Data Mining & Machine Learning

CS57300
Purdue University

April 3, 2018

Tasks Deep Learning Excels

- Image/Video Recognition

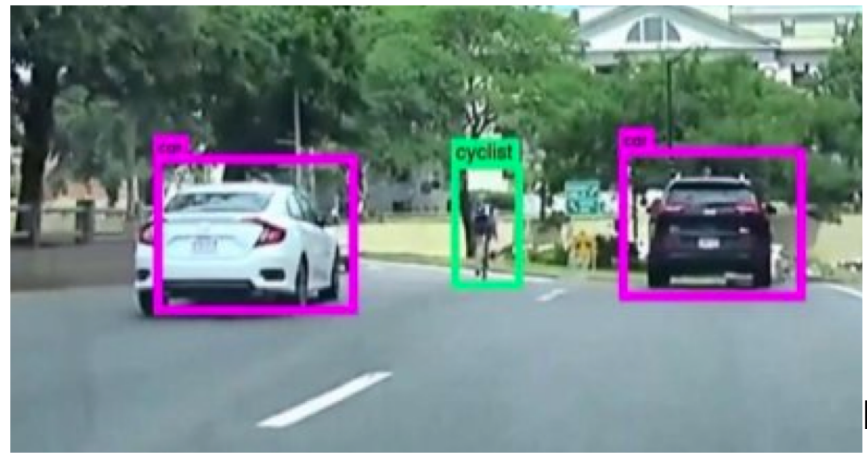


Image by Neurala

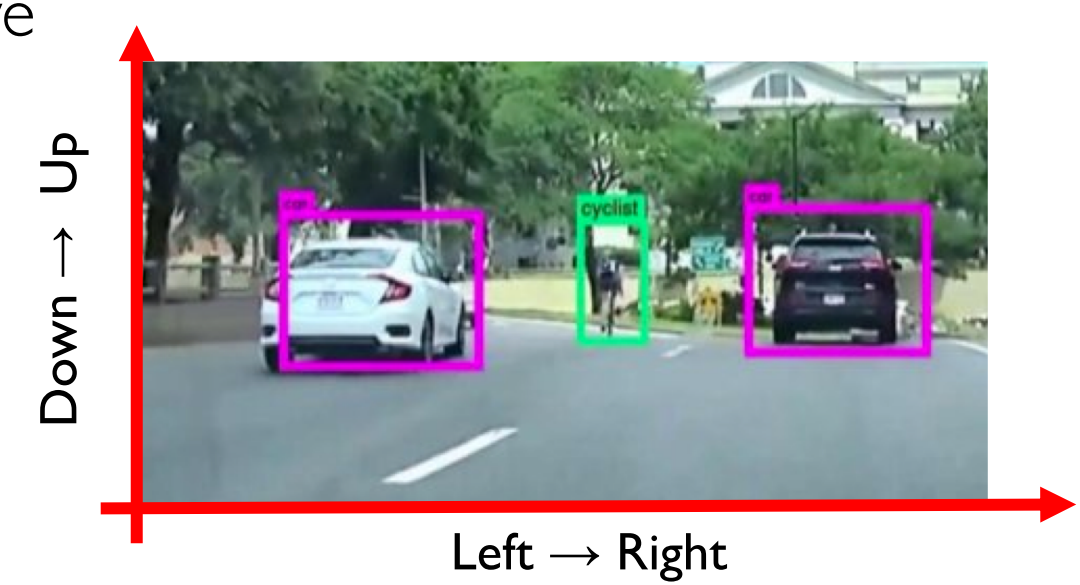
- Text Recognition / Prediction
  - The quick brown fox jumps over the lazy dog

- Speech Recognition

quick brown fox

Deep Learning Needs A Canonical Orientation

- Canonical orientation means that all training examples must have the same "orientation"



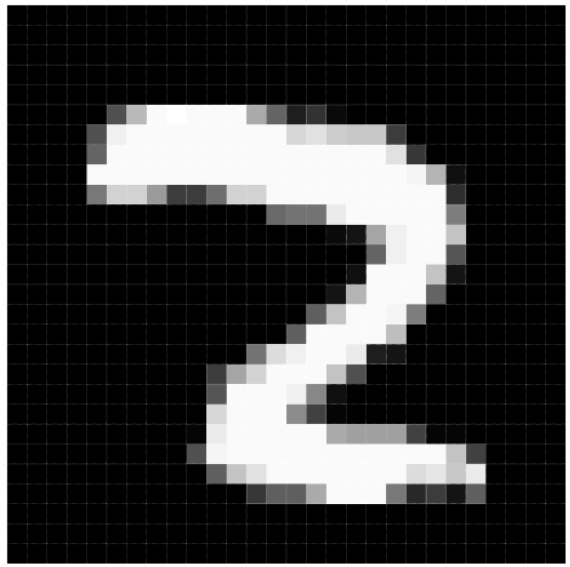
The quick brown fox jumps over the lazy dog

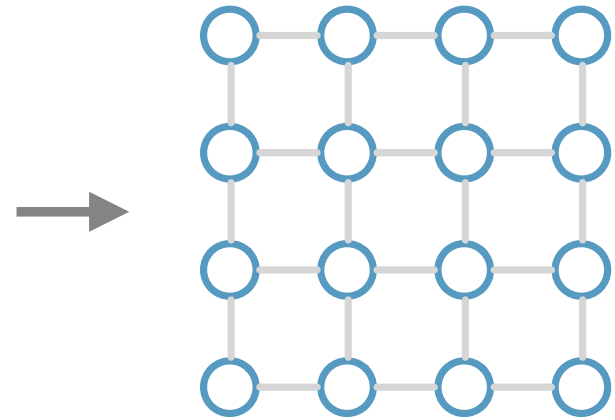
Left → Right

Sound wave:

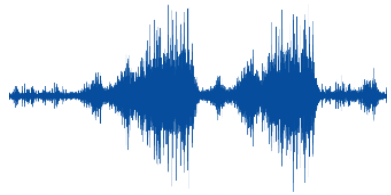
Past → Future

Tasks where Deep Learning Really Well





2D grid



1D grid

The lazy dog jumped over

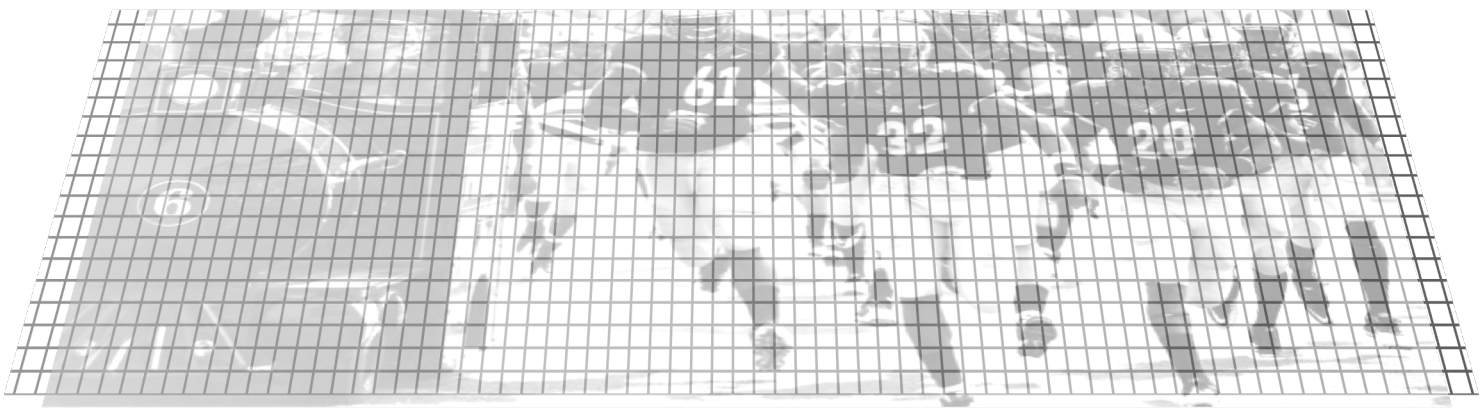
1D grid

Ack: Thomas Kipf

Role of Structure (in images)

