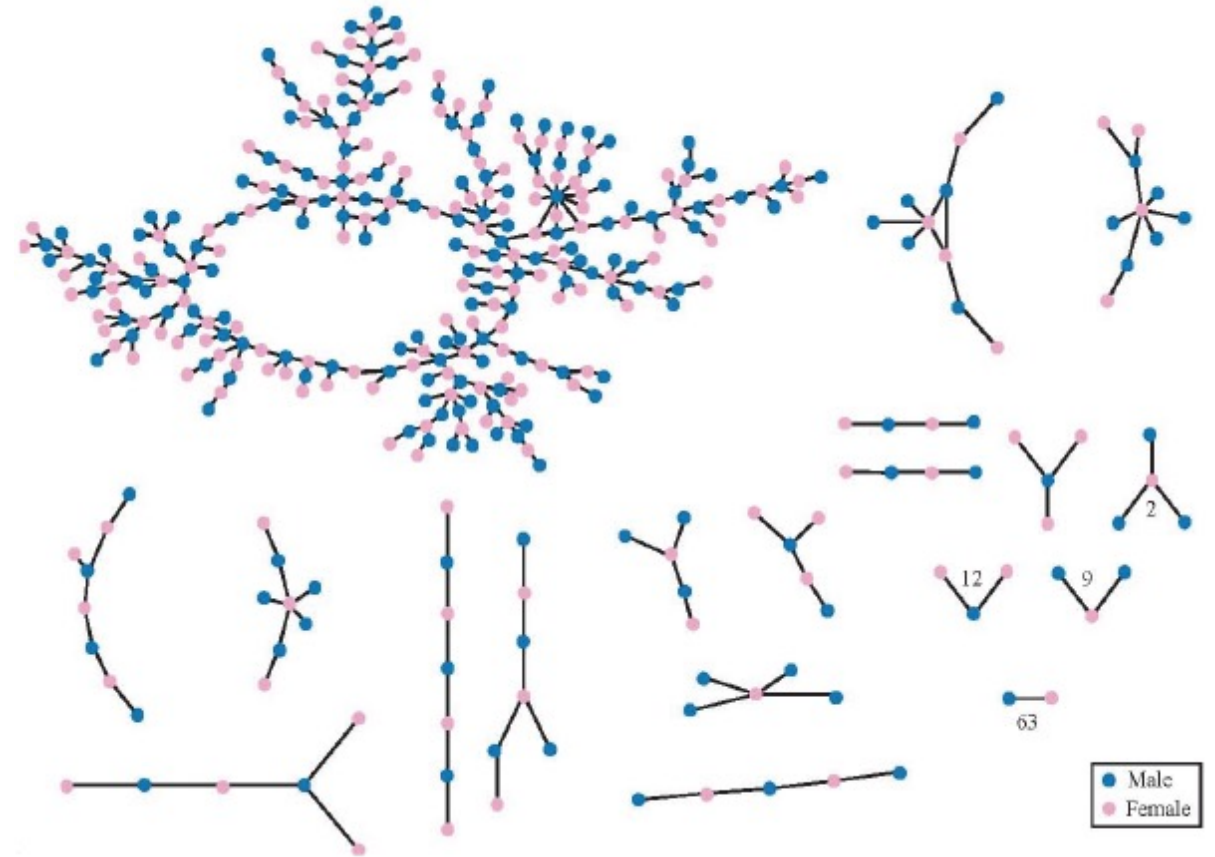
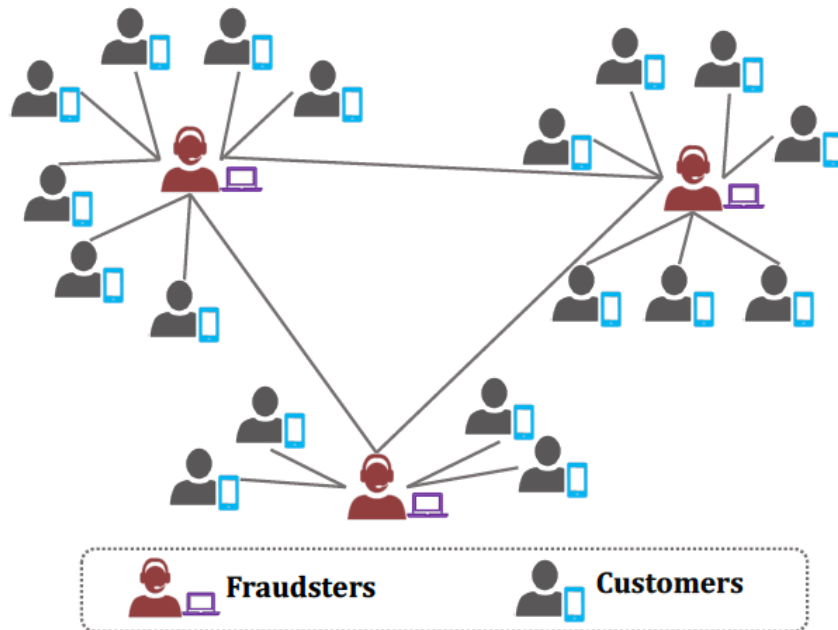


Highschool network colored by ethnic groups [Easley & Kleinberg, 2010]



Color: ethnic group
Middle and high school clusters

- Opposite of homophily: dissimilar nodes tend to be connected
- Examples
 - Sentimental networks
 - Fraudsters-customers networks



- **Definition 1.** Average homophily ratio: fraction of edges that connect nodes belonging to the same class, i.e.

$$p_h = \frac{1}{|\mathcal{E}|} \sum_{(u,v) \in \mathcal{E}} \mathbb{I}(y_u = y_v)$$

- **Definition 2.** Per-node homophily ratio: fraction of node u 's neighbors belonging to the same class as node u , i.e.

$$p_{h,u} = \frac{1}{|\mathcal{N}_u|} \sum_{v \in \mathcal{N}_u} \mathbb{I}(y_v = y_u)$$

- **Definition 3.** Per-node-degree homophily ratio: average of the per node homophily ratios of all nodes with degree d , i.e.

$$p_h(d) = \frac{1}{|\mathcal{V}_d|} \sum_{v_i \in \mathcal{V}_d} p_{h,i} \quad \mathcal{V}_d: \{v_i \in \mathcal{V} \text{ s.t. } |\mathcal{N}_i| = d\}$$

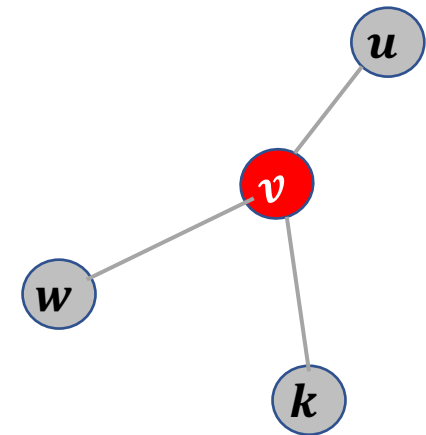
- Markov model: the label y_v of one node v depends on the labels of its neighbors \mathcal{N}_v , i.e.

$$p(y_v) = p(y_v \mid \{y_u, u \in \mathcal{N}_v\})$$

- Let $t_{k,\ell}$ denote the probability that a randomly chosen neighbor (u) of a randomly chosen class k node (v) is of class ℓ , $t_{k,\ell} \approx \Pr[y_u = \ell \mid y_v = k]$ (assumed constant)
- $t_{k,\ell}$ can be estimated as

$$t_{k,\ell} = \frac{1}{|\tilde{\mathcal{V}}_k|} \sum_{v \in \tilde{\mathcal{V}}_k} \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} \mathbb{I}(y_u = \ell)$$

$$\tilde{\mathcal{V}}_k = \{v; y_v = k \text{ and } \mathcal{N}_v \neq \emptyset\}$$



- Class transition matrix ($t_{k,\ell} = \Pr[y_u = \ell | y_v = k]$):

$$\mathbf{T} = \begin{pmatrix} t_{1,1} & \cdots & t_{1,c} \\ \vdots & \ddots & \vdots \\ t_{c,1} & \cdots & t_{c,c} \end{pmatrix}$$

- If random connections, $t_{k,\ell} = \pi_\ell$
(π_ℓ : proportion of class ℓ nodes)

$$\mathbf{T} = \begin{pmatrix} \pi_1 & \cdots & \pi_C \\ \vdots & \ddots & \vdots \\ \pi_1 & \cdots & \pi_C \end{pmatrix}$$

- If perfect homophily

$$\mathbf{T} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \mathbf{I}_{C \times C}$$

Label-label potential matrix is proportional to \mathbf{T}

Homophily

Citation graphs

■ **Cora**: 2708 nodes, 5429 edges, 7 classes, $F=1433$

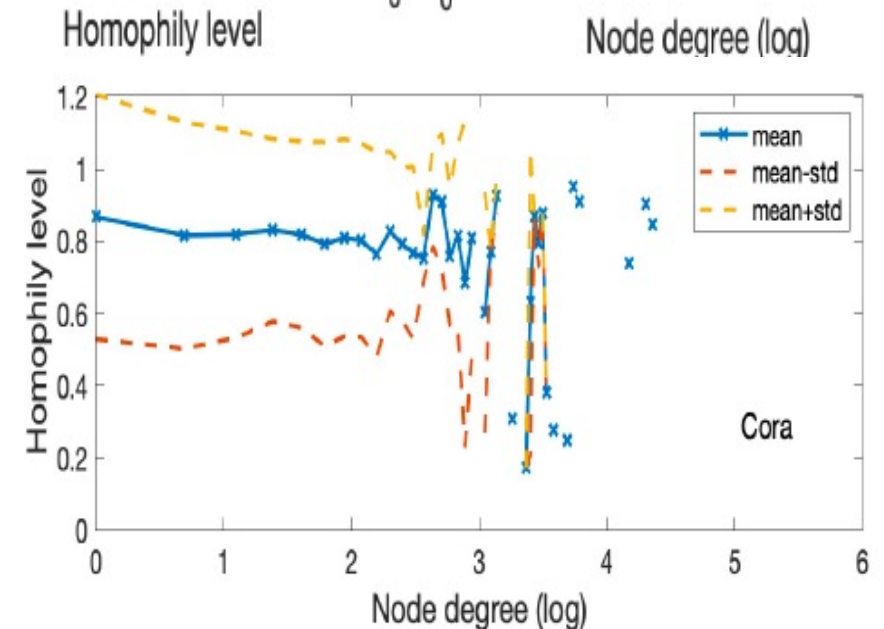
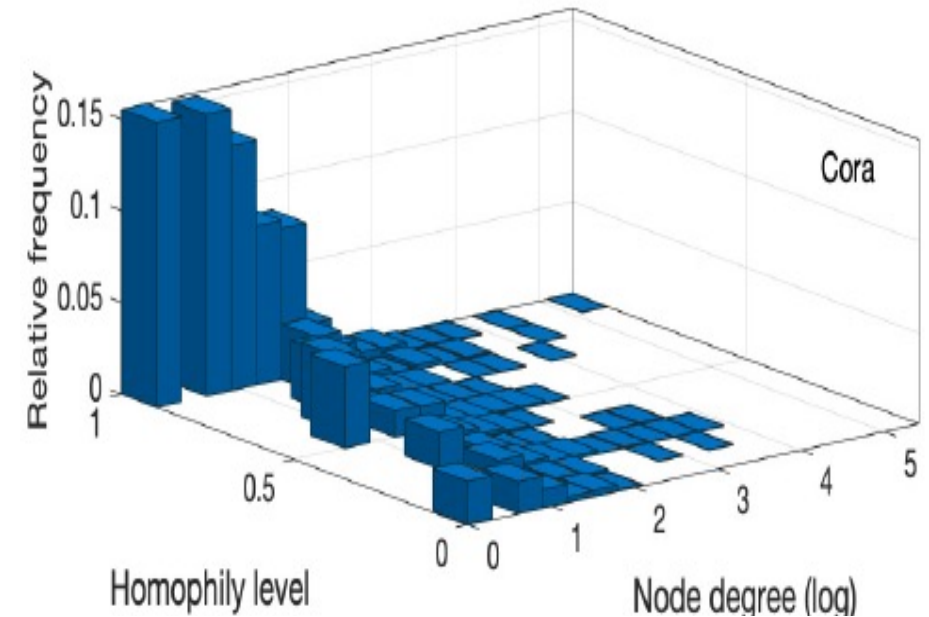
- Homophily: $p_h=0.81$; Connectivity= $1.5 \cdot 10^{-3}$
- Node degree: mean=3.9; std=5.2

1	0.74	0.03	0.01	0.08	0.05	0.04	0.05
2	0.04	0.77	0.06	0.08	0.02	0.03	0
3	0.01	0.02	0.92	0.03	0	0.01	0.01
4	0.06	0.02	0.02	0.84	0.05	0.02	-0.01
5	0.06	0.01	0	0.06	0.85	0.01	0.01
6	0.06	0.02	0.03	0.05	0.02	0.79	0.03
7	0.11	0	0	0.02	0.01	0.07	0.79
	1	2	3	4	5	6	7

Class transition matrix

SOTA accuracy: 84.6%

(training: 20 labeled nodes per class)



Homophily

Citation graphs

■ **Citeseer**: 3312 nodes, 4732 edges, 6 classes, $F = 3703$

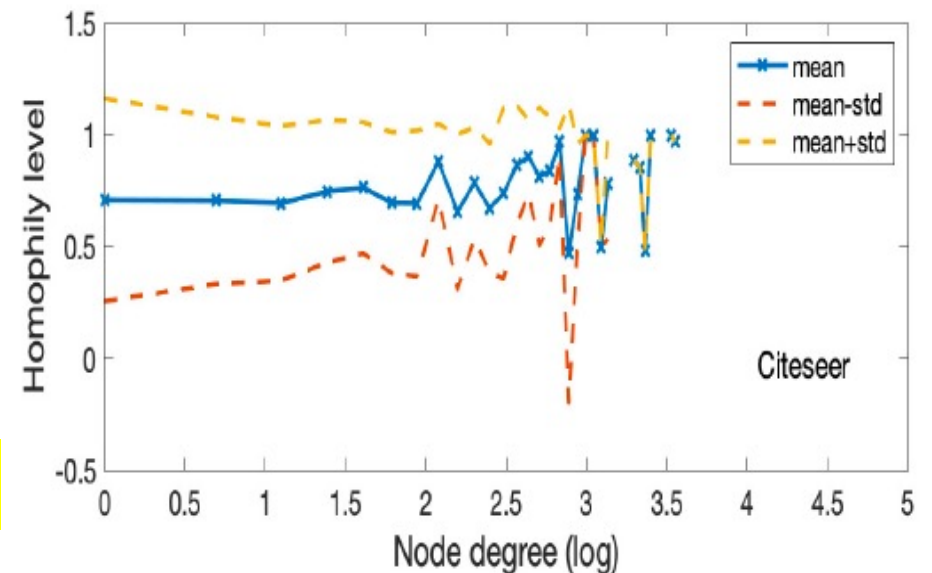
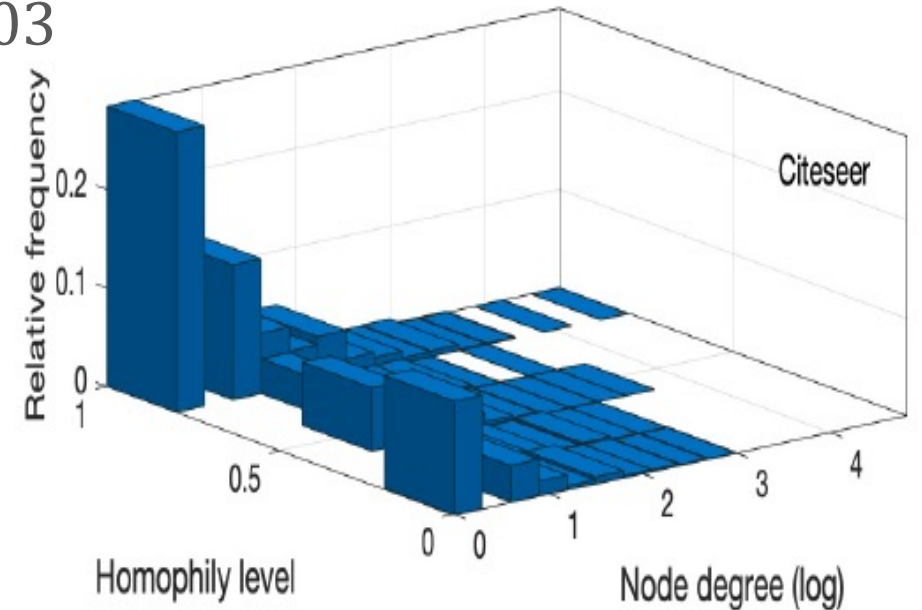
- Homophily: $p_h=0.76$; Connectivity= $0.8 \cdot 10^{-3}$
- Node degree: mean=2.7; std=3.4

1	0.74	0.03	0.01	0.08	0.05	0.04	0.05
2	0.04	0.77	0.06	0.08	0.02	0.03	0
3	0.01	0.02	0.92	0.03	0	0.01	0.01
4	0.06	0.02	0.02	0.84	0.05	0.02	-0.01
5	0.06	0.01	0	0.06	0.85	0.01	0.01
6	0.06	0.02	0.03	0.05	0.02	0.79	0.03
7	0.11	0	0	0.02	0.01	0.07	0.79
	1	2	3	4	5	6	7

Class transition matrix

SOTA accuracy: 73%

(training: 20 labeled nodes per class)



- Given a network and a node v , there are three distinct types of ‘correlations’ that can be utilized to classify v :
 1. The correlations between the label of v and the observed attributes of v .
 2. The correlations between the label of v and the observed attributes (including observed labels) of objects in the neighborhood of v .
 3. The correlations between the label of v and the unobserved labels of objects in the neighborhood of v .