- Feedforward networks use image as input as a vector

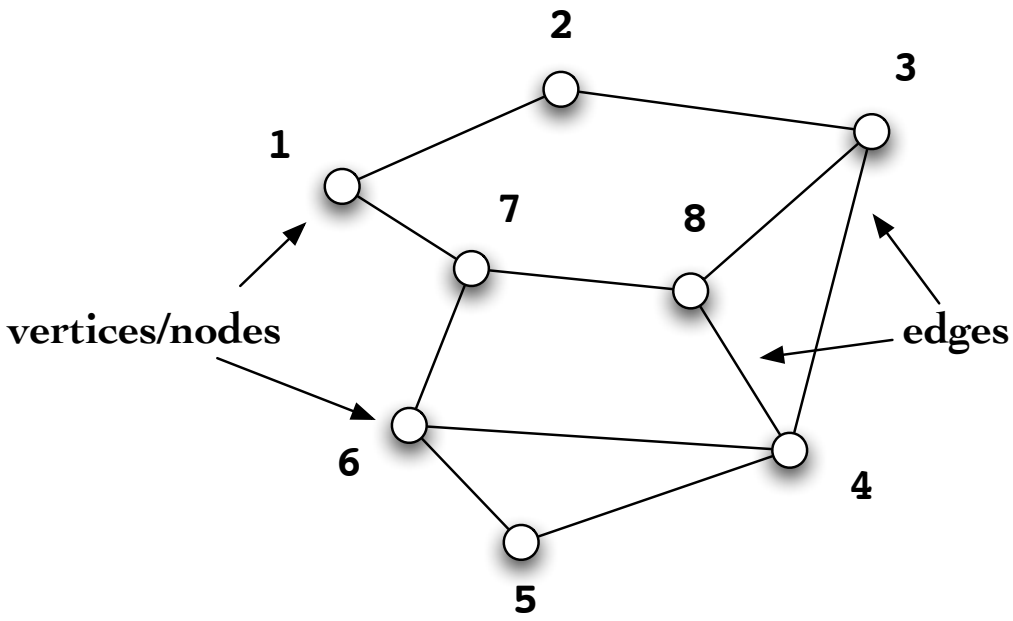
- From image to vector… hard to account for spatial correlations in the vector representation (which pixels are next to each other?)



- Images are grids, can we make neural networks that consider more complex structure?

What is a Graph?

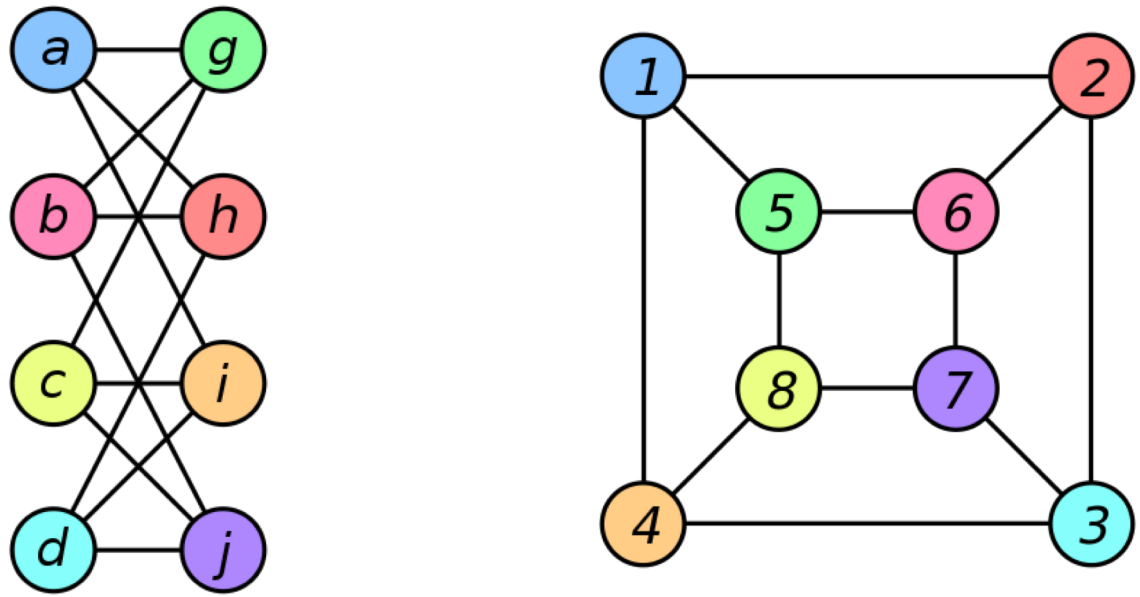
- A graph is a representation of a relationship



Adjacency matrix

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

Relationships Don't Have Simple Canonical "Orientations"

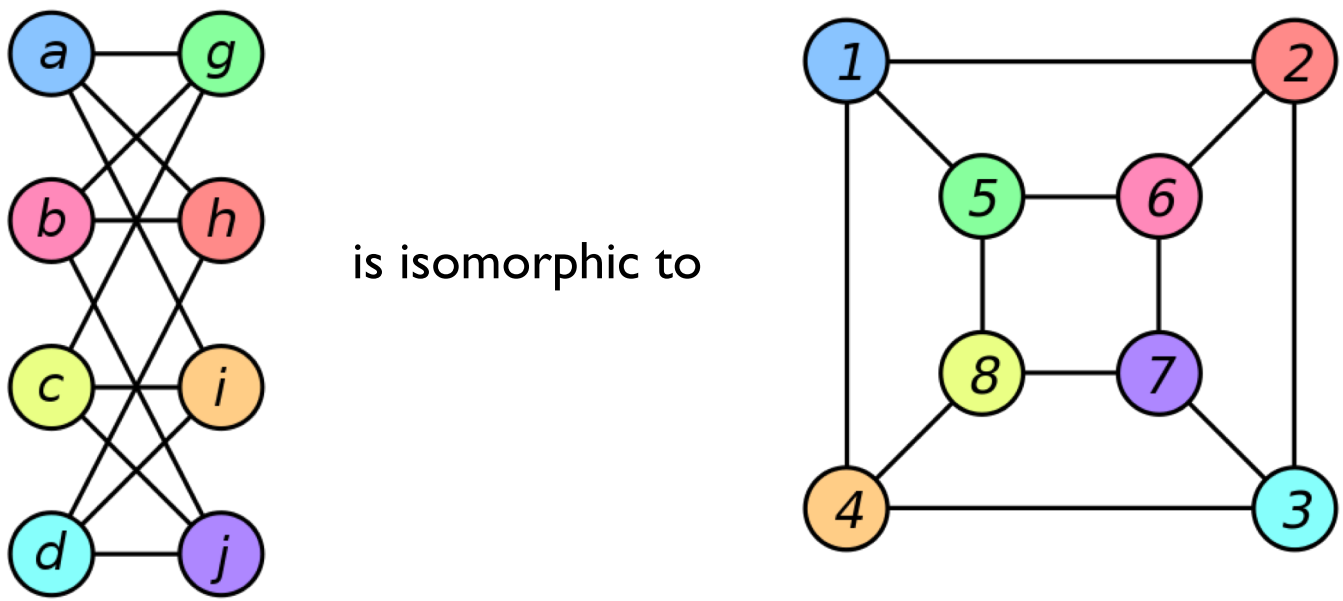


- These two graphs represent relationships in a dataset
- Are these two relationship graphs identical except for the node labels?
- No canonical orientation = deep learning will have difficulties

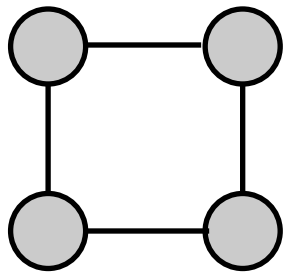
The Anatomy of a Solution: Making Deep Learning Understand Relationships

PURDUE UNIVERSITY

- Solutions must come from solving the graph "canonical orientation" problem
  - This is called "the graph isomorphism problem" in CS/Math

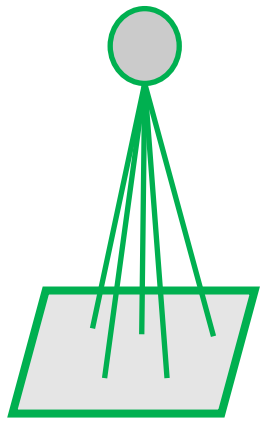


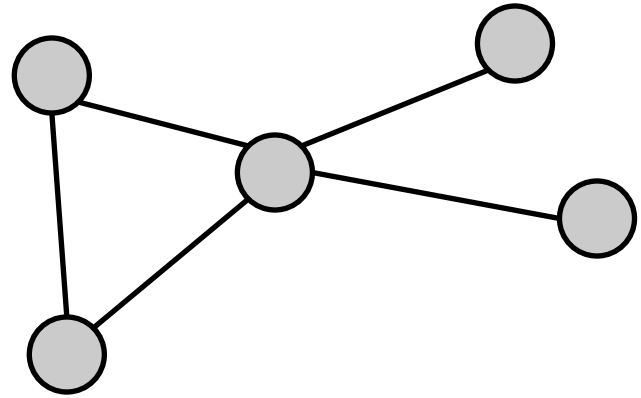
How to Deal with Structure on Neural Networks?



- Lattice

Conv. Net. Neuron



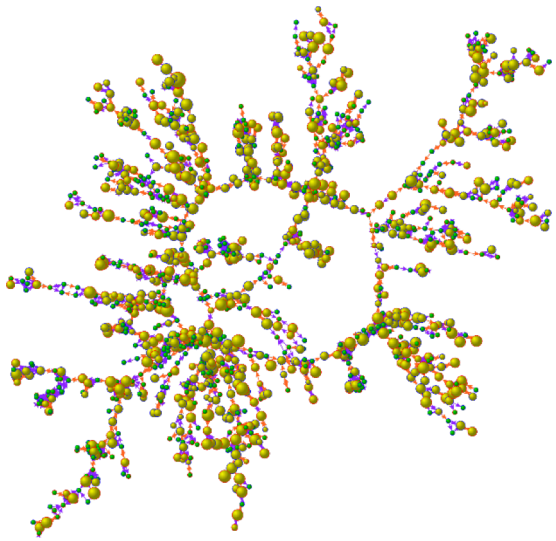


- Arbitrary graph

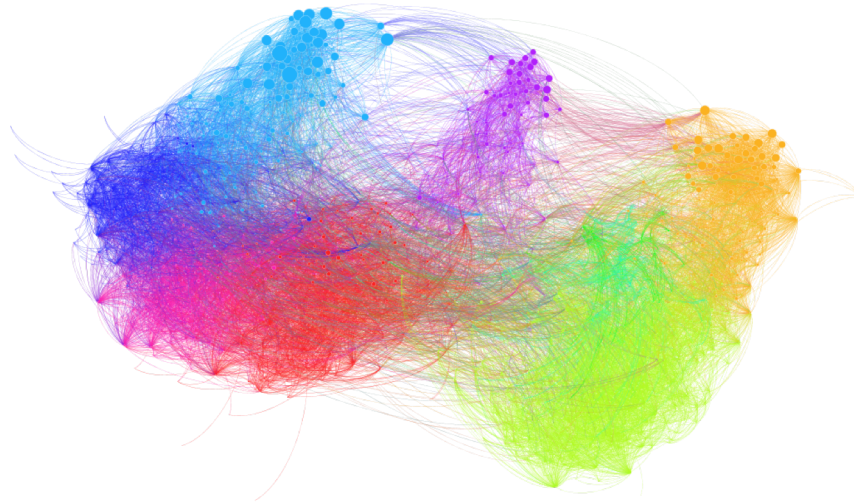
General Graph Neuron

What would be the graph equivalent?