- Exact collective inference ($p(\{y_v, v \in \mathcal{V}^U\} | \mathbf{X}, \{y_v, v \in \mathcal{V}^L\})$) is usually infeasible due to the complicated structures between object labels
- Collective node classification methods are iterative; based on message passing
- Different approaches:
 - **Relational classifiers**: iteratively updating class probabilities (label propagation); do not use node's features;
 - **Iterative classification**: iteratively updating classifier using node's features and estimated neighbors' labels
 - **Loopy belief propagation**: iteratively, neighbors pass on messages (beliefs) to each other

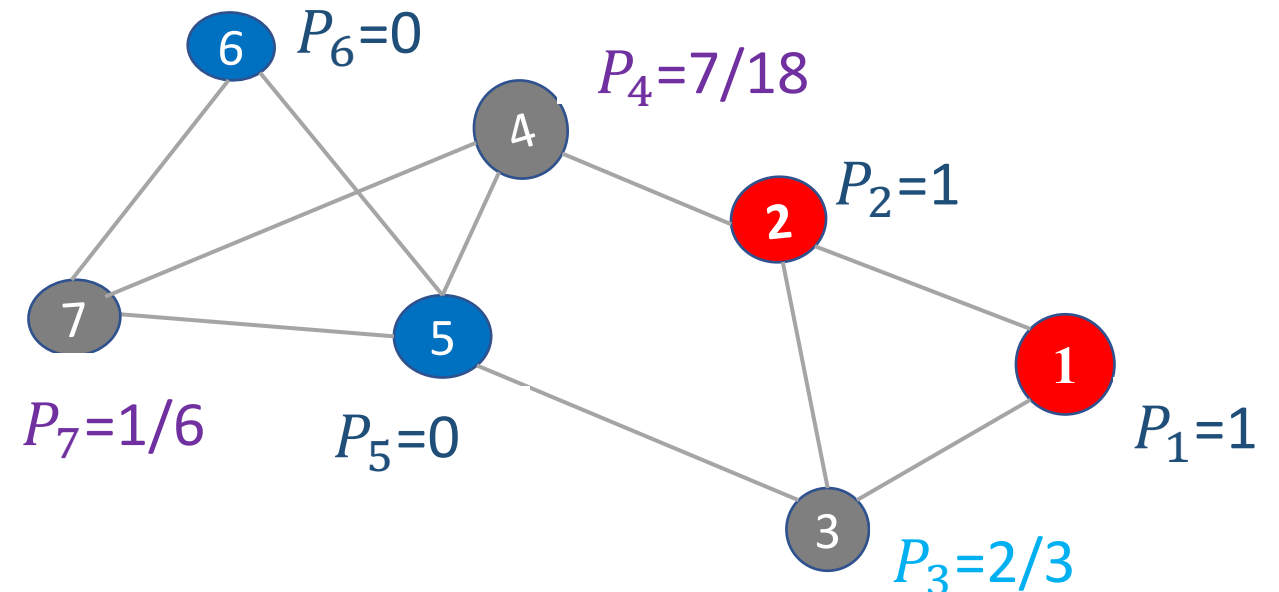
- **Basic idea**: iteratively set class probability y_v of node v as a weighted average of class probabilities of its neighbors

$$\Pr(y_v = k) \leftarrow \sum_{u \in \mathcal{N}_v} w_u \Pr(y_u = k) \quad \sum_{u \in \mathcal{N}_v} w_u = 1, \text{ e.g. } w_u = \frac{1}{|\mathcal{N}_v|}$$

- Initialization: initialize probabilities for unlabeled nodes to $\Pr(y_v = k) = \frac{1}{C}, \forall k$
- **Illustration**: $C = 2$, ($k = 1$ for red)

let $P_v = \Pr(y_v = 1)$, $w_u = \frac{1}{|\mathcal{N}_v|}, \forall u$

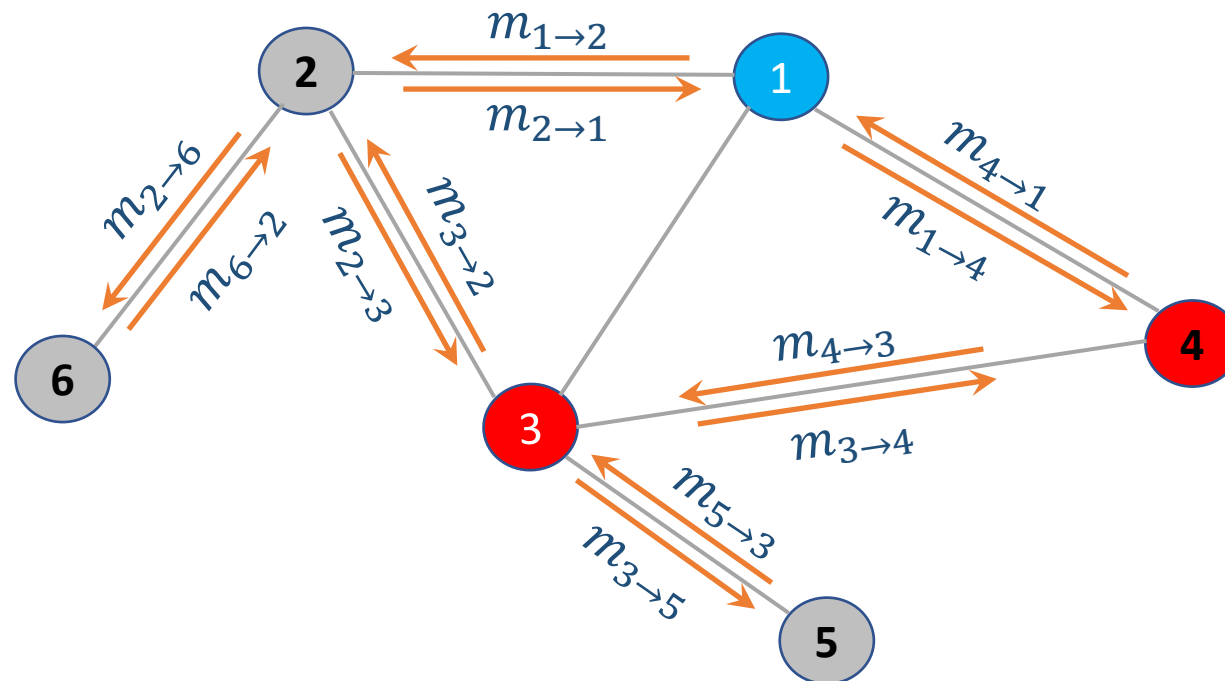
- Convergence not guaranteed
- Cannot leverage node features



- **Step 1**- using a training set:
 - Train a local classifier, $f_{\theta}(\mathbf{x}_v)$
 - Train another classifier, $g_{\phi}(\mathbf{x}_v, \mathbf{r}_v)$, where $\mathbf{r}_v \in \mathbb{R}^M$ is a relational vector (summary of labels of v 's neighbors, e.g class proportions)
- **Step 2**- on the test set
 1. First predict the labels of each node using classifier $f_{\theta}(\mathbf{x}_v)$
 2. Compute \mathbf{r}_v
 3. Update the nodes' labels using $g_{\phi}(\mathbf{x}_v, \mathbf{r}_v)$
 4. Repeat 2 and 3 until convergence

Convergence not guaranteed

- Belief Propagation is a dynamic programming approach to answering probability queries in a graph (e.g. probability of node v belonging to class k)
- Iteratively, neighbors pass on messages (beliefs) to each other
- The message from node u to v ($m_{u \rightarrow v}$) is node u 's belief about the class of node v



- Label-label potential matrix Ψ : $\psi(y_u, y_v) \propto \Pr[y_v | y_u]$
- Prior belief Φ : $\phi_u(y_u) \propto \Pr[y_u]$
- Message $m_{u \rightarrow v}(y_v)$: u 's message/belief of v being in class y_v

- Algorithm

1. Initialize all messages to $1/C$
2. Repeat for each node

$$m_{u \rightarrow v}(y_v) = \sum_{y_u=1}^C \psi(y_u, y_v) \phi(y_u) \prod_{w \in \mathcal{N}_u \setminus v} m_{w \rightarrow u}(y_u) \quad y_v = 1, \dots, C$$

Label-label potential
 Sum over all states
 Prior belief
 All messages sent by neighbors from previous round

- After convergence, compute node v 's belief of being in class y_v :

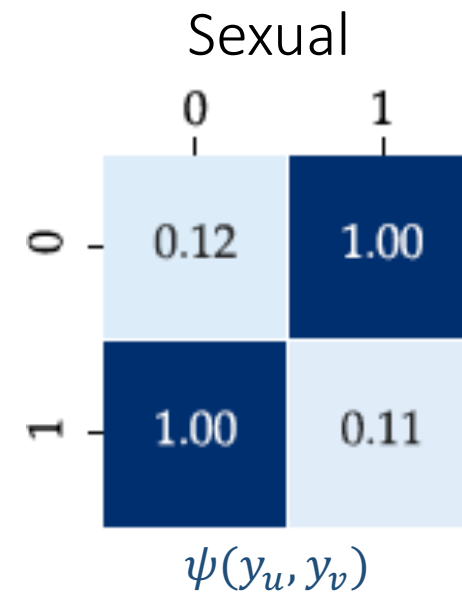
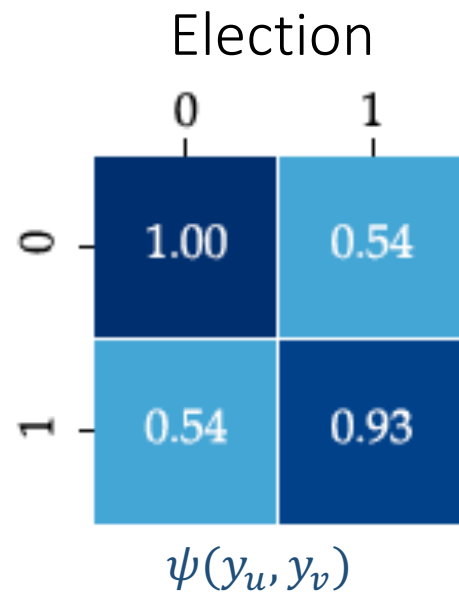
$$b_v(y_v) = \overset{\text{Prior belief}}{\phi_v(y_v)} \overset{\text{All messages from neighbors}}{\prod_{u \in \mathcal{N}_v} m_{u \rightarrow v}(y_v)} \quad y_v = 1, \dots, C$$

Convergence not guaranteed (because of cycles)

- When nodes have features (Graph Belief Propagation Networks [Gia et al., 2021]):
 1. Compute self-potentials (replace prior beliefs) using a local classifier (e.g. MLP), $g_{\theta}(y_v; \mathbf{x}_v)$
 2. Initialization: $p_v^{(0)}(y_v) = g_{\theta}(y_v; \mathbf{x}_v)$ and $m_{u \rightarrow v}^{(0)}(y_v) = 1/C$
 3. Run T iterations of
$$p_v^{(t)}(y_v) = p_v^{(0)}(y_v) \prod_{u \in \mathcal{N}_v} m_{u \rightarrow v}^{(t)}(y_v), \quad m_{u \rightarrow v}^{(t)}(y_v) = \sum_{y_u=1, \dots, C} \psi(y_u, y_v) \frac{p_u^{(t-1)}(y_u)}{m_{v \rightarrow u}^{(t-1)}(y_u)}$$
- The final belief is $p_v^{(T)}(y_v)$. In practice, for numerical stability, we work in log-space.
- Training: maximize the log-marginal-likelihood of training labels

$$\{\theta^*, \Psi^*\} = \arg \max_{\{\theta, \Psi\}} \sum_{v \in \mathcal{V}^L} \log p(y_v | \mathbf{X}) \cong \arg \max_{\{\theta, \Psi\}} \sum_{v \in \mathcal{V}^L} \log p_v(y_v)$$

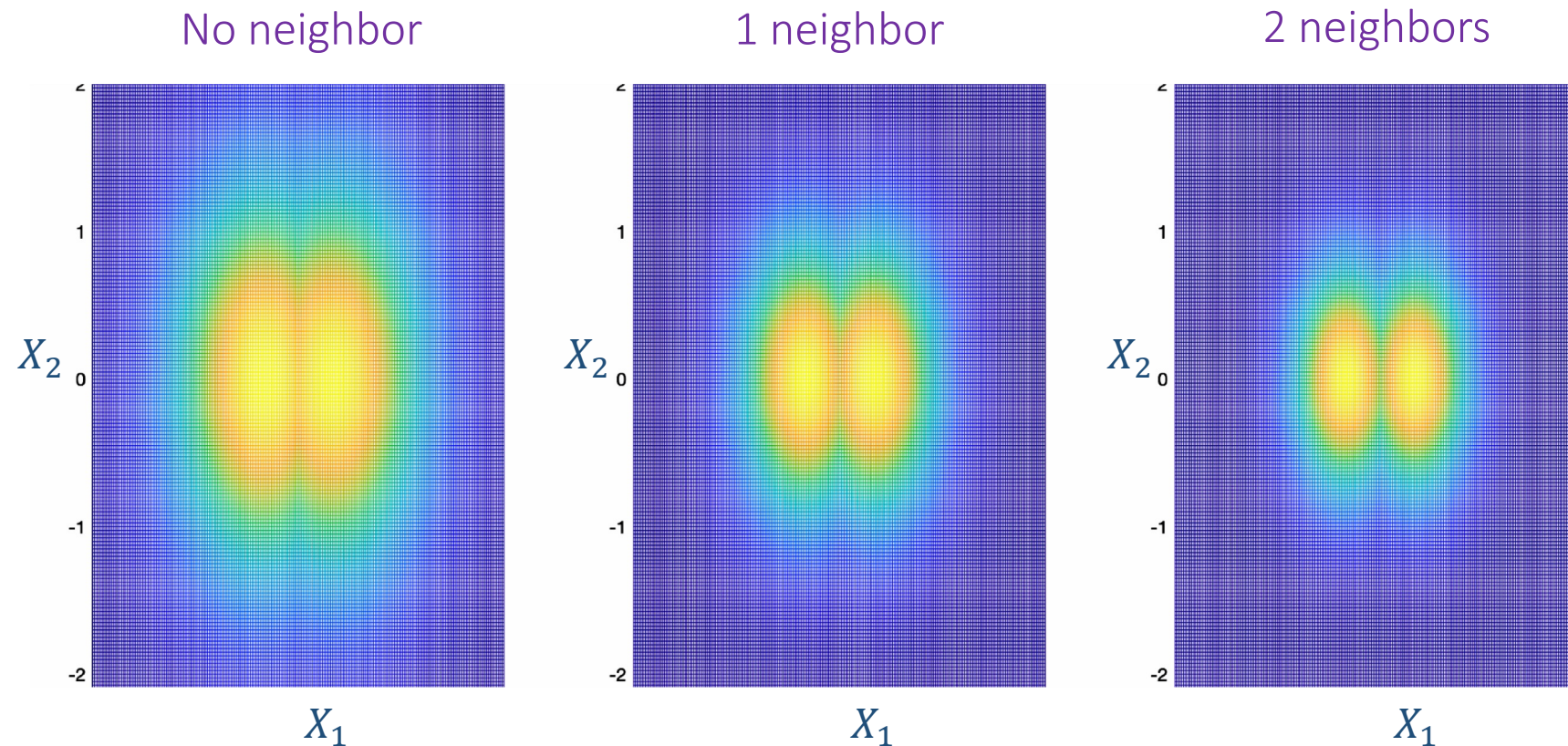
dataset	MLP	GCN	SAGE	GAT	GMNN	DeeperGNN	GBPN-I	GBPN
Cora	72.1 \pm 1.3	87.1 \pm 0.7	86.9 \pm 0.8	87.1 \pm 0.9	86.4 \pm 0.9	87.2 \pm 0.8	85.6 \pm 0.7	86.4 \pm 0.9
CiteSeer	71.2 \pm 0.9	73.5 \pm 1.0	73.2 \pm 1.0	73.1 \pm 1.2	72.9 \pm 1.2	73.9 \pm 0.8	74.7 \pm 1.3	74.8 \pm 0.8
PubMed	86.5 \pm 0.2	87.1 \pm 0.3	87.8 \pm 0.4	88.1 \pm 0.3	86.7 \pm 0.2	84.7 \pm 0.3	88.4 \pm 0.3	88.5 \pm 0.3
CS	94.2 \pm 0.2	93.2 \pm 0.2	93.7 \pm 0.2	94.0 \pm 0.3	93.3 \pm 0.3	94.9 \pm 0.2	95.5 \pm 0.2	95.5 \pm 0.3
Physics	95.8 \pm 0.1	96.1 \pm 0.1	96.3 \pm 0.1	96.3 \pm 0.1	96.1 \pm 0.1	96.7 \pm 0.1	96.9 \pm 0.1	96.9 \pm 0.1
Election	89.6 \pm 0.6	88.0 \pm 0.6	90.8 \pm 0.6	90.5 \pm 0.7	87.3 \pm 0.7	85.4 \pm 0.7	90.1 \pm 0.8	90.3 \pm 0.9
Sexual	74.5 \pm 1.4	83.9 \pm 1.2	93.3 \pm 0.8	93.6 \pm 0.9	77.0 \pm 1.7	65.0 \pm 1.1	97.1 \pm 0.5	97.4 \pm 0.4