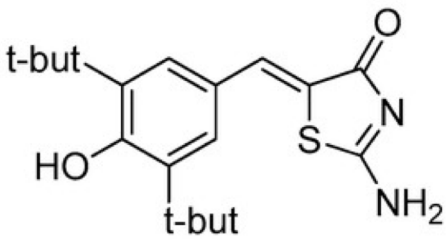
Examples of Graphs



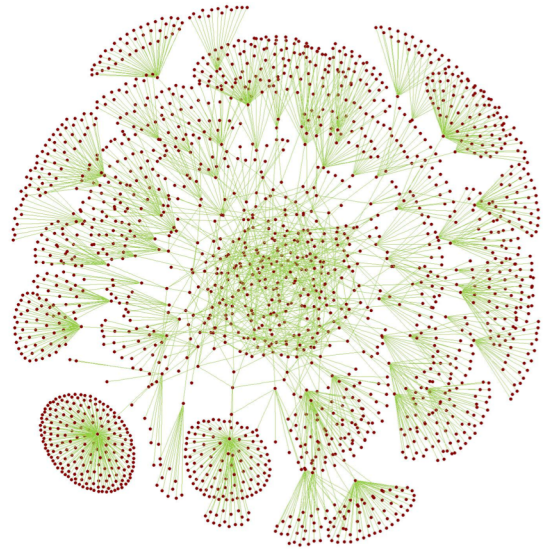
Biological Graphs



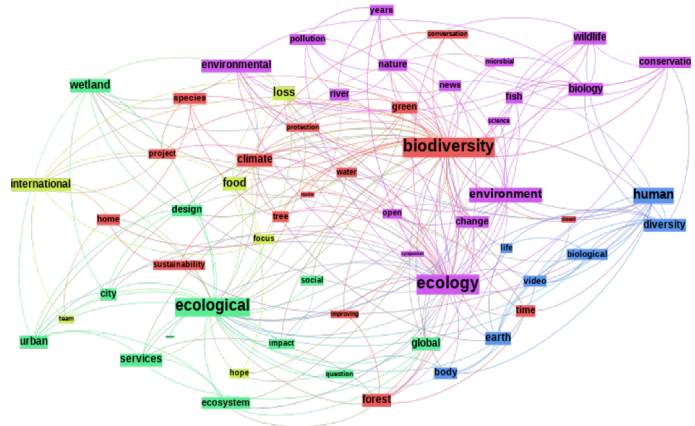
Social Graphs



Molecules

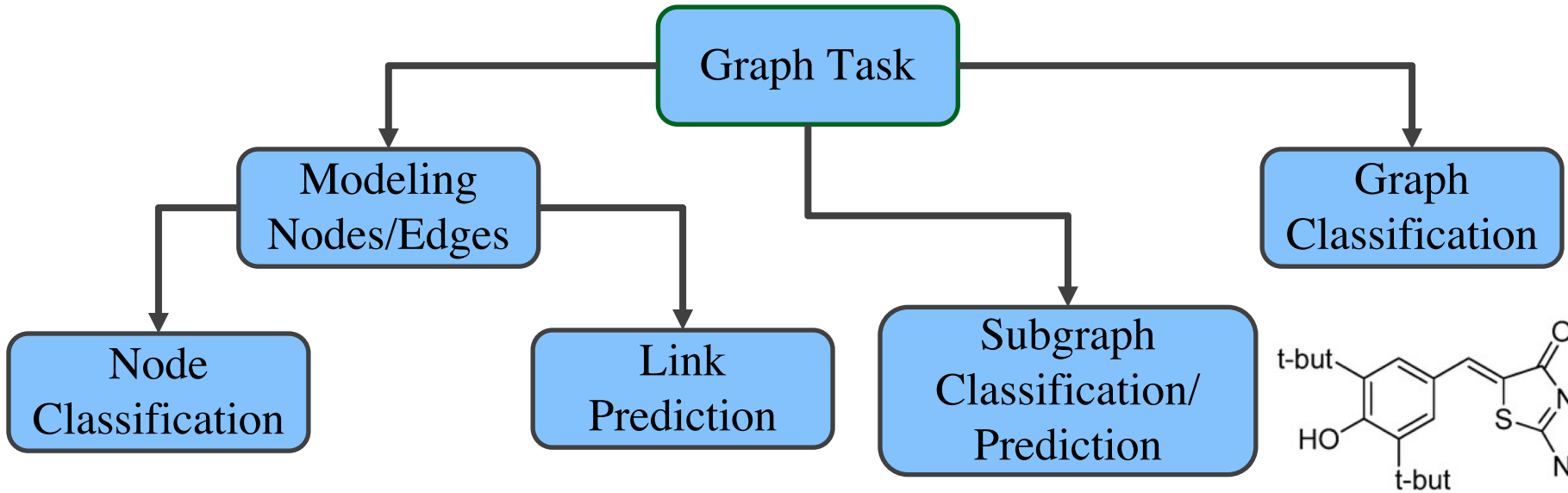


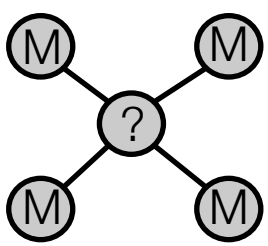
The Web



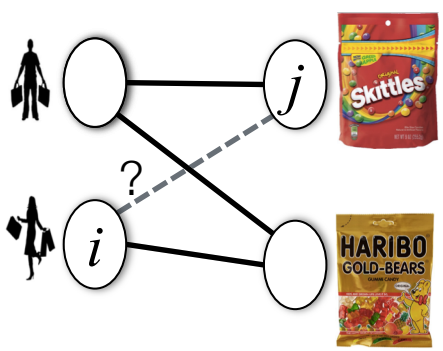
Ecological Graphs

Graph Tasks





M – Male
F- Female



Predict edge (i,j)

Predict group dynamics

Toxic
Not Toxic

Before solutions,
a few questions

1) Isomorphism Question

- Are two graphs different?

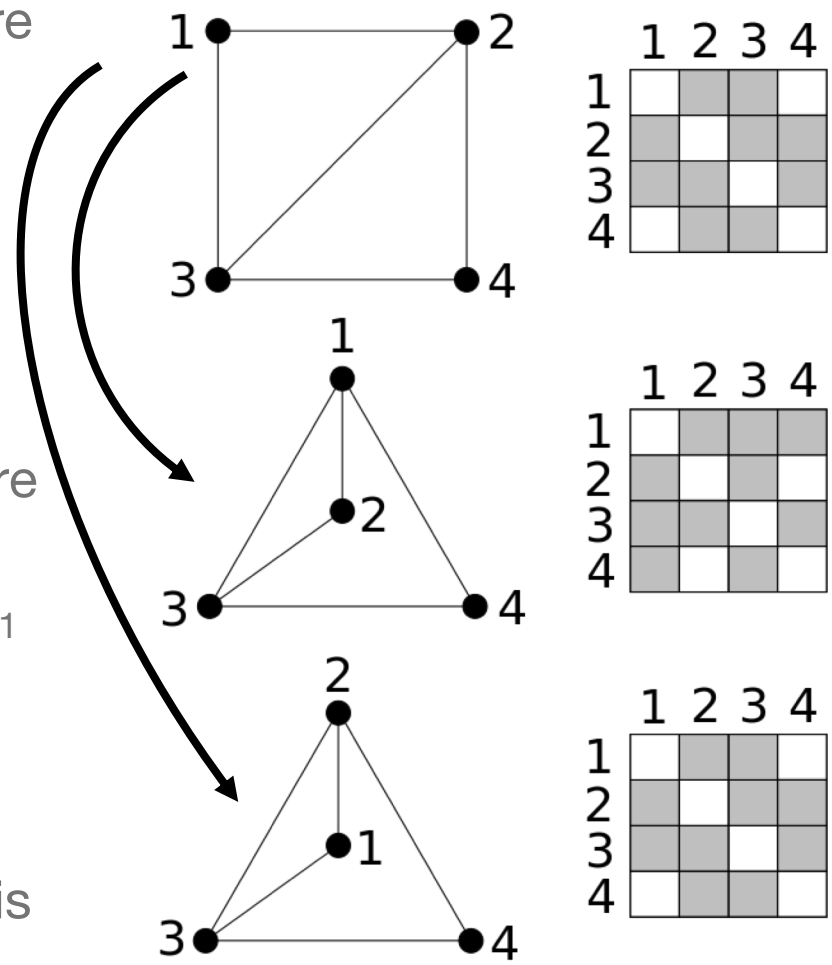
Isomorphic-invariant neural network gives same results over all isomorphic graphs

  - If the adjacency matrices A_1 and A_2 are the same, the graphs are the same

  - If adjacency matrices are different… are the graphs different?

    - Two graphs are isomorphic is there is a permutation π that when applied to rows and columns of A_1 makes it equal to A_2

    - A neural network is invariant to isomorphism when the its output is invariant to graph isomorphism



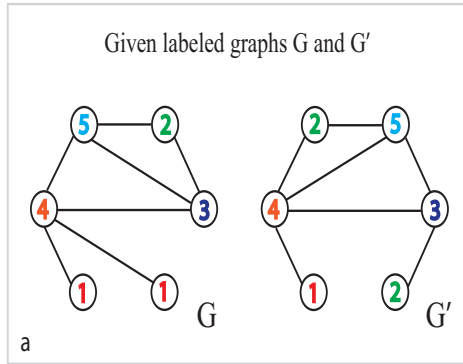
Source: wikipedia

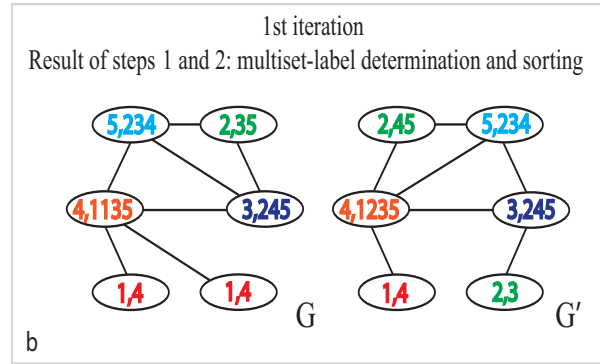
Graph Isomorphism

- Babai (2017)
  - Proves graph isomorphism can be solved in quasi-polynomial time.
  - Can be solved more quickly if we allow some false positives

Weisfeiler-Lehmann Algorithm

- Recursive algorithm to determine if two graphs are isomorphic
  - Valid isomorphism test for almost all graphs (Babai and Kucera, 1979)
  - Cai et al., 1992 shows examples that cannot be distinguished by it
  - Belongs to class of color refinement algorithms that iteratively update vertex “colors” (hash values) until it has converged to unique assignments of hashes to vertices
  - Final hash values encode the structural roles of vertices inside a graph
  - Often fails for graphs with a high degree of symmetry, e.g. chains, complete graphs, tori and stars





Shervashidze et al. 2011

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

$t \leftarrow 0$;

repeat

  for $v_i \in \mathcal{V}$ **do**

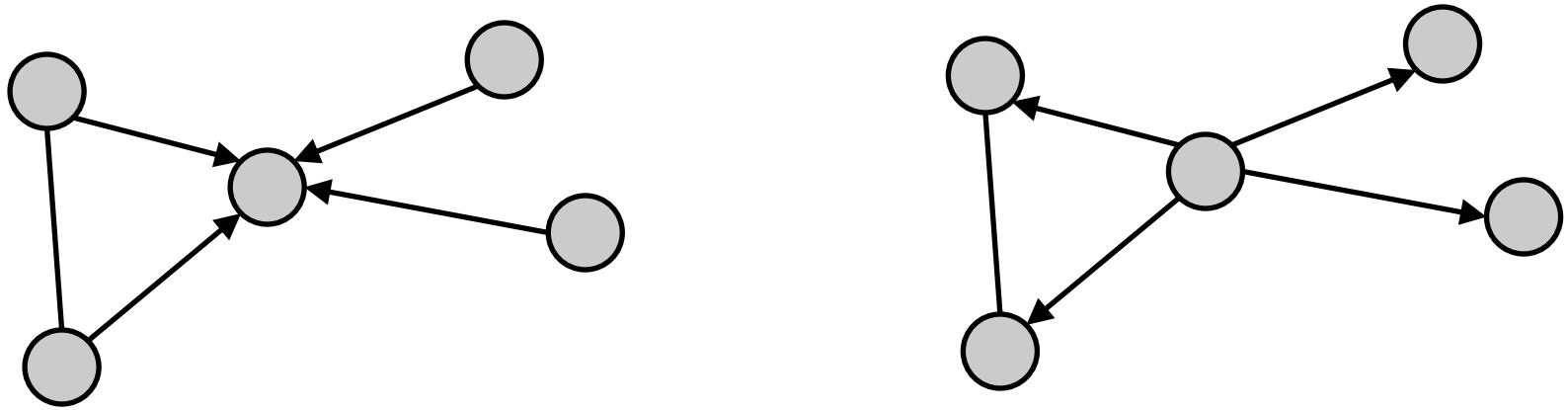
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;

  $t \leftarrow t + 1$;

until *stable node coloring is reached*;

Neighbors of node i

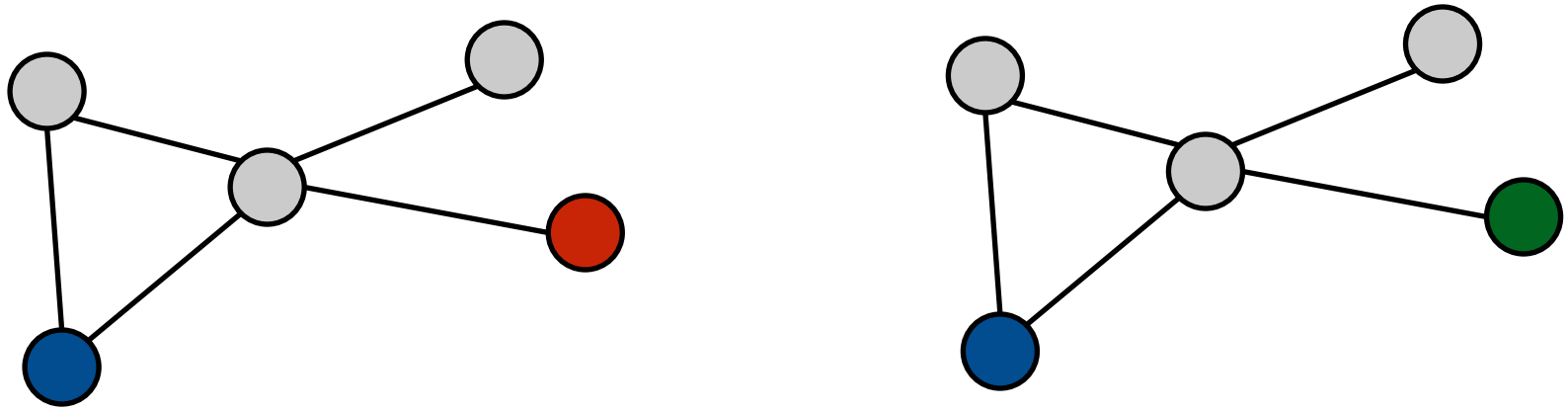
2) Is the Graph Directed?



- Should a neural network receiving these two graphs as input, give different output values?

- Some of the algorithms we see today cover directed graphs

3) Is the Attributed?



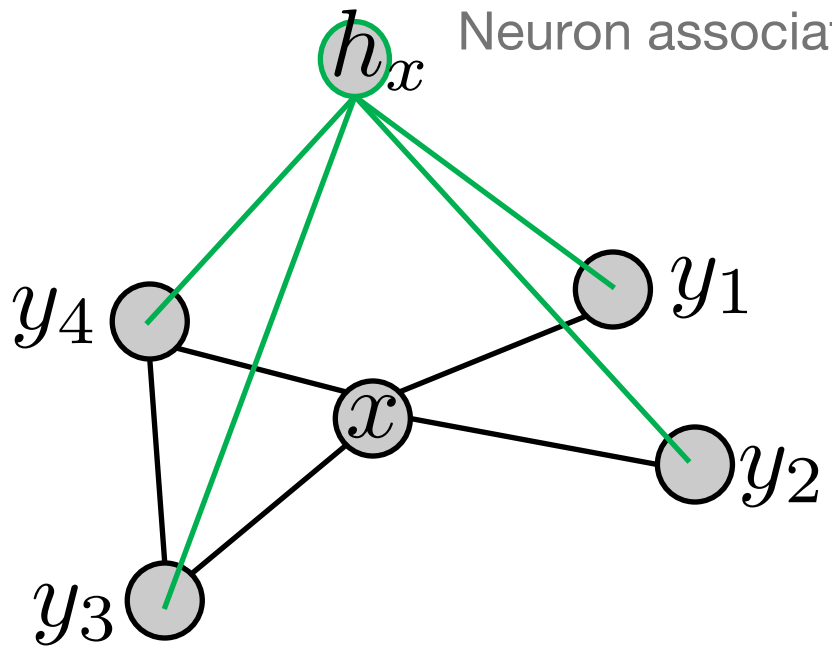
- Graph attributes are node and edge categorical labels or real values associated to nodes and edges

- Should a neural network receiving these above topological equivalent graphs with distinct attributes give different output values?