- GNN methods ‘ignore’ the correlations between object labels; they focus on learning effective object representations for label prediction
- GNN infers the label distributions of nodes **independently**
- GNN exploits the correlations through **neighborhood aggregation**, e.g. (1st GNN layer)

$$h_v = \frac{1}{|\mathcal{N}_v|} \left(\mathbf{x}_v + \sum_{u \in \mathcal{N}_v} \mathbf{x}_u \right)$$

- Illustration of the effects of neighborhood aggregation (Gaussian features)



- Averaging the feature vector of \mathbf{x}_v with those of the node v 's neighbors reduces the overlap between candidate distributions, if the neighbors belong to the same class as v

- 1st-layer node representation:

$$\mathbf{h}_v^{(1)} = \text{MLP}^{(1)} \left((1 + \epsilon^{(1)}) \mathbf{x}_v + \sum_{u \in \mathcal{N}_v} \mathbf{x}_u \right)$$

\equiv

$$\mathbf{h}_v^{(1)} = \text{MLP}^{(1)} \left(\mathbf{x}_v + \sum_{u \in \mathcal{N}_v} \alpha^{(1)} \mathbf{x}_u \right) \quad \alpha^{(1)} \leq 1$$

- Fixed weight for all neighbors at each layer
- GIN is as powerful as *Weisfeiler-Lehman test* of isomorphism in distinguishing graph structures unlike GCN according to [Xu et al., 2019]

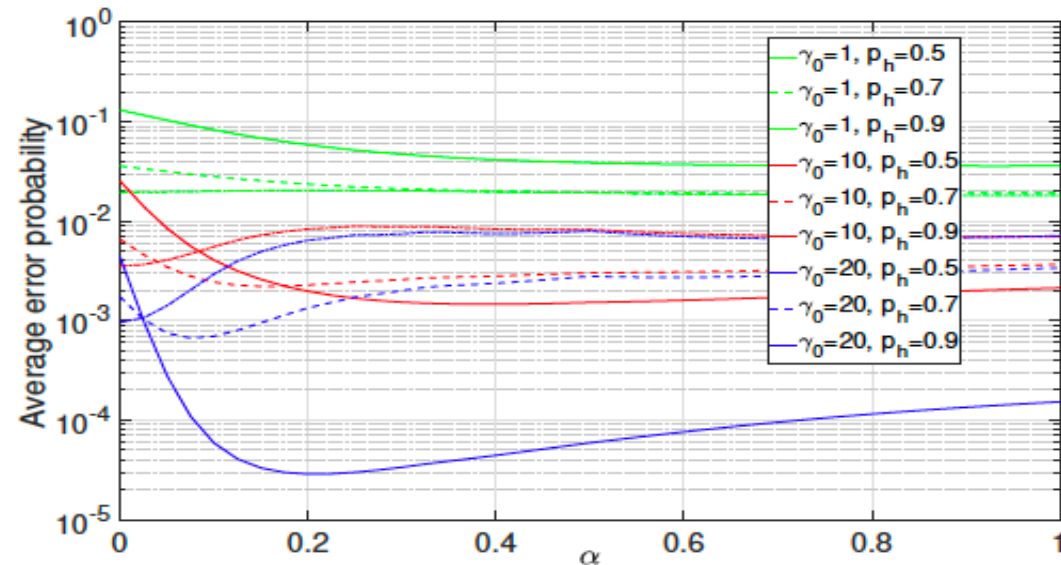
- Since real-world graphs are not purely homophilous, neighbors must not be trusted 100% -> a smaller weight must be given to neighbors if node features are more important than graph structure for node classification

$$\mathbf{h}_v^{(1)} = \text{MLP}^{(1)} \left(\mathbf{x}_v + \sum_{u \in \mathcal{N}_v} \alpha^{(1)} \mathbf{x}_u \right) \quad \alpha^{(1)} \leq 1$$

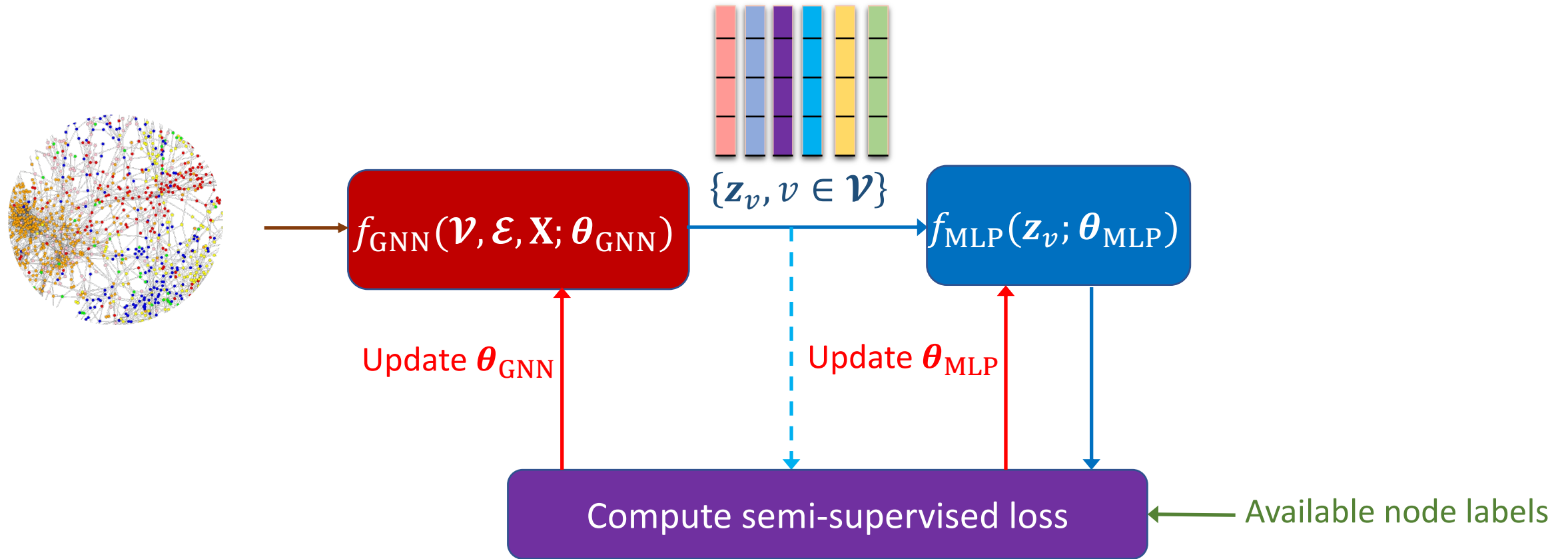
- The higher the homophily rate, p_h , the higher the value of $\alpha^{(1)}$

Example: block stochastic model with 2 classes and Gaussian feature vectors of different means (μ_1 and μ_2) and same covariance $\sigma^2 \mathbf{I}$

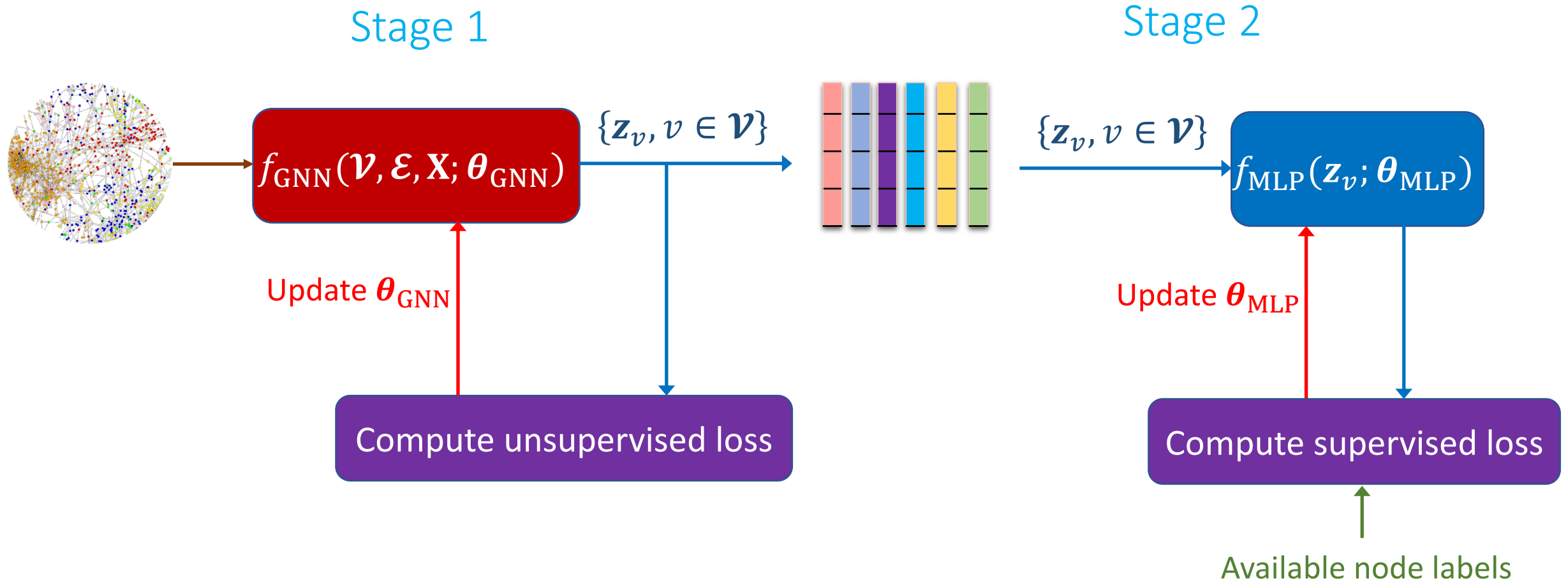
$$\gamma_0 = \frac{\|\mu_2 - \mu_1\|^2}{\sigma^2}$$



- **Approach 1. End-to-end semi-supervised learning**
 - Optimize the embeddings (e.g. GNN) and classifier jointly w.r.t. the downstream task (node or graph classification) using available labels. One may also add a regularization term...
- **Approach 2. Two-stage semi-supervised learning**
 - **Stage 1:** optimize the GNN in an unsupervised manner to generate node embeddings
 - **Stage 2:** build a supervised classifier using the node embeddings and available labels

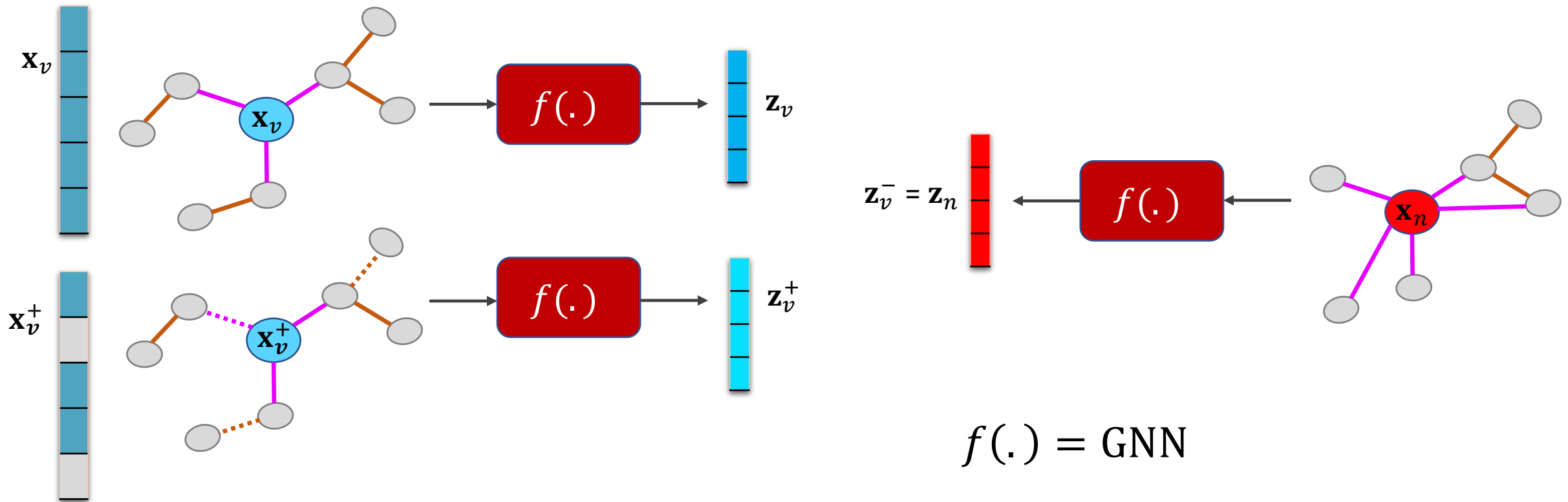


$f_{\text{MLP}}(\mathbf{z}_v; \theta_{\text{MLP}})$ is generally a shallow MLP (1 or 2 hidden layers)



The unsupervised loss generally requires negative sampling

- **Reminder:** GraphCL learns node embeddings by maximizing similarity between representations of **perturbed versions of the same node's** local subgraph.



- Classification is performed by training a shallow MLP on node embeddings

GNN-based methods

GraphCL (evaluation)

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,707	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
ogbn-arxiv	Transductive	169,343	1,166,243	128	40	Time
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilabel)	44,906/6,154/5,524 (20/2/2 graphs)

- Transductive Learning Datasets:
 - Cora, Citeseer, Pubmed, ogbn-arxiv
- Inductive Learning Datasets:
 - PPI (Protein-Protein Interaction): Multiple graphs for different human tissues; node features: positional gene sets, motif gene sets, immunological signatures
 - Reddit: nodes are Reddit posts represented by GloVe embeddings

Transductive				
Method	Cora	Citeseer	Pubmed	ogbn-arxiv
Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$	$55.50 \pm 0.23\%$
DeepWalk [69]	67.2%	43.2%	65.3%	$70.07 \pm 0.13\%$
DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$	–
EP-B [21]	$78.1 \pm 1.5\%$	$71.0 \pm 1.4\%$	$79.6 \pm 2.1\%$	$68 \pm 0.00\%$
DGI [80]	$82.3 \pm 0.6\%$	$71.8 \pm 0.7\%$	$76.8 \pm 0.6\%$	$70.18 \pm 0.12\%$
GraphCL	$83.6 \pm 0.5\%$	$72.5 \pm 0.7\%$	$79.8 \pm 0.5\%$	$70.18 \pm 0.17\%$
GraphCL*	$84.6 \pm 0.4\%$	$73.1 \pm 0.6\%$	$80.1 \pm 0.5\%$	$71.38 \pm 0.13\%$
GCN(supervised)[51]	81.5%	70.3%	79.0%	$71.74 \pm 0.002\%$

[Hafidi et al., 2021a]:

GraphCL* (optimized GraphCL, i.e. best negative sampling strategy, choice of encoders, and embedding size)

Inductive			
	Method	Reddit	PPI
Unsupervised	Raw features	0.585	0.422
	GraphSage-GCN [38]	0.908	0.465
	GraphSage-mean [38]	0.897	0.486
	GraphSage-LSTM [38]	0.907	0.482
	GraphSage-pool [38]	0.892	0.502
	DGI [80]	0.940 ± 0.001	0.638 ± 0.002
	GraphCL	0.951 ± 0.01	0.659 ± 0.006
	GraphCL*	0.960 ± 0.01	0.841 ± 0.004
Supervised	FastGCN [10]	0.937	—
	GaAN [94]	0.958 ± 0.001	0.969 ± 0.002

[Hafidi et al., 2021a]:

- **Goal**: develop a Bayesian framework for ‘aggregating’ neighbors of different orders
- **Idea**: use Markov model to characterize **first and higher-order** label-label transition matrices.
- **Notations**
 - \mathcal{V}_u : set of nodes ‘helping’ with the classification of node u
 - $\mathcal{X}_u = \{\mathbf{x}_u\} \cup \{\mathbf{x}_v, v \in \mathcal{V}_u\}$: feature vectors of u and \mathcal{V}_u
 - $D_m(\mathbf{x})$: distribution of \mathbf{x} in class m ; an assumption is made on its shape and associated parameters are estimated

- **Aim:** compute the posterior probability that node u belongs to class m given \mathcal{X}_u and some information about the graph, \mathcal{I}_G , (typically its partial connectivity through u and \mathcal{V}_u)

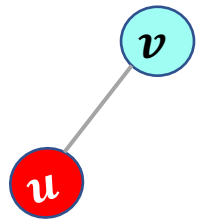
$$P_u(k) = \Pr(y_u = k \mid \mathcal{X}_u, \mathcal{I}_G)$$

- Using Bayes rule:

$$P_u(k) \propto \Pr(\mathcal{X}_u \mid y_u = k, \mathcal{I}_G) \Pr(y_u = k) = Q_u(k) \pi_k$$

(assuming $\Pr(y_u = k \mid \mathcal{I}_G) = \Pr(y_u = k)$ which is reasonable)

- Using one first-order neighbor only and Markov model:



$$Q_u(k) = \sum_{k'} \Pr(\{x_u, x_v\} | y_u = k, y_v = k') \Pr(y_v = k' | y_u = k) \quad \mathcal{I}_{\mathcal{G}}: \text{not relevant here}$$

$$\Rightarrow Q_u(k) = D_k(\mathbf{x}_u) \sum_{\ell} t_{k,\ell} D_{\ell}(\mathbf{x}_v)$$

Mixture distribution

- $t_{k,\ell} = \Pr(y_v = \ell | y_u = k)$: Label-label transitions
- $Q_u(k)$ can also be interpreted as a **nonlinear aggregation** of node features
- The aggregation in GNN is linear except for Graph Attention Network (GAT)

- Further, if 2 classes and nodes have $F = 1$ Gaussian feature only:

$$Q_u(1) = D_1(x_u)(t_{1,1}D_1(x_v) + t_{1,2}D_2(x_v))$$

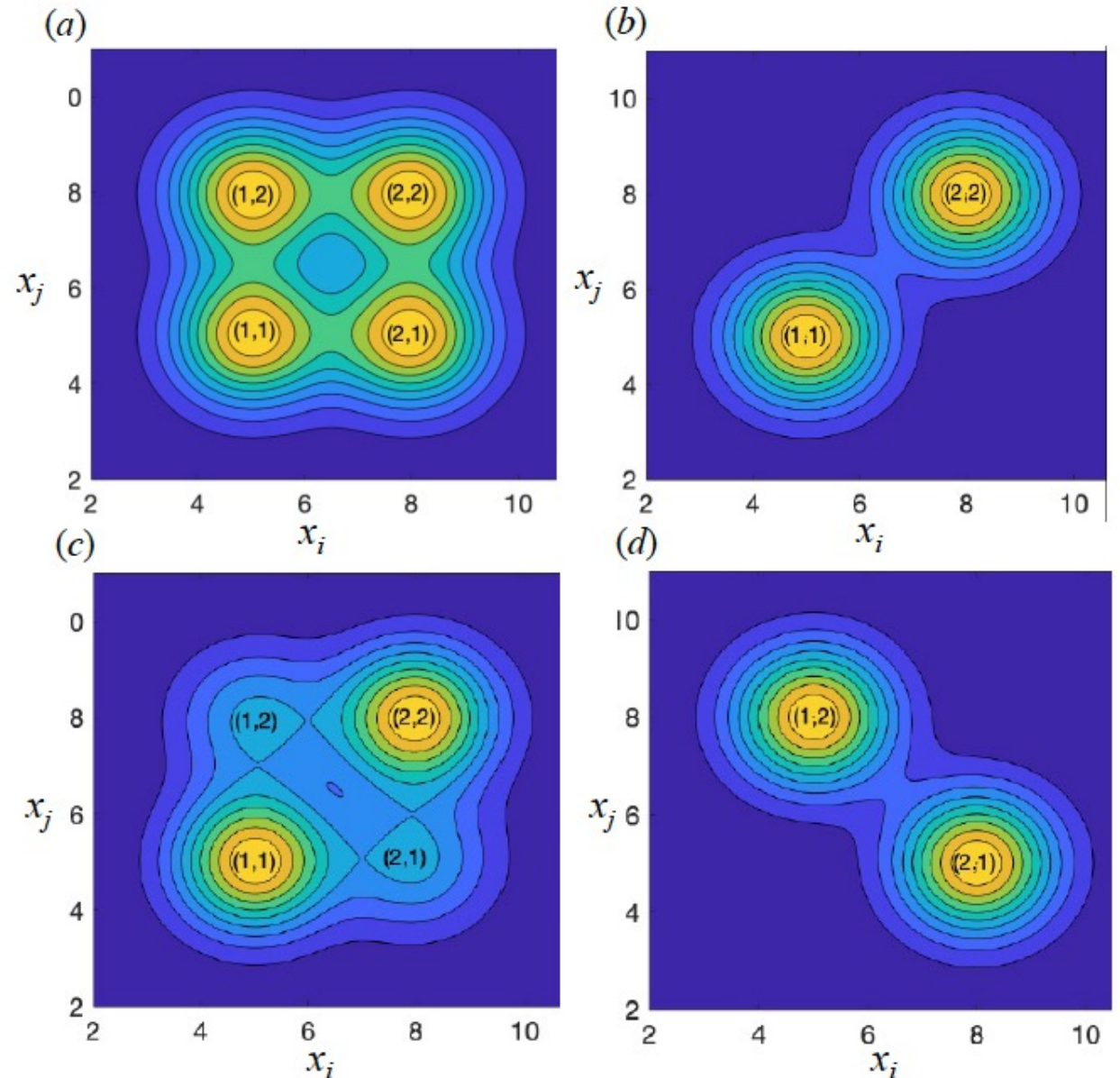
$$Q_u(2) = D_2(x_u)(t_{2,1}D_1(x_v) + t_{2,2}D_2(x_v))$$

(a): $t_{1,1} = t_{2,2} = 0.5$ (random connection)

(b): $t_{1,1} = t_{2,2} = 1$ (pure homophily)

(c): $t_{1,1} = t_{2,2} = 0.8$

(d): $t_{1,2} = t_{2,1} = 1$ (pure heterophily)



- Conditioning on labels of \mathcal{V}_u and then averaging:

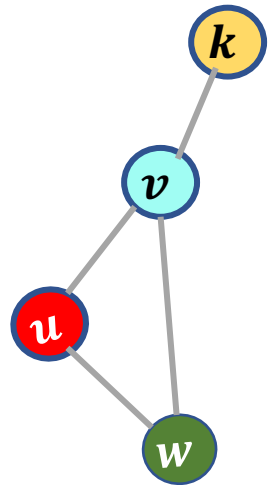
$$Q_u(k) = \sum_{\{k_v\}_{v \in \mathcal{V}_u}} \Pr(\mathbf{x}_u | y_u = k, \{y_v = k_v\}_{v \in \mathcal{V}_u}) \Pr(\{y_v = k_v\}_{v \in \mathcal{V}_u} | y_u = k, \mathcal{I}_G)$$

(when conditioning on labels of \mathcal{V}_u , \mathcal{I}_G becomes redundant)

$$\rightarrow Q_u(k) = D_k(\mathbf{x}_u) \sum_{\{k_v\}_{v \in \mathcal{V}_u}} \left(\prod_{v' \in \mathcal{V}_u} D_{k_{v'}}(\mathbf{x}_{v'}) \right) \Pr(\{y_v = k_v\}_{v \in \mathcal{V}_u} | y_u = k, \mathcal{I}_G)$$

- For tractability, we ignore the correlations between neighbors' labels, i.e.

$$\Pr(\{y_v = k_v\}_{v \in \mathcal{V}_u} | y_u = k, \mathcal{I}_G) \rightarrow \prod_{v \in \mathcal{V}_u} \Pr(y_v = k_v | y_u = k, \mathcal{I}_G)$$

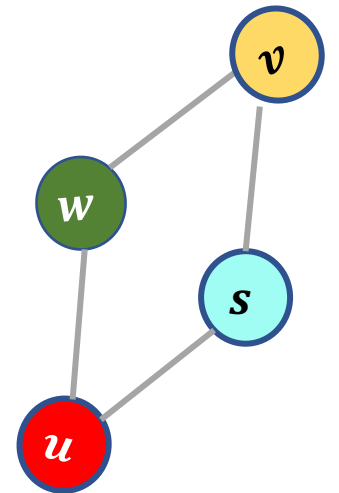


- Resulting posterior probability

$$P_u(k) = D_k(x_u) \prod_{v \in \mathcal{V}_u} \left(\sum_{\ell=1}^K r_{u,v}(k, \ell) D_\ell(x_v) \right) \pi_k$$

$$r_{u,v}(k, \ell) = \Pr(y_v = \ell \mid y_u = k, \mathcal{I}_G)$$

- $r_{u,v}(k, \ell)$: probability that neighbor v is in class ℓ given that node u is in class k and graph information \mathcal{I}_G
- Even under Markov modeling, deriving $r_{u,v}(k, \ell)$ for non-first order neighbors is tedious due combinatorial complexity



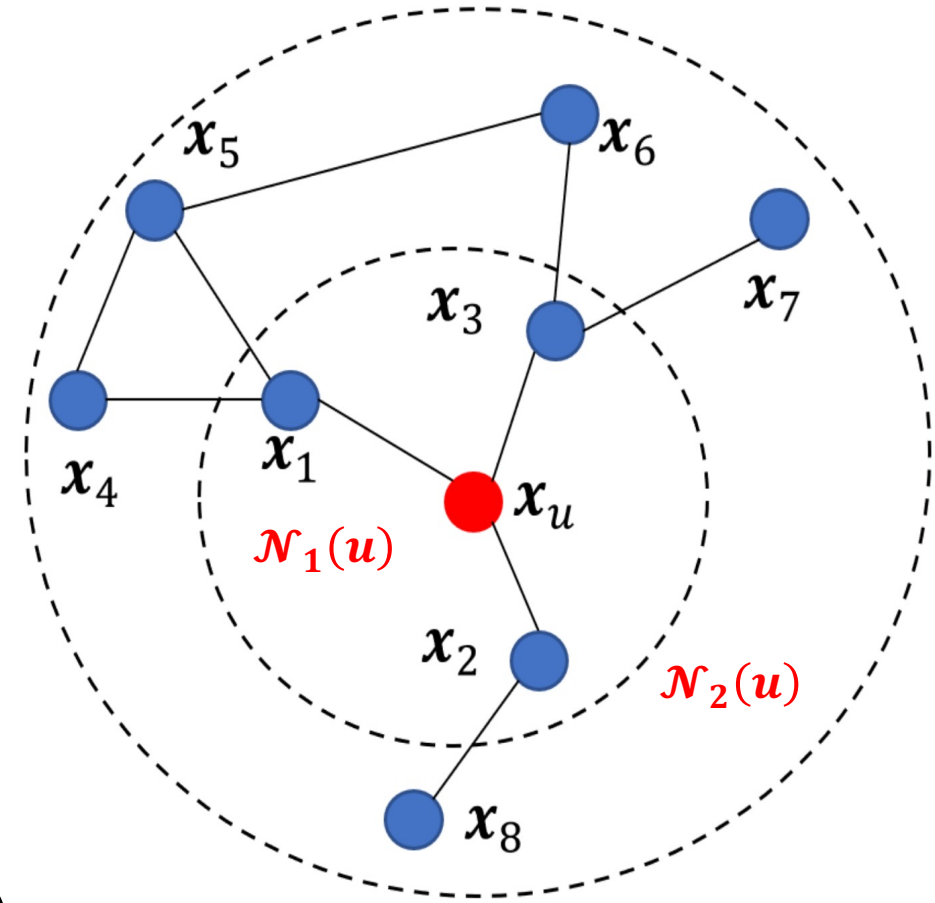
- To obtain a **practical algorithm**, we assume:

$$\mathcal{I}_{\mathcal{G}} = \{\text{distance of each node of } \mathcal{G} \text{ to root } u\}$$

- So, we structure \mathcal{V}_u as :

$$\mathcal{V}_u = \mathcal{N}_1(u) \cup \mathcal{N}_2(u) \dots \cup \mathcal{N}_{\Delta_u}(u)$$

- $\mathcal{N}_d(u)$: set of **d -hops** neighbors of u and Δ_u is the maximum distance from node u
- So, $r_{u,v}(k, \ell)$ no longer depends on the specific path(s) connecting u to v



- We assume $r_{u,v}(k, k')$ is the same for all pairs nodes (u, v) that are a fixed number of hops apart:

$$r_{u,v}(k, k') = r^{(d)}(k, k')$$

- Introducing hyperparameter γ_d to control the contributions of neighbors of different orders:

$$\hat{y}_u = \arg \max_k \pi_k D_k(\mathbf{x}_u) \prod_{d=1}^{\Delta_u} \prod_{v \in \mathcal{N}_d(u)} \left(\sum_{\ell=1}^K \left(r^{(d)}(k, \ell) \right)^{\gamma_d} D_{\ell}(\mathbf{x}_v) \right)$$

- For $d = 1$: $r^{(1)}(k, \ell) = t_{k, \ell}$ (transition matrix defined previously)
- Let \mathcal{C}_d : information that considered nodes are connected in d hops:

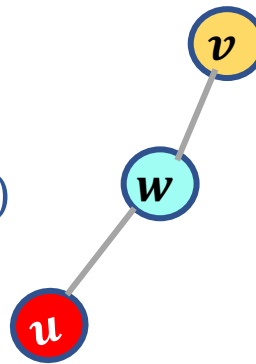
$$r^{(d)}(k, \ell) = \Pr(y_v = \ell | y_u = k, \mathcal{C}_d)$$