Our algorithms must have the above 3 properties

Traditional Convolutional Networks are not Isomorphic-invariant

Example of Graph Convolution that is not Isomorphic Invariant



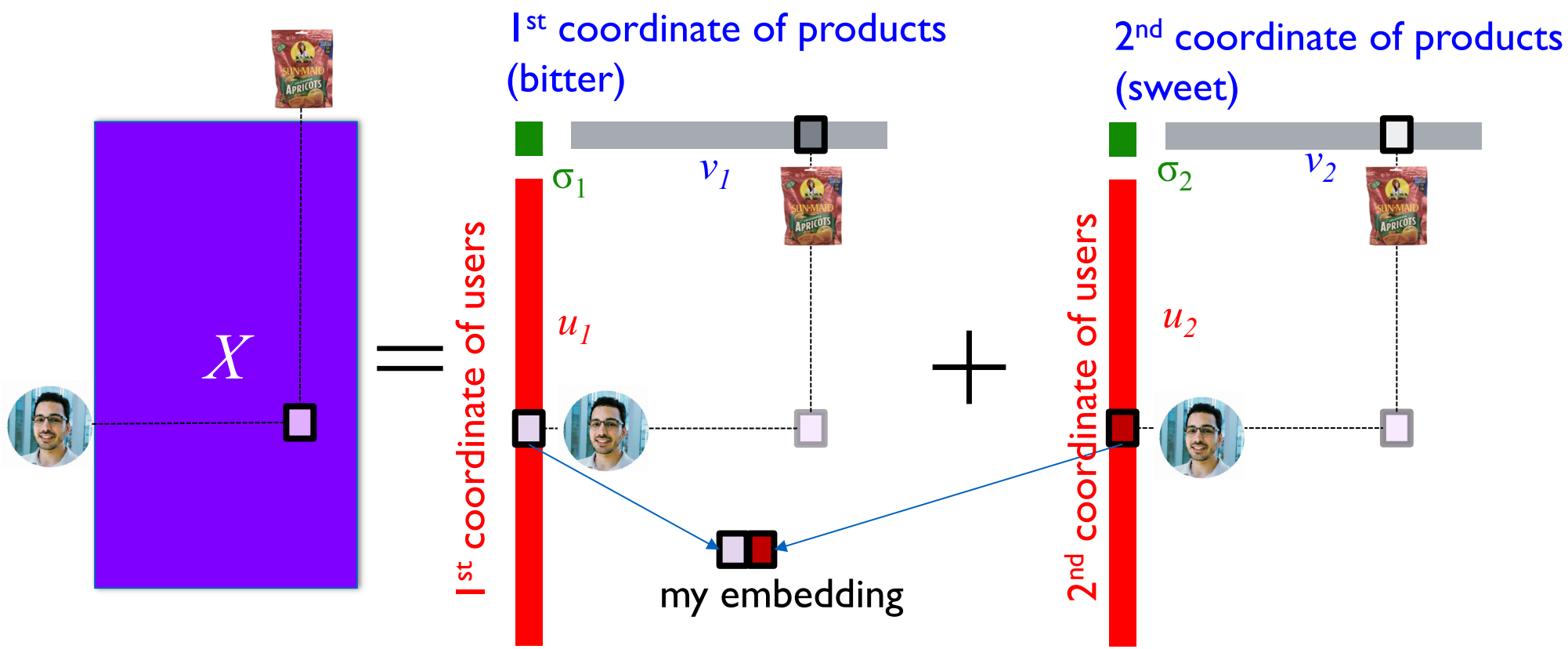
Arbitrary graph

$$h_x = \sigma\left(\sum_i w_i y_i + b\right)$$

Revisiting an Example of an Isomorphic Invariant “Neural Network”

Review: Matrix Factorization

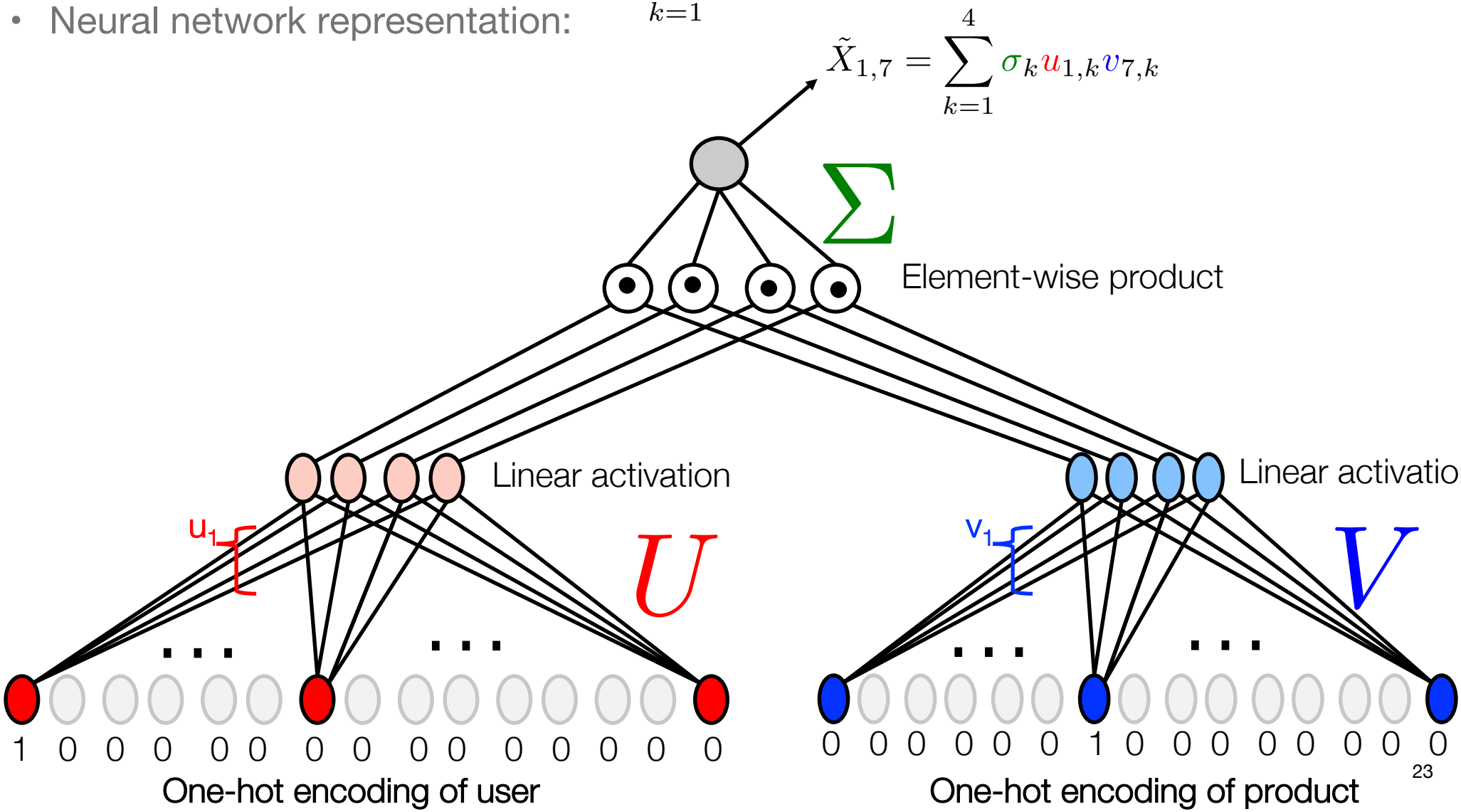
Illustration of a rank-**K** matrix approximation (assuming rank(**X**) < K) :



$$X \approx U\Sigma V^T = \sum_{k=1}^{K} \sigma_k u_k v_k^T$$

Matrix Factorization as a Neural Network (v2)

- But neural networks **can** have isomorphic network representations
- Matrix factorization: $X \approx U\Sigma V^T = \sum_{k=1}^{K} \sigma_k u_k v_k^T$
- Neural network representation:



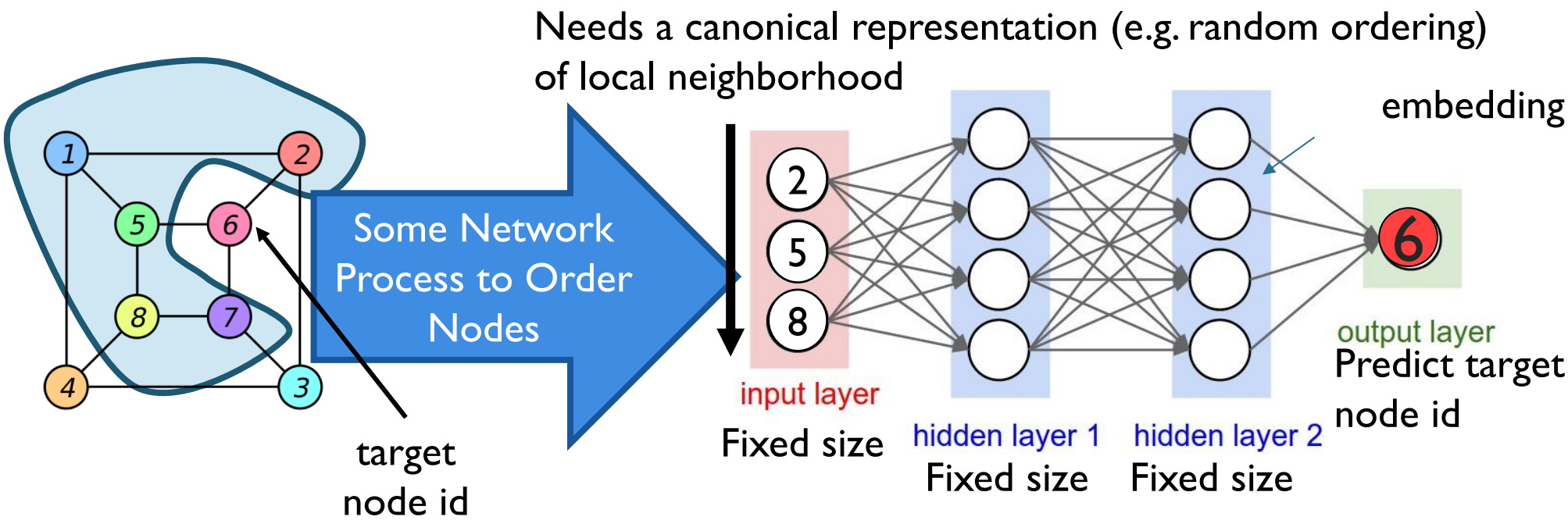
Possible Solutions

- Transductive methods
  - Works only for the “training data” (seen examples)