- Let $\mathbf{R}^{(d)} = \{r^{(d)}(k, \ell)\}$. We show that

$$\mathbf{R}^{(d)} = (\mathbf{R}^{(1)})^d = \mathbf{T}^d$$

- $d = 2$ case

$$\begin{aligned} r^{(2)}(k, \ell) &= \Pr(y_v = \ell | y_u = k, \mathcal{C}_2) \\ &= \sum_{k'} \Pr(y_v = \ell | y_u = k, y_w = k', \mathcal{C}_2) \\ &\quad \Pr(y_w = k' | y_u = k, \mathcal{C}_1) \end{aligned}$$



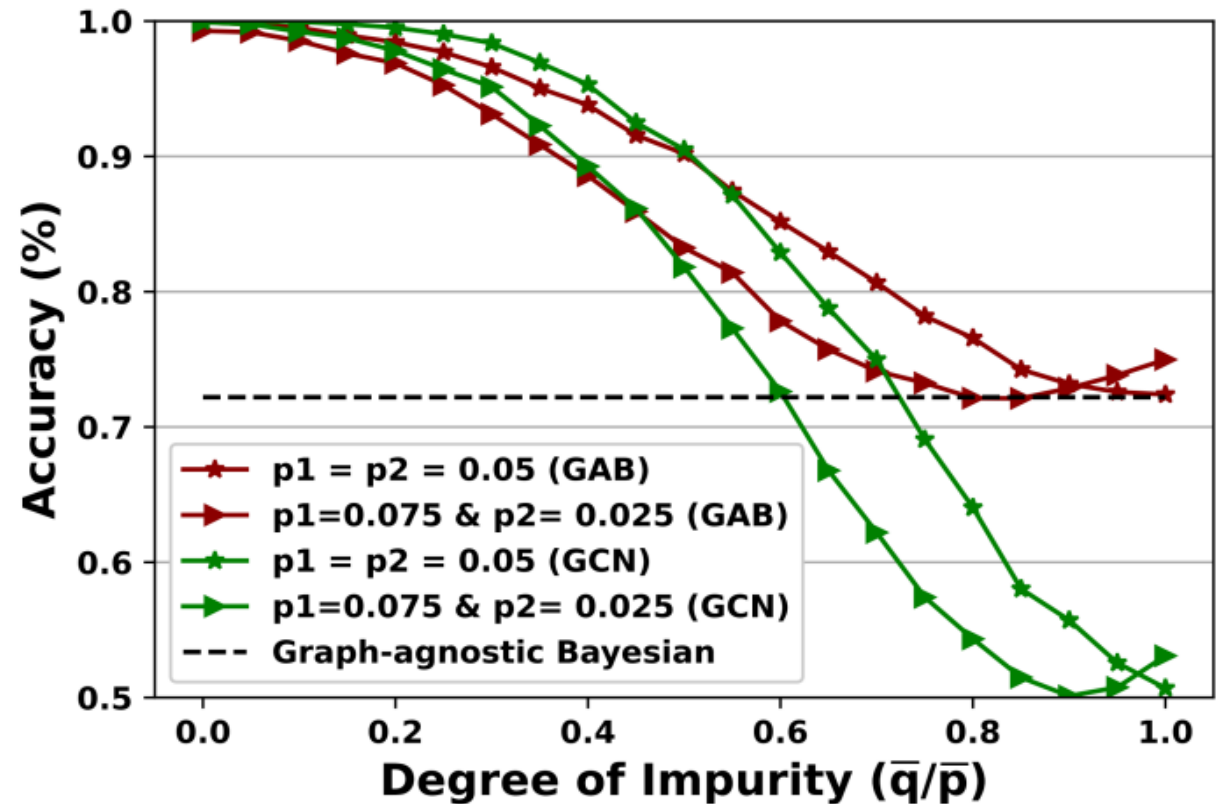
$$\begin{aligned} r^{(2)}(k, \ell) &= \sum_{k'} \Pr(y_v = \ell | y_w = k', \mathcal{C}_1) r^{(1)}(k, k') \\ r^{(2)}(k, \ell) &= \sum_{k'} r^{(1)}(k', \ell) r^{(1)}(k, k') \end{aligned}$$

- Arbitrary d $\mathbf{R}^{(d)} = \mathbf{R}^{(d-1)}$

- For continuous feature vectors $D_k(\cdot)$ is assumed multivariate Gaussian; mean vectors and covariance matrices are estimated
- For binary features, they are assumed to be independent Bernoulli random variables; probabilities estimated using Laplace smoothing
- Another approach: MLP-based estimation of likelihood $D_k(\mathbf{x}_u)$. This works for all types of feature

Synthetic Datasets Experiment

- 2 classes; $D_k(\cdot) = \mathcal{N}(\boldsymbol{\mu}_k, \text{diag}(\boldsymbol{\sigma}^2))$; elements of $\boldsymbol{\mu}_k$ random from $[0,1]$
elements of $\boldsymbol{\sigma}^2$ random from $[0.5,3.5]$
- $\pi_1 = \pi_2$ (same priors)
- $t_{k,\ell} = t_{\ell,k}$ (symmetry)
- $p(k)$: Pr(2 randomly selected nodes of class k are directly connected)



$$\bar{p} = \text{average}(\{t_{k,k}\}); \bar{q} = \text{average}(\{t_{k,\ell}\}, k \neq \ell)$$



GAB classifier is more robust to 'impurities'

- Considered real datasets

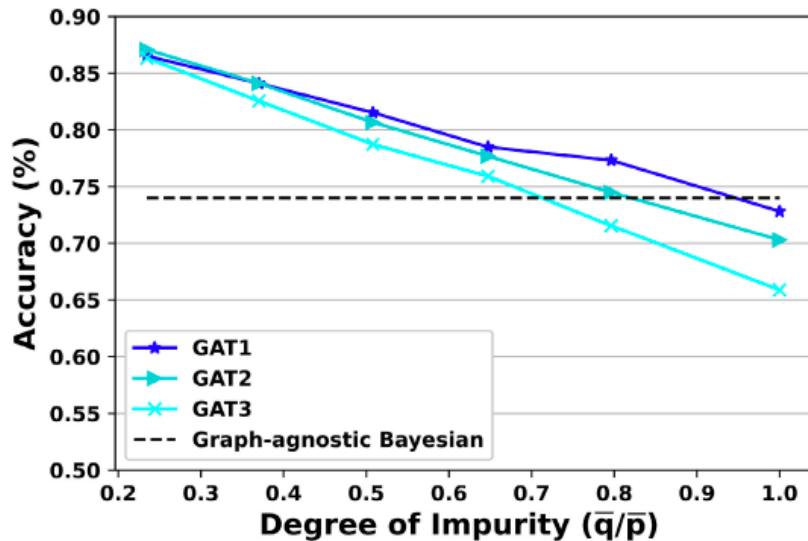
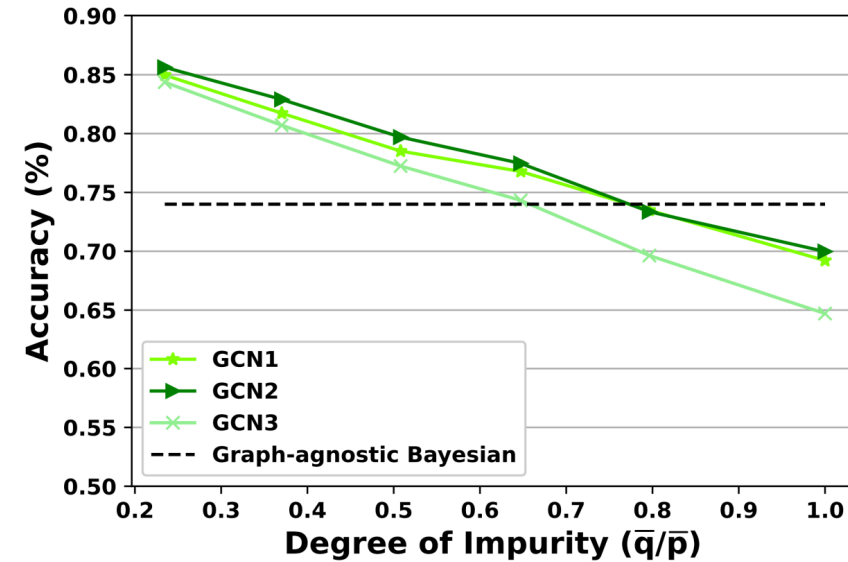
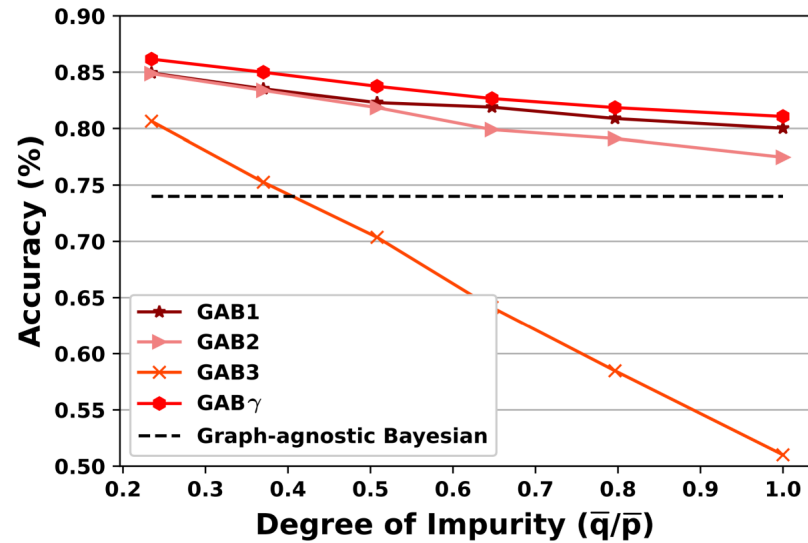
Dataset	Classes	Nodes	Edges	Features
Cora	7	2,708	5,429	1,433
CiteSeer	6	3,327	4,732	3,707
PubMed	3	19,717	44,338	500
CS	15	18,333	163,788	6,805
Physics	5	34,493	495,924	8,415
Sexual	2	1,888	2,096	20

- Using 30% of the graph for training:

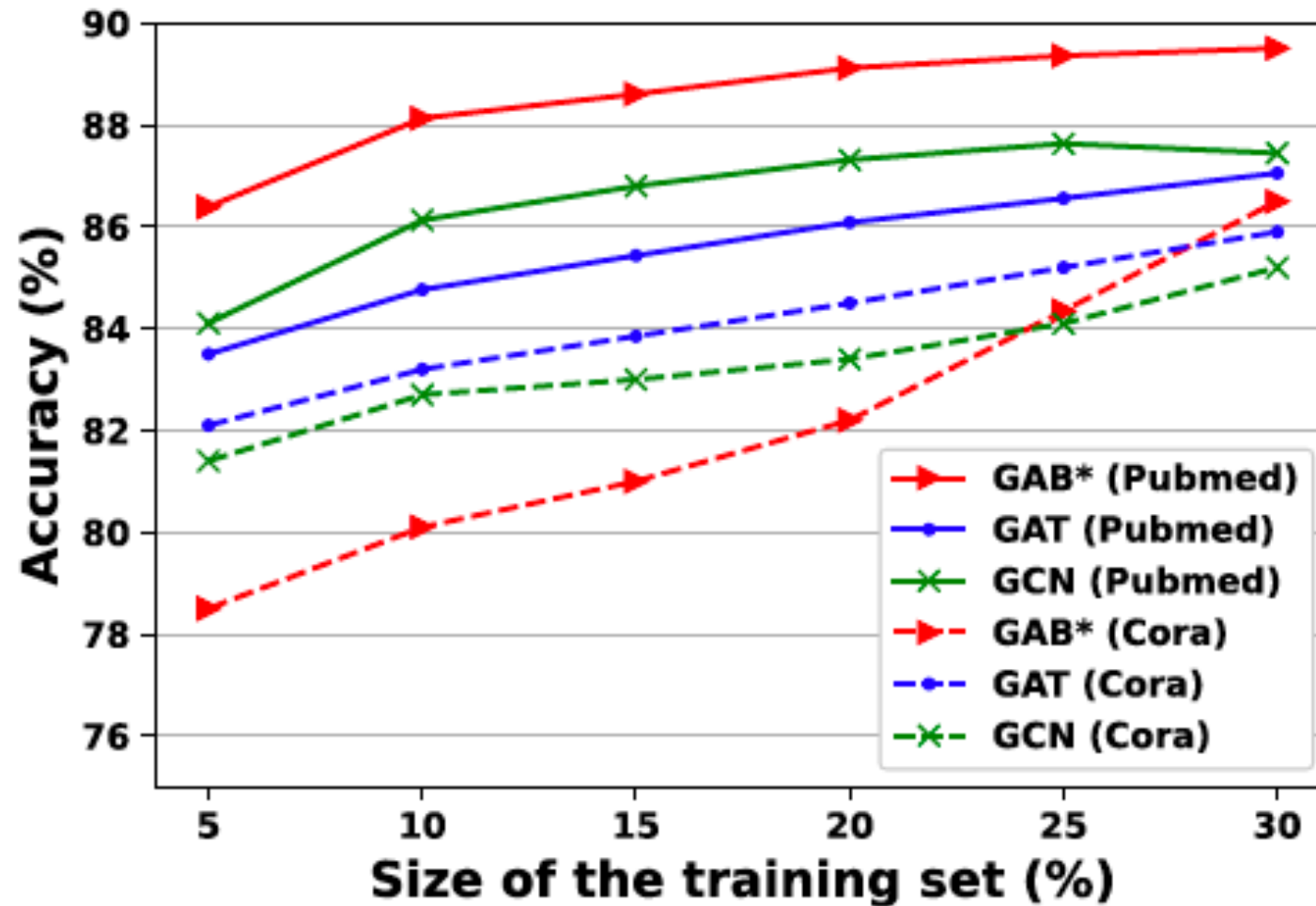
Dataset	MLP	GCN	SAGE	GAT	GMNN	DeeperGCN	GBPN	GAB γ	GAB \star
Cora	72.1(9)	87.1(3)	86.9(4)	87.1(2)	86.4(6)	87.2(1)	86.4(6)	86.9(4)	86.3(8)
CiteSeer	71.2(9)	73.5(5)	73.5(5)	73.1(7)	72.9(8)	73.9(4)	74.8(2)	75.2(1)	74.7(3)
PubMed	86.5(6)	87.1(5)	87.8(4)	88.1(3)	86.7(7)	84.7(9)	88.5(2)	86.4(8)	89.5(1)
CS	94.2(5)	93.2(9)	93.7(7)	94.0(6)	93.3(8)	94.9(3)	95.5(1)	94.5(4)	95.2(2)
Physics	95.8(9)	96.1(6)	96.3(5)	96.3(6)	96.1(8)	96.7(3)	96.9(1)	96.4(4)	96.7(2)
Sexual	74.5(8)	83.9(6)	93.3(5)	93.6(4)	77.0(7)	65.0(9)	97.4(1)	96.5(3)	97.1(2)
	84.4(7.6)	86.8(5.6)	88.9(5)	89.0(4.6)	85.2(7.3)	83.0(4.8)	89.9(2.2)	90.1(2)	

- In GAB \star , likelihoods are estimated using a 2-hidden layer MLP
- The best versions of each approach are selected
- GAB optimized up to fifth-order neighborhood

- Results when adding more impurities to Cora



- Effects of size of training set



- Simple scenario: 2 classes, 1st-order neighborhood only
- GAB classifier: decide $y_u = 1$ if

$$\log\left(\frac{\pi_1}{\pi_2}\right) + \log\left(\frac{\bar{q}\pi_1 + p(2)\pi_2}{p(1)\pi_1 + \bar{q}\pi_2}\right) + \log(S(\mathbf{x}_u)) + \sum_{v \in \mathcal{N}(u)} \log\left(\frac{\bar{q}\pi_2 + p(1)\pi_1 S(\mathbf{x}_v)}{p(2)\pi_2 + \bar{q}\pi_1 S(\mathbf{x}_v)}\right) > 0$$

Contribution of node
u's features
Contribution of
neighbors

- $$S(\mathbf{x}) = \frac{D_1(\mathbf{x})}{D_2(\mathbf{x})}$$

- Node features nonlinearly aggregated unlike GNN except for GAT

- Further, if $p(1) = p(2)$: decide $y_u = 1$ if

$$\log\left(\frac{\pi_1}{\pi_2}\right) + \log\left(\frac{\text{DoI } \pi_1 + \pi_2}{\pi_1 + \text{DoI } \pi_2}\right) + \log(S(\mathbf{x}_u)) + \sum_{v \in \mathcal{N}(u)} \log\left(\frac{\text{DoI } \pi_2 + \pi_1 S(\mathbf{x}_v)}{\pi_2 + \text{DoI } \pi_1 S(\mathbf{x}_v)}\right) > 0$$

- For a purely homophilic graph, decide $y_u = 1$ if

$$\log\left(\frac{\pi_1}{\pi_2}\right) + (1 - |\mathcal{N}(u)|)\log\left(\frac{p(2)\pi_2}{p(1)\pi_1}\right) + \sum_{v \in \mathcal{N}(u) \cup \{u\}} \log(S(\mathbf{x}_v)) > 0$$

- Linear aggregation if $\log(S(\mathbf{x}_v))$ is a linear function of \mathbf{x}_v

- Cases where GAB is a GNN, i.e. decide $y_u = 1$ when

$$\omega_0 + \boldsymbol{\omega}_1^T \sum_{v \in \mathcal{N}(u) \cup \{u\}} \mathbf{x}_v > 0$$

- Pure homophily and $D_k(\cdot) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$:

$$\omega_0 = \log\left(\frac{\pi_2}{\pi_1}\right) + (1 + |\mathcal{N}(u)|)(\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 - \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1), \quad \boldsymbol{\omega}_1^T = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}$$

- Pure homophily and independent binary features $\left(\Pr(x_{v,f} = 1 | y_u = k) = \alpha_f^{(k)}\right)$

$$\omega_0 = \log\left(\frac{\pi_2}{\pi_1}\right) + F \sum_{f=1}^F \log\left(\frac{1 - \alpha_f^{(2)}}{1 - \alpha_f^{(1)}}\right), \quad \omega_{1,f} = -\log\left(\frac{\alpha_f^{(1)}}{\alpha_f^{(2)}} \frac{1 - \alpha_f^{(2)}}{1 - \alpha_f^{(1)}}\right)$$

Outline

Introduction

- Attributed graphs
- ML tasks on graphs
- Very brief introduction to ML

Graph embedding

- Direct encoding
- Graph neural networks (GNN)

Node & graph Classification

- Homophily
- Collective classification
- GNN-based classifiers
- Graph-assisted Bayesian classifiers

Very brief introduction to knowledge graphs

- KG embedding
- KG completion

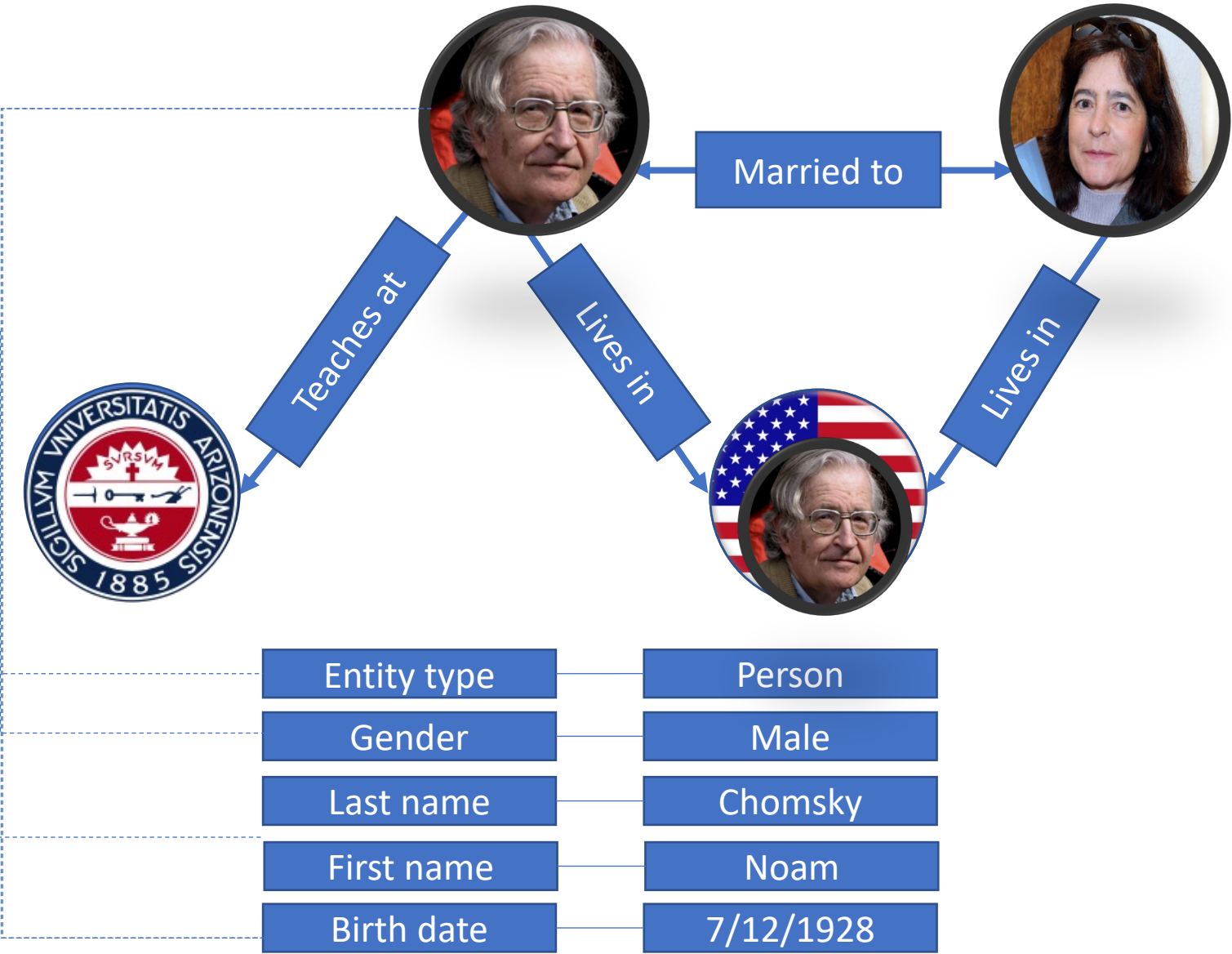
- In a multirelational graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, edges are defined as triples $e = (h, r, t)$ indicating the presence of a particular relation $r \in \mathcal{R}$ between two nodes, the head (h) and the tail (t).
- Multirelational graphs are also known as knowledge graphs: tuple $e = (h, r, t)$ specifies a particular « fact » between two entities

e.g. $r = \text{TREATS}$, h = a particular drug, t = a particular disease

- Adjacency tensor $\mathcal{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{R}| \times |\mathcal{V}|}$, i.e.:

$$\mathcal{A}[h, r, t] = 1 \text{ if } (h, r, t) \in \mathcal{E} \text{ \& } 0 \text{ otherwise}$$

- Practical graphs are usually sparse, i.e. $|\mathcal{E}| \ll |\mathcal{V}|^2 |\mathcal{R}|$
- A knowledge graph (KG) can be built manually or using automatic information extraction methods on some source text (e.g. Wikipedia, scientific literature).



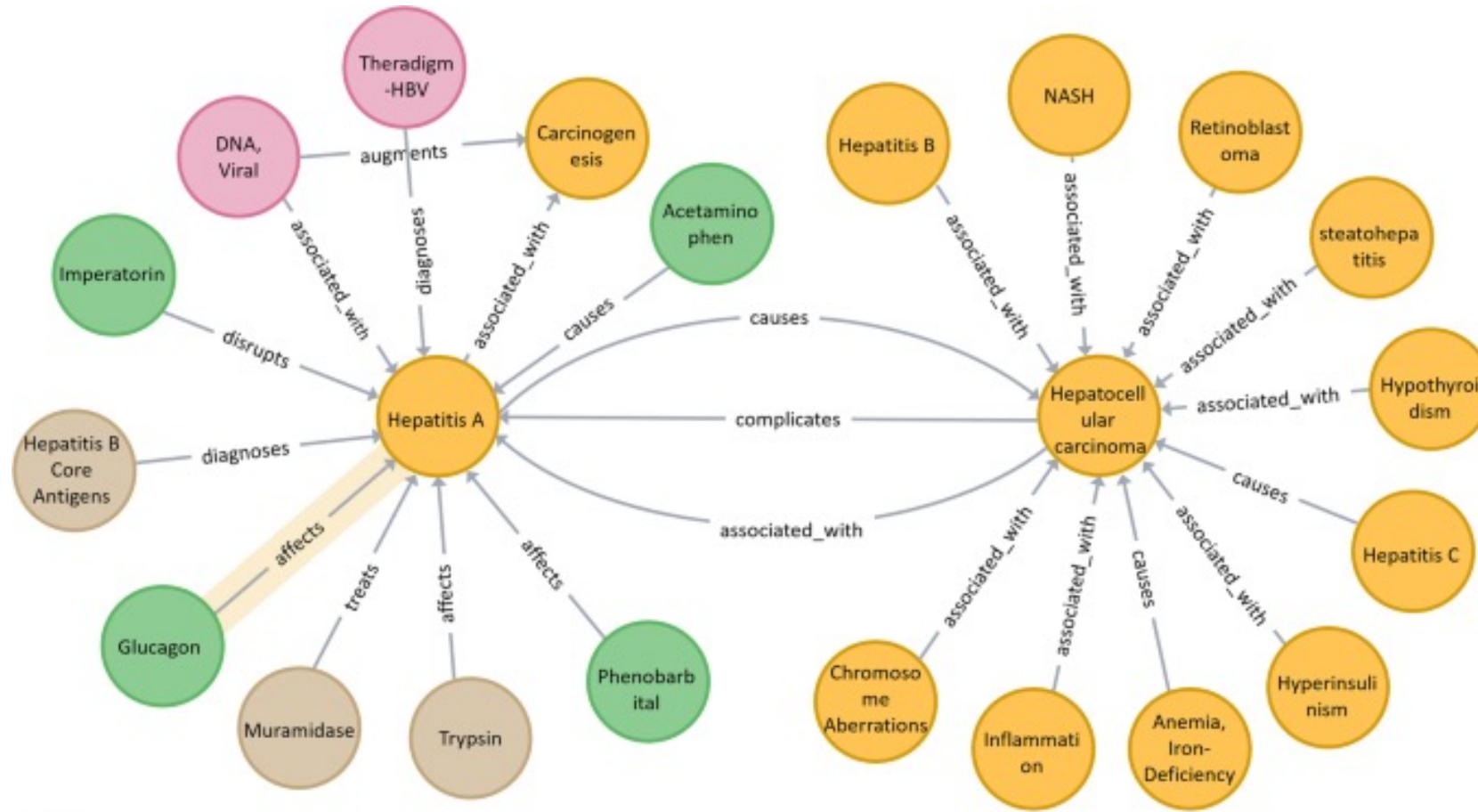
Chomsky's knowledge graph project

"Chomsky's work is tremendously valuable to people across many disciplines. Our goal is to make Chomsky's work searchable in the context of topics and concepts, readable in excerpts, and easily available to journalists, scientists, technologists, students, philosophers, and historians as well as the general public." F. Davis (Exec. Dir. of project)



The KG will link to over **1,000 articles** and over **100 books** that Chomsky has authored about linguistics, mass media, politics and war, **Hundreds of media interviews** (TV, print and online), and more than a dozen movies

KG for hepatocellular carcinoma: 5028 entities and 13,296 triples: 1328 drugs, 1849 proteins, 1403 diseases, 160 cells, 140 DNAs, 54 phenotypic abnormalities, 50 genes, 35 therapeutic-techniques and 9 RNAs [Li et al., 2020]

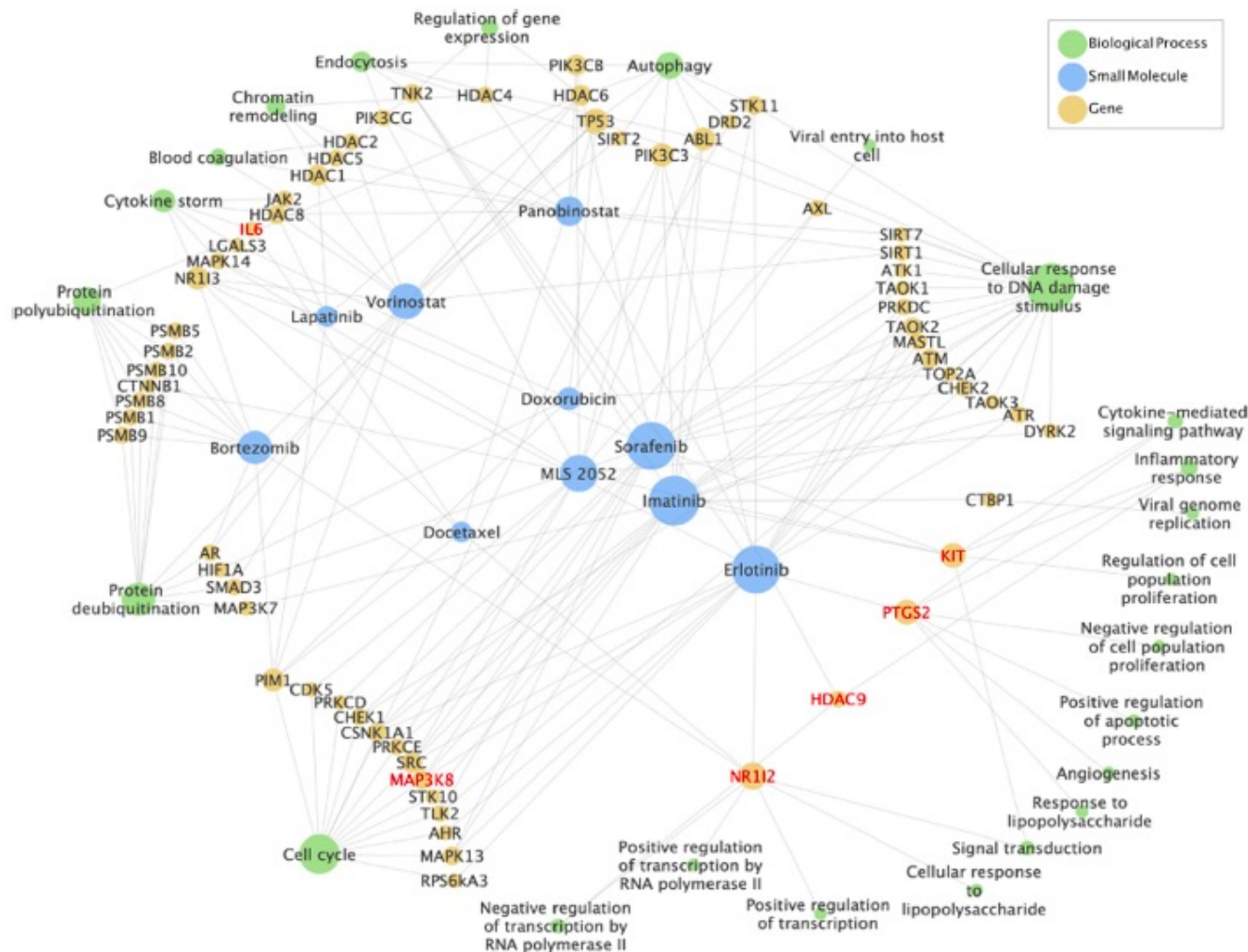


Parts of KGHC between hepatocellular carcinoma and Hepatitis A

- KG are incomplete, i.e. ‘facts’ are missing. Statistical/ML models can be used to infer missing facts.
- Formally: given two nodes u and v of a multirelational graph, the goal of link prediction is to estimate the likelihood of the existence of a particular relation, $r \in \mathcal{R}$, between the two nodes

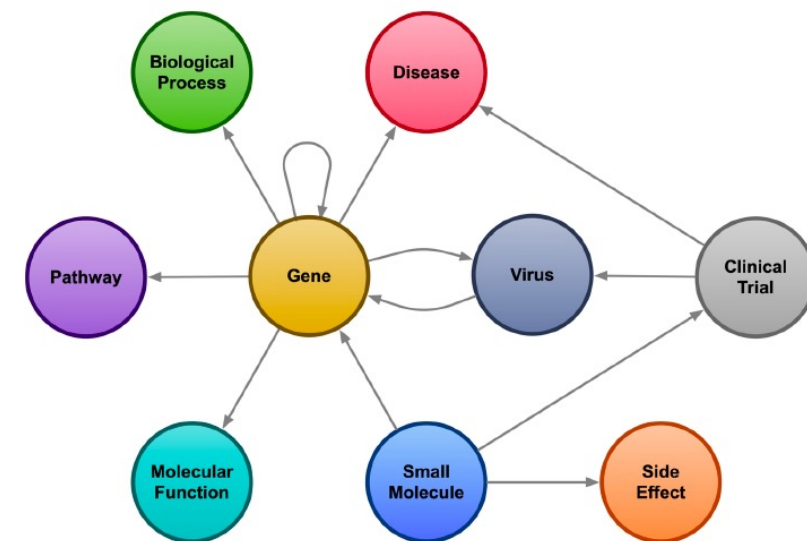
Knowledge graph completion

Drug repurposing



Network diagram showing the connections of the top 10 scoring drugs from the results. Gene names in red represent genes that have a greater than 2-fold change in expression in response to SARS-CoV-2 infection. The size of the node corresponds to the number of connections to other nodes.