- Inductive methods
  - Generalizes to “test data” (unseen examples)

Node Classification

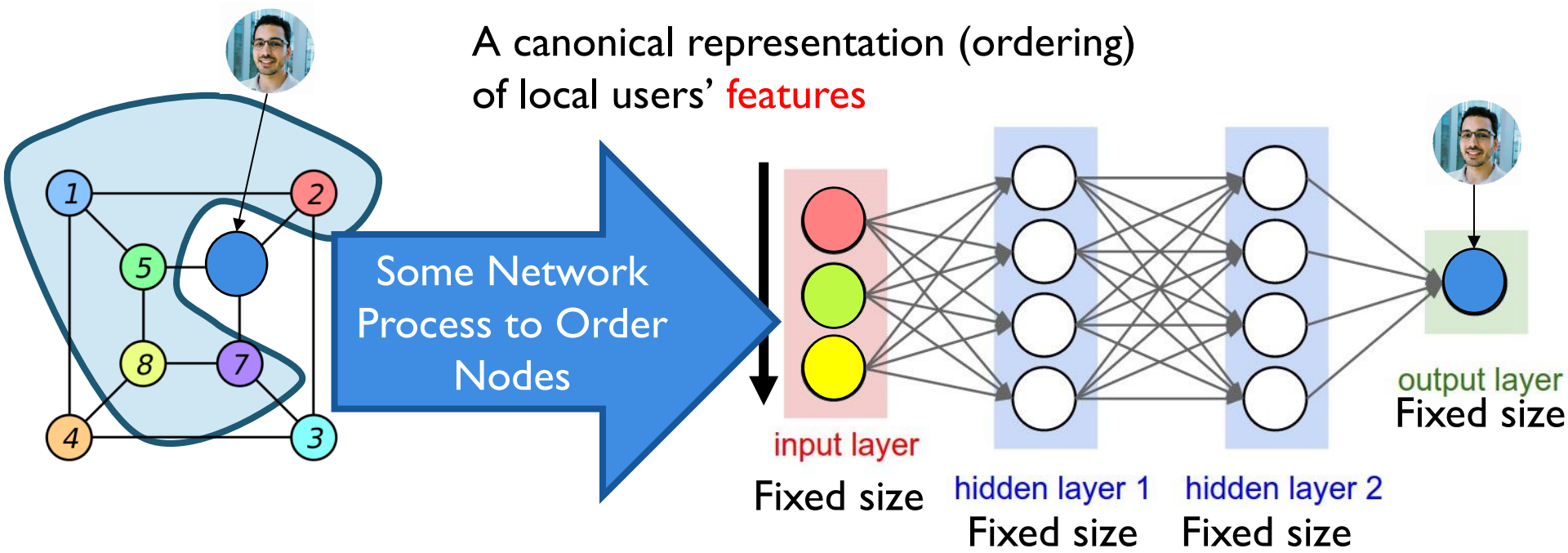
Transductive Learning with Node IDs



- Issues:
  - Node ids as input will not generalize to unseeing examples
  - Network neighborhoods of different sizes (needs to cut input)
  - Heuristic used to order neighbors

  - E.g. node2vec, DeepWalk, LINE, most ___2vec methods

Inductive Learning with Node Features



- Inductive Learning Uses Node Features Rather Than User IDs
- Works well if you have a task that has a canonical ordering

- Without a specific task, same issues as before:
  - Network neighborhoods of different sizes (needs to cut input)
  - Heuristic used to order neighbors

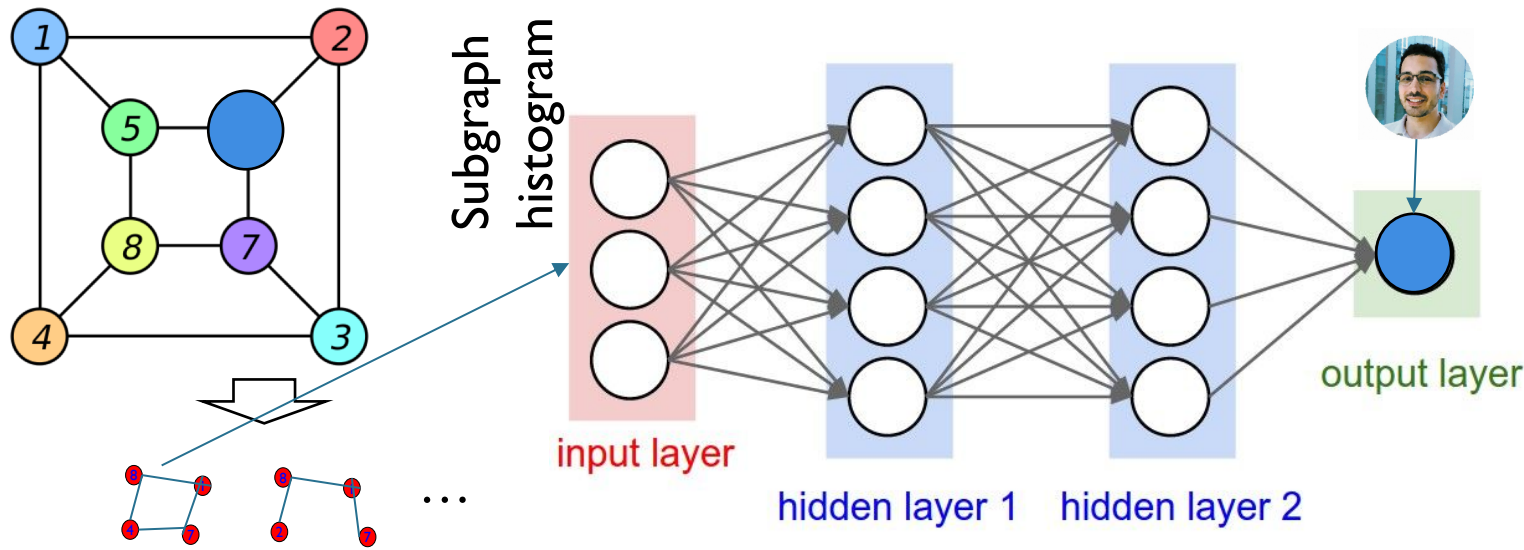
Graph Kernels + Deep Learning

(Orsini et al. 2015) and (Yanardag and Vishwanathan, 2015a, 2015b)

PURDUE UNIVERSITY

Idea:

break network down graph into smaller subgraphs



- Input layer receives frequency of each subgraph patterns in a ball of radius B around node
- Issues:
  - Diameter of network
    - Because of social network have short diameters, problem quickly becomes intractable
    - 100,000 node social network can have as many as 10^{20} subgraphs with 4-nodes
  - High-cost to capture more complex patterns (larger subgraphs)
  - Cannot interpret learned neural network parameters