<https://pubs.acs.org/doi/10.1021/acs.jcim.1c00642>



Top 10 Drug Repurposing Candidates

Drug Name	Drug Class
vorinostat	HDAC inhibitors
bortezomib	protease inhibitors
doxorubicin	DNA metabolism-related
sorafenib	kinase inhibitors
erlotinib	kinase inhibitors
lapatinib	kinase inhibitors
docetaxel	microtubule-regulating agents
MLS 2052	kinase inhibitors
panobinostat	HDAC inhibitors
imatinib	kinase inhibitors

- Rule-based approaches e.g. AMIE3 [Lajus et al., 2020]
- Embedding-based approaches (our focus)
- Hybrid models, e.g. RLvLR [Omran et al., 2018]

- **Objective**: given the embeddings **h** and **t** of two nodes (head and tail), the goal of the decoder is to reconstruct the relationship between these nodes
- **Challenge**: presence of different relationships (types of edges)
- **Solution**: use a **multi-relational decoder**:

$$\text{DEC}_r: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+ \quad \text{DEC}_r(\mathbf{z}_h, \mathbf{z}_t) \triangleq f_r(\mathbf{z}_h, \mathbf{z}_t)$$

- $f_r(\mathbf{z}_h, \mathbf{z}_t)$ is often referred to as **score function**
- Encoding approaches: **direct methods** (lookup table) and **GNN-based methods**
- Embedding methods differ in the used score function and loss function.

- Reconstruction loss:

$$\mathcal{L} = \sum_{h \in \mathcal{V}} \sum_{t \in \mathcal{V}} \sum_{r \in \mathcal{R}} (f_r(\mathbf{z}_h, \mathbf{z}_t) - \mathcal{A}[h, r, t])^2$$

- Example: RESCAL method

$$f_r(\mathbf{z}_h, \mathbf{z}_t) = \mathbf{z}_h^T \mathbf{R}_r \mathbf{z}_t$$

$\mathbf{R}_r \in \mathbb{R}^{d \times d}, r \in \mathcal{R}$, are learnable matrices

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \|\mathbf{Z}^T \mathbf{R}_r \mathbf{Z} - \mathbf{A}_r\|_F^2$$

\mathbf{A}_r : adjacency matrix for relation r

Similarity measure is almost always directly based on the adjacency tensor because:

- it is difficult to define higher-order neighborhood relationships in multi-relational graphs
- generally multi-relational embedding are designed for relation prediction

- High computational cost:
 - The nested sums require $O(|\mathcal{V}|^2 |\mathcal{R}|)$ operations; prohibitive for large graphs. Ideally, the computational complexity of the loss function should be $O(|\mathcal{E}|)$ where $|\mathcal{E}| \ll |\mathcal{V}|^2 |\mathcal{R}|$ due to sparsity
 - The optimization is generally framed as a tensor factorization
- Reconstruction is framed as a regression task:
 - The adjacency tensor, to be reconstructed, has binary values
 - So, instead of MSE, a classification loss is more appropriate

- Negative sampling is a process that corrupts head and tail entities to generate negative examples
- Cross-entropy with negative sampling:

$$\mathcal{L} = \sum_{(h,r,t) \in \mathcal{E}} \left[-\log \left(\sigma \left(f_r(\mathbf{z}_h, \mathbf{z}_t) \right) \right) - \sum_{t_n \in P_{n,h}(\mathcal{V})} -\log \left(1 - \sigma \left(f_r(\mathbf{z}_h, \mathbf{z}_{t_n}) \right) \right) \right]$$

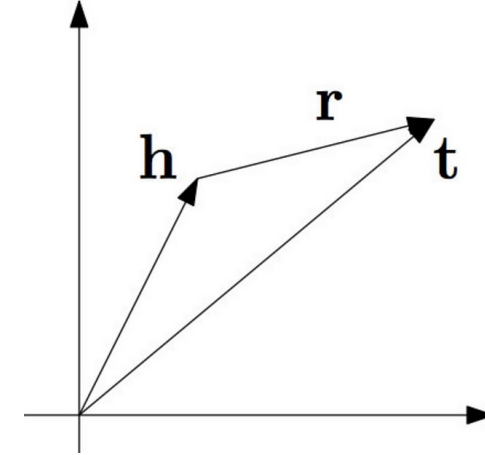
- h : head, t : head; $\sigma(\cdot)$: logistic function, scores in $[0,1]$, interpreted as probabilities
 - $\sigma(\text{DEC}(\mathbf{z}_h, r, \mathbf{z}_t))$: likelihood that (h, r, t) is a ‘fact’
 - $P_{n,h}(\mathcal{V})$: negative sampling distribution over the set of nodes \mathcal{V} (might depend on h)
- Max-margin loss with negative sampling:

$$\mathcal{L} = \sum_{(h,r,t) \in \mathcal{E}} \sum_{t_n \in P_{n,h}(\mathcal{V})} \max(0, -f_r(\mathbf{z}_h, \mathbf{z}_t) + f_r(\mathbf{z}_h, \mathbf{z}_{t_n}) + \Delta)$$

- Closed world assumption: "all unseen facts are false facts."
- Basic negative sampling:
 - Step 1: sample a triple (h, r, t)
 - Step 2: randomly change either the head and/or the tail N times
- Advanced negative sampling:
 - Constrained sampling: given a known schema, restrict the types of negative entities to be of certain types
 - Generative sampling [Sun et al., 2019]: learns adaptively the sampling probabilities for the entities

- Let's first simplify notations: $\mathbf{z}_h \rightarrow \mathbf{h}$, $\mathbf{z}_t \rightarrow \mathbf{t}$
- TransE [Bordes et al., 2013]:

$$f_r(\mathbf{h}, \mathbf{t}) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|$$

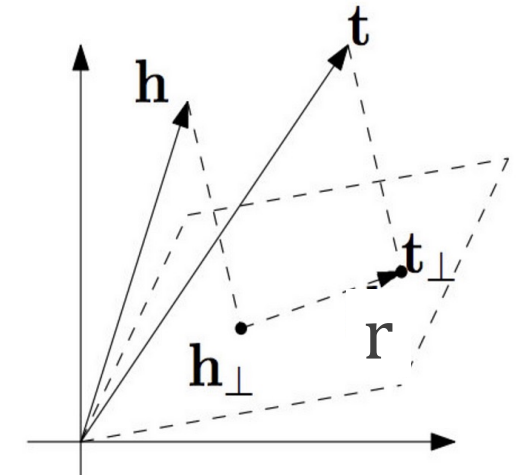


This distance-based scoring function measures the plausibility of facts by calculating the distance between $\mathbf{h} + \mathbf{r}$ and \mathbf{t} . Likelihood is high when $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$

- Issues with TransE:
 - Cannot model symmetric relations
 - Can only model one-to-one relations, and not one-to-many/many-to-one relation; for example with triples (EiffelTower, LocatedIn, Paris) and (HEC Paris, LocatedIn, Paris), embeddings vectors of 'EiffelTower' and 'HEC Paris' will be very close !

- TransH (Translating on Hyperplanes):

$$f_r(\mathbf{h}, \mathbf{t}) = -\|\mathbf{h} - \mathbf{w}_r^T \mathbf{h} \mathbf{w}_r + \mathbf{r} - (\mathbf{t} - \mathbf{w}_r^T \mathbf{t} \mathbf{w}_r)\|$$



Basic idea: interpreting a relation as a translating operation on a hyperplane.

TransH deals with the one-to-many/many-to-one/many-to-many relations.

- TransX:

$$f_r(\mathbf{h}, \mathbf{t}) = -\|g_{1,r}(\mathbf{h}) + \mathbf{r} - g_{2,r}(\mathbf{t})\|$$

- **Analogy** (Translating on Hyperplanes) [Liu et al., 2017] :

$$f_r(\mathbf{h}, \mathbf{t}) = \mathbf{h}^T \mathbf{M}_r \mathbf{t}$$

It measures the plausibility of facts by semantic matching; $\mathbf{h}^T \mathbf{M}_r \approx \mathbf{t}$ implies high plausibility

- **DistMult** [Yang et al., 2015]:

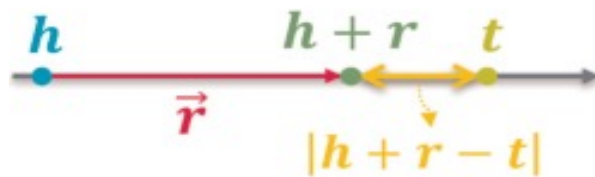
$$f_r(\mathbf{h}, \mathbf{t}) = \sum_i [\mathbf{h} \odot \mathbf{r} \odot \mathbf{t}]_i$$

- RotatE [Sun et al., 2019] :

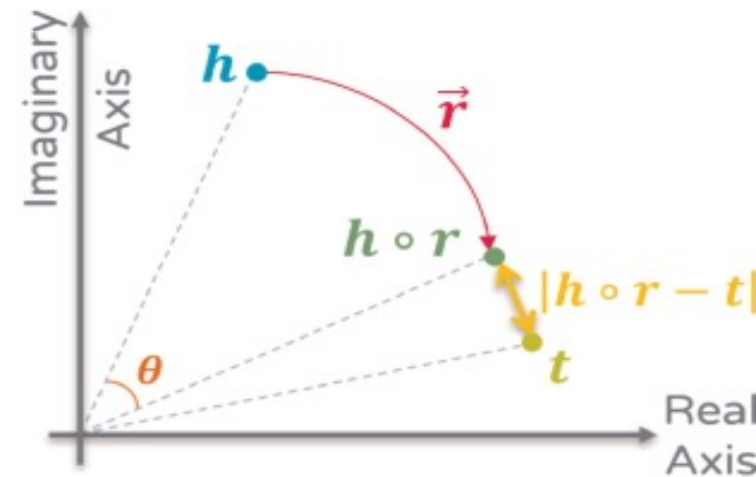
$$f_r(\mathbf{h}, \mathbf{t}) = -\|\mathbf{h} \circ \mathbf{r} - \mathbf{t}\|$$

$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$; $r(i) = e^{i\theta_{r,i}}, \forall i$; rotation in the complex plane

Can model and infer various relation patterns including symmetry/antisymmetry, inversion, and composition



(a) TransE



(b) RotatE

- Representational ability of a score function refers to the different relational patterns that it can model. Most common relational patterns are:
 - **Symmetry**: $r(x, y) \Rightarrow r(y, x)$
 - **Antisymmetry**: $r(x, y) \Rightarrow \neg r(y, x)$
 - **Inversion**: r_1 is inverse to r_2 if $r_1(x, y) \Rightarrow r_2(y, x)$
 - **Composition**: r_1 is composed of r_2 and r_3 if $r_2(x, y) \wedge r_3(y, z) \Rightarrow r_1(x, z)$

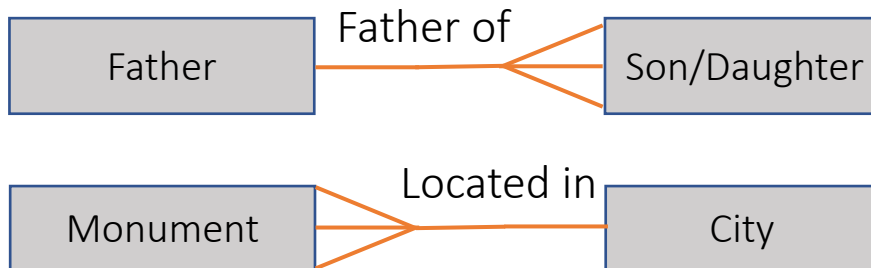
Model	Symmetry	Antisymmetry	Inversion	Composition
SE[Bordes et al., 2011]	✗	✗	✗	✗
TransE [Bordes et al., 2013]	✗	✓	✓	✓
TransX (i.e. TransE Variants)	✓	✓	✗	✗
DistMult [Yang et al., 2014]	✓	✗	✗	✗
Complex [Trouillon et al., 2016]	✓	✓	✓	✗
ROTATE [Sun et al., 2019]	✓	✓	✓	✓

■ There are main four types of relationships:

- **One-to-One**: can only relate one head and one tail



- **One-to-Many (Many-to-One)**: can relate one head (tail) and many tails (heads)



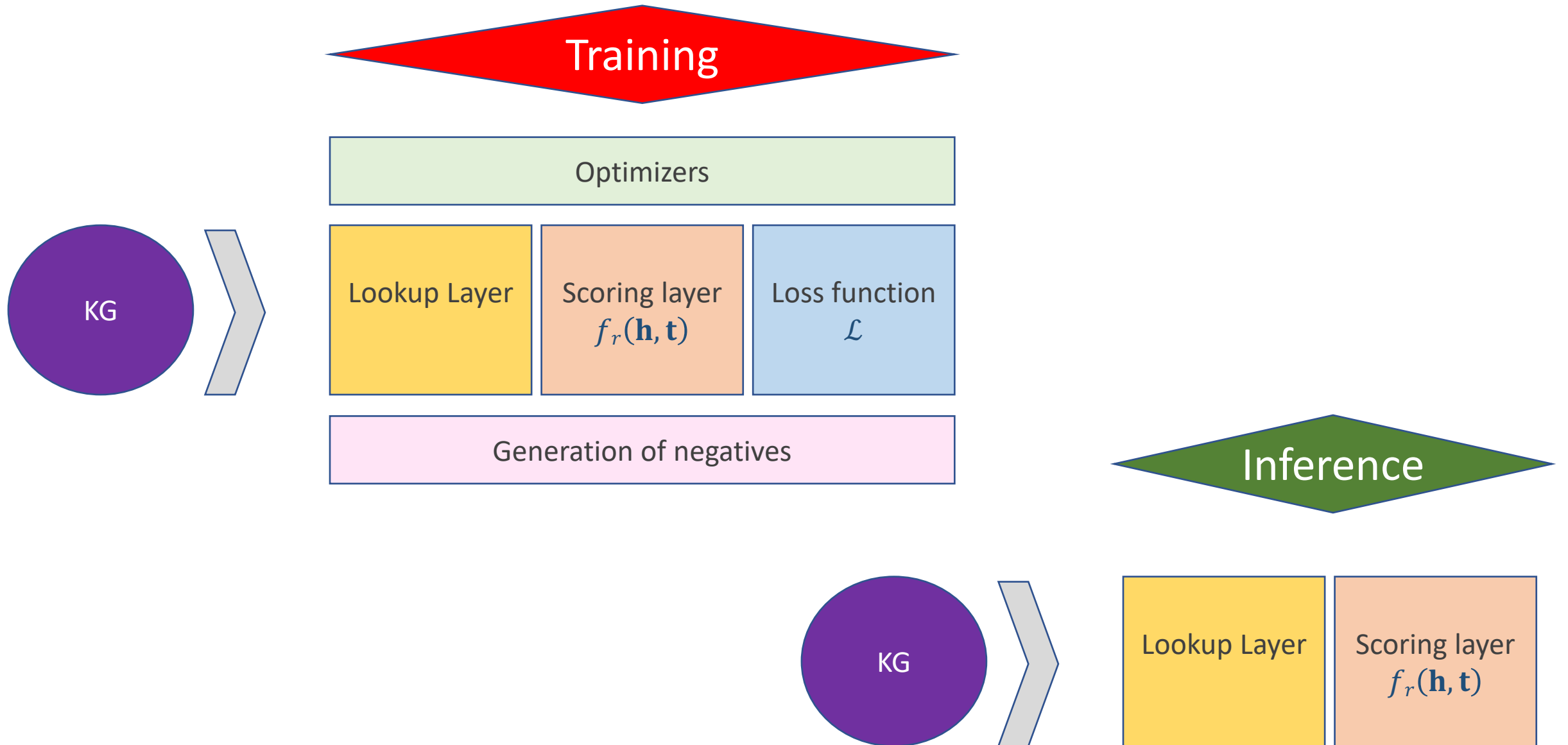
- **Many-to-Many**: can relate many heads and many tails



Relation Category	1-to-1	1-to-N	N-to-1	N-to-N
Tasks	Prediction Head (Hits@10)			
TransE	.437	.657	.182	.472
TransH (bern)	.668	.876	.287	.645
KG2E_KL (bern)	.923	.946	.660	.696
TransE	.894	.972	.567	.880
ComplEx	.939	.969	.692	.893
RotatE	.922	.967	.602	.893
Tasks	Prediction Head (MRR)			
TransE	.701	.912	.424	.737
ComplEx	.832	.914	.543	.787
RotatE	.878	.934	.465	.803

Experimental results on FB15k dataset by relation category

[Liu et al., 2017]



- NSF Loss: we focus on Barlow Twins (BT) and Uniformity-Alignment (UA) losses.
- BT loss:

$$\mathcal{L}_{\text{BT}}(\mathbf{X}, \mathbf{Y}) = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2$$

with
$$C_{ij} = \frac{\sum_b x_{b,i} x_{b,j}}{\sqrt{\sum_b x_{b,i}^2 \sum_b x_{b,j}^2}}$$

- UA loss:

$$\mathcal{L}_{\text{AU}}(\mathbf{X}, \mathbf{Y}) = I_{\text{align}}(\mathbf{X}, \mathbf{Y}) + I_{\text{unif}}(\mathbf{X}) + I_{\text{unif}}(\mathbf{Y})$$

- with:

$$I_{\text{align}}(\mathbf{X}, \mathbf{Y}) = -\frac{1}{B} \sum_{b=1}^B \|\mathbf{x}_b - \mathbf{y}_b\|_2^\alpha$$

$$I_{\text{unif}}(\mathbf{X}) = \log \left(\frac{1}{N^2} \sum_{i=1}^N \sum_{j \neq i}^N \exp \left(-\tau \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \right) \right)$$

- Training objective of NSF-KG [Bahaj & Ghogho, 2022]:

$$\mathcal{L} = \mathcal{L}_{\text{NSF}}(\hat{\mathbf{H}}, \mathbf{T}) + \mathcal{L}_{\text{NSF}}(\hat{\mathbf{T}}, \mathbf{H})$$

$\mathbf{H} = [\mathbf{h}_b]_{b=1}^B \in \mathbb{R}^{B \times d}$, $\mathbf{T} = [\mathbf{t}_b]_{b=1}^B \in \mathbb{R}^{B \times d}$, $\mathbf{R} = [\mathbf{r}_b]_{b=1}^B \in \mathbb{R}^{B \times d}$; \mathcal{B} : batch of triplets
 $\hat{\mathbf{H}}$ and $\hat{\mathbf{T}}$ are defined as:

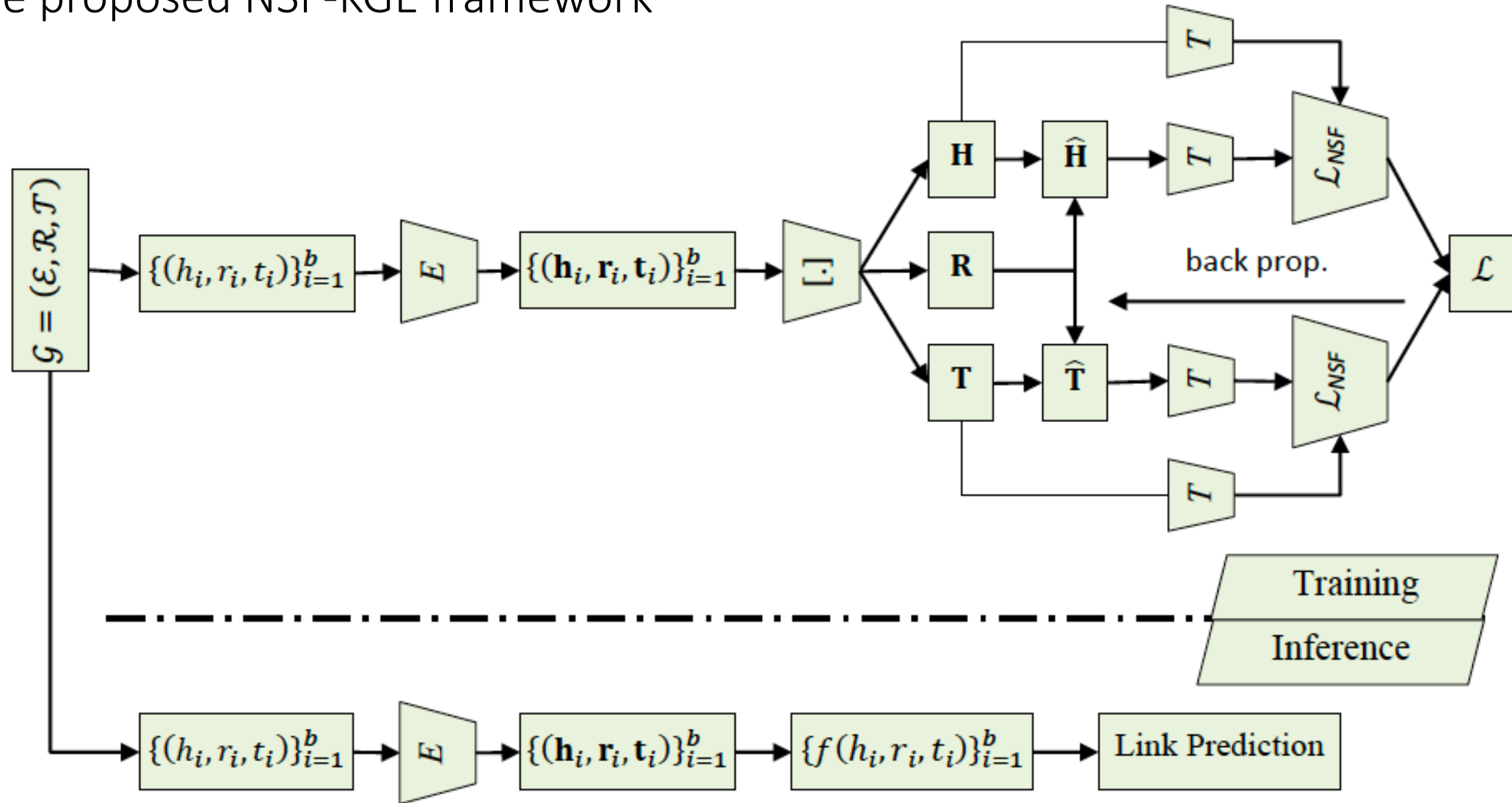
- NSF-TransE:

$$\hat{\mathbf{H}} = \mathbf{H} + \mathbf{R}, \quad \hat{\mathbf{T}} = \mathbf{T} - \mathbf{R}$$

- NSF-DistMult:

$$\hat{\mathbf{H}} = \mathbf{H} \odot \mathbf{R}, \quad \hat{\mathbf{T}} = \mathbf{T} \odot \mathbf{R}$$

- The proposed NSF-KGE framework



Loss	Model	BN Layer	FB15K					WN18				
			Hit@1	Hit@10	Hit@3	MR	MRR	Hit@1	Hit@10	Hit@3	MR	MRR
NS-Based	TransE+	DistMult \$.471		125			.892		251	
				.577			.35		.942			.83
AU	Distmult	w/ BN	.3279	.6233	.4947	204.31	.4347	.7027	.9315	.8992	713.10	<u>.8039</u>
		w/o BN	.2094	.4443	.3291	460.67	.2930	.6498	.9221	.8741	709.89	.7672
	TransE	w/ BN	.2688	.5861	.4469	<u>147.63</u>	.3834	.1129	.6183	.3699	<u>193.65</u>	.2845
		w/o BN	.2295	.5168	.4035	247.37	.3378	.1943	.6234	.3868	159.07	.3323
BT	Distmult	w/ BN	.3044	<u>.6058</u>	.4727	179.53	<u>.4132</u>	.6975	<u>.9329</u>	.8964	785.02	.7999
		w/o BN	.2318	.4942	.3734	311.51	.3265	.6905	<u>.9329</u>	.8936	626.86	.7955
	TransE	w/ BN	.2445	.5646	.4256	158.68	.3610	.1699	.6053	.3610	361.91	.3084
		w/o BN	.2371	.5089	.3919	256.94	.3367	.0721	.7859	.4834	460.85	.3149

- Challenge: augmenting message passing to include multiple types of relations
- Example: [Schlichtkrull et al., 2018] augmented the aggregation function to accommodate multiple relation types by specifying a separate transformation matrix per relation type:

$$m_{\mathcal{N}(u)} = \sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{N}_r(u)} \frac{\mathbf{W}_r \mathbf{h}_v}{f_n(\mathcal{N}_u, \mathcal{N}_v)} \quad \text{Relational Graph Convolutional Network (RGCN)}$$

- Reduced complexity RGCN: sharing basis matrices $\{\mathbf{B}_i, i = 1, \dots, b\}$

$$\mathbf{W}_r = \sum_{i=1}^b \alpha_{i,r} \mathbf{B}_i \quad m_{\mathcal{N}(u)} = \sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{N}_r(u)} \frac{\alpha_r \times_1 \mathcal{B} \times_2 \mathbf{h}_v}{f_n(\mathcal{N}_u, \mathcal{N}_v)}$$

$\mathcal{B} = (\mathbf{B}_1, \dots, \mathbf{B}_b)$ is a tensor \times_i denotes a tensor product along mode i

- Further details in [Schlichtkrull et al., 2018]
- Other GNN-based embedding methods for KG are available in the literature...

References

- [Abate et al., 2023] C. Abate, S. Decherchi, A. Cavalli, Graph neural networks for conditional de novo drug design. WIREs Computational Molecular Science, 2023
- [Balogh et al., 2022] Balogh, O. M. et al. Efficient link prediction in the protein–protein interaction network using topological information in a generative adversarial network machine learning model. BMC Bioinformatics, 2022
- [Stokes et al., Cell’20] Stokes, J. M., et al. A Deep Learning Approach to Antibiotic Discovery. Cell, vol. 180(4), 2020
- [Zhu et al., 2009] X. Zhu and A. B. Goldberg, ‘Introduction to Semi-Supervised Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2009.
- [Hastie et al., 2009] T. Hastie, R. Tibshirani, and J. Friedman. Overview of supervised learning. In The elements of statistical learning, pages 9–41. Springer, 2009.
- [Jing et al., 2020] Longlong Jing and Yingli Tian, Self-supervised visual feature learning with deep neural networks: A survey, IEEE TPAMI, 2020
- [Chen et al, 2020] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representation, ICML 2020
- [Konkle et al., 2022] Talia Konkle & George A. Alvarez A self-supervised domain-general learning framework for human ventral stream representation, Nature Communications volume 13, Article number: 491
- [Hamilton, 2020] L. W Hamilton. Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning

References

- [Belkin et al., 2002] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. NeurIPS 2002
- [Ahmed et al., 2013] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A.J. Smola. Distributed large-scale natural graph Factorization. WWW 2013
- [Cao et al., 2015] S. Cao et al. Learning graph representation with global structure information. KDD 2015
- [Ou et al., 2016] M. Ou et al. Asymmetric transitivity preserving graph embedding. KDD 2016
- [B. Perozzi, 2014] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. KDD 2014
- [Grover et al., 2016] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. KDD 2016.
- [Cao et al., 2016] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. AAAI 2016
- [Wang et al., 2016] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. KDD 2016
- [Qi et al., 2017] C.R. Qi, H. Su, K. Mo, and L.J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. CVPR 2017.
- [Morris et al., 2019] C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In AAAI, 2019.
- [Xu et al., 2019] K. Xu, W. Hu, J. Leskovec, S. Jegelka. How Powerful are Graph Neural Networks? ICLR 2019
- [Kipf and Welling, 2016] T.N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. ICLR 2016.
- [Dai et al., 2016] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In ICML, 2016.

References

- [Zaheer et al., 2017] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets. NeurIPS 2017.
- [Murphy et al., 2018] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. ICLR 2018
- [Bahdanau et al., 2015] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. ICLR 2015.
- [Veličković et al. 2018] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. ICLR 2018.
- [Xu et al., 2018] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. ICML 2018.
- [Hamilton et al., 2017a] W. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull., 2017.
- [Vinyals et al., 2015] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In ICLR, 2015.
- [Pham et al., 2017] Pham, T. Tran, D. Phung, and S. Venkatesh. Column networks for collective classification. AAAI 2017.
- [Li et al., 2015] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. ICLR 2015.
- [Selsam et al., 2019] D. Selsam, M. Lamm, B. Bollig, P. Liang, L. de Moura, and D. Dill. Learning a SAT solver from single-bit supervision. ICLR 2019.

References

- [Murphy et al., 2019] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Relational pooling for graph representations. In ICML, 2019.
- [Maron et al., 2019] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In NeurIPS, 2019.
- [Wu et al., 2019] F. Wu, T. Zhang, C. Souza, A. and Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In ICML, 2019.
- [Klicpera et al., 2019] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In ICLR, 2019.
- [Chen et al., 2020] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton A Simple Framework for Contrastive Learning of Visual Representation, ICML 2020
- [Hafidi et al., 2021a] H. Hafidi, M. Ghogho, P. Ciblat, A. Swami, GraphCL: Contrastive Self-Supervised Learning of Graph Representations. arXiv:2007.08025v1
- [Hafidi et al., 2021b] H. Hafidi, M. Ghogho, P. Ciblat, A. Swami. Negative sampling strategies for contrastive self-supervised learning of graph representations. Signal Processing. Vol. 190, Jan 2022
- [Easley & Kleinberg, 2010] D. Easley and J Kleinberg. Networks, Crowds, and Markets: Reasoning about a Highly Connected World. Cambridge University Press, 2010
- [Gia et al., 2021] Junteng Jia, Cenk Baykal, Vamsi K. Potluru, Austin R. Benson, Graph Belief Propagation Networks. arXiv:2106.03033

References

- [Ghogho, 2023] M. Ghogho, « A Simple Theoretical Investigation of Graph Neural Networks for Node Classification » to appear on Arxiv
- [Hafidi et al., 2023] H. Hafidi, P. Ciblat, M. Ghogho, A. Swami. Graph-Assisted Bayesian Node Classifiers. IEEE Access, 2023
- [Lajus et al., 2020] Lajus, J., Galarraga, L., and Suchanek, F. (2020). Fast and exact rule mining with amie 3.
- [Omran et al., 2018] Omran, P. G., Wang, K., and Wang, Z. Scalable rule learning via learning representation. IJCAI 2018.
- [Bordes et al., 2013] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. Advances in neural information processing systems.
- [Sun et al., 2019] Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. arXiv preprint arXiv:1902.10197.
- [Trouillon et al., 2016] Trouillon, T., Welbl, J., Riedel, S., Gaussier, ÅLE., and Bouchard, G. (2016). Complex embeddings for simple link prediction. ICML 2016
- [Liu et al., 2017] Liu, H., Wu, Y., and Yang, Y. (2017). Analogical inference for multi-relational embeddings. In International conference on machine learning, PMLR, 2017
- [Yang et al., 2015] Yang, B., Yih, S. W.-t., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. ICLR 2015.

References

- [Yang et al., 2014] Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. arXiv preprint arXiv:1412.6575.
- [Bahaj & Ghogho, 2022] Bahaj, A., and M. Ghogho, M. (2022). KG-NSF: Knowledge Graph Completion with a Negative-Sample-Free Approach. arXiv:2207.14617v1
- [Schlichtkrull et al., 2018] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. ESWC 2018