F. Orsini, P. Frasconi, and L. De Raedt (2015). Graph Invariant Kernels. In *IJCAI*

P. Yanardag and S. Vishwanathan (2015a). A Structural Smoothing Framework For Robust Graph Comparison. In *NIPS*

P. Yanardag and S. Vishwanathan (2015b). Deep Graph Kernels. In *SIGKDD*

Inductive Graph Convolutions

Node Classification and Edge Prediction

Type A: (WL) Graph Convolution Neural Networks

- Weisfeiler-Lehman-type algorithm

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

$t \leftarrow 0$;

repeat

  for $v_i \in \mathcal{V}$ **do**

    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;

  $t \leftarrow t + 1$;

until *stable node coloring is reached*;

Replace by a neuron

Common weights for all neighbors

$$\mathbf{h}_i^{(t+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(t)} \mathbf{W}^{(t)}\right)$$

Vector of hidden activations associated with vertex i at step t+1

Hidden activations associated with vertex j at step t

"weight" of edge (i,j)

Neighbors of node i

Node Label or Link Prediction from Node Embedding

- At last step K, aggregate all intermediate representation of nodes in a vector:

$$(\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})$$

- Loss function is a function of this vector
- Node classification (classes 1 and 0):
  - Prediction: $\hat{y}_i = \sigma((\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})^T \mathbf{w})$
  - Loss function: $L = (\hat{y}_i)^{y_i}(1 - \hat{y}_i)^{1-y_i}$

parameter vector (to learn)

- Edge prediction:
  - Predict $A_{i,j}$: $\hat{A}_{i,j} = \sigma((\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})^T \mathbf{W_F} (\mathbf{h}_j^{(1)}, \ldots, \mathbf{h}_j^{(K)}))$
  - Loss function: $L = (\hat{A}_{i,j})^{A_{i,j}}(1 - \hat{A}_{i,j})^{1-A_{i,j}}$

parameter matrix (to learn)

- Entire neural network is trained using backpropagation

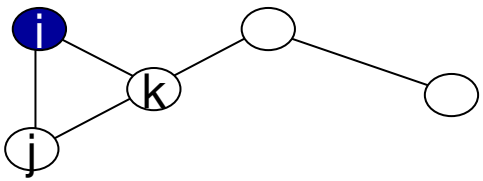
Type B: (Diffusion Kernel) Graph Conv. Neural Networks

- Diffusion Kernel: Probability that a random walker starting at node i reaches node j in t steps
- If A is the adjacency matrix, one-step transition probability from node i to j

$$\mathbf{P}_{i,j} = \frac{1}{d_i} A_{i,j}$$

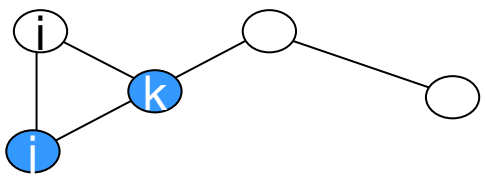
Evolution:

- t = 0



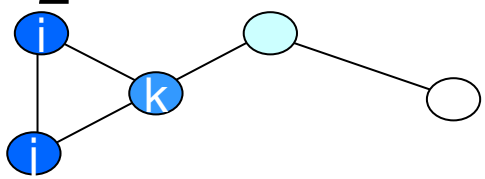
$$\pi^{(0)} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- t = 1



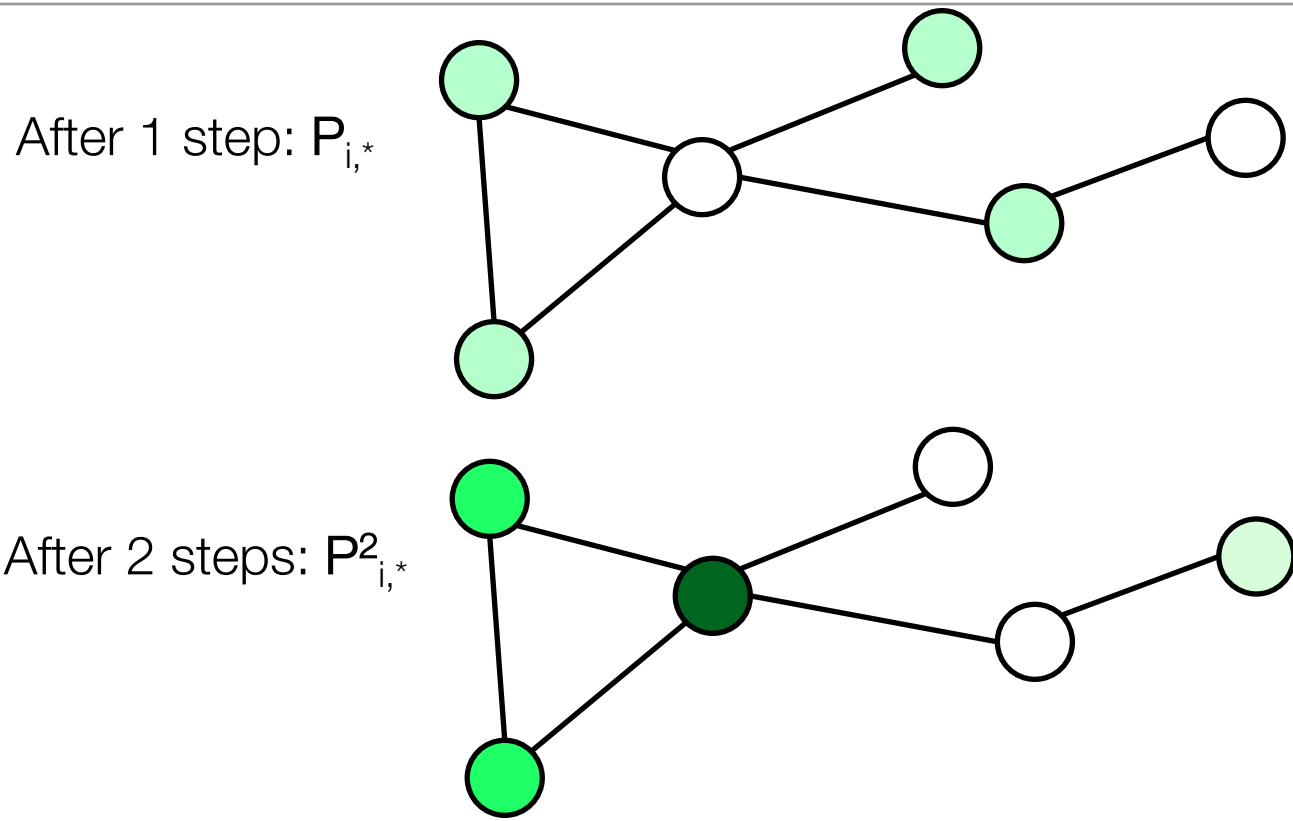
$$\pi^{(1)} = \pi^{(0)} P$$

- t = 2



$$\pi^{(2)} = \pi^{(0)} P^2$$

Type B: Diffusion Kernel (cont)



- Hidden activations associated with vertex i at step t+1:

$$\mathbf{h}_i^{(t+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} x_j (\mathbf{P}^t)^T_{i,\star}\right)$$

Vector of hidden activations associated with vertex i at step t+1

Encoding of vertex j (e.g., degree, label, centrality metric, …)

Neighbors of node i

Row i of P^t

Node Label or Link Prediction from Node Embedding

- At last step K, aggregate all intermediate representation of nodes in a vector:

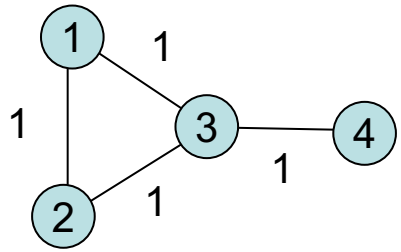
$$(\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})$$

- Loss function is a function of this vector
- Node classification (classes 1 and 0):
  - Prediction: $\hat{y}_i = \sigma((\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})^T \mathbf{w})$
  - Loss function: $L = (\hat{y}_i)^{y_i}(1 - \hat{y}_i)^{1-y_i}$

- Edge prediction:
  - Predict $A_{i,j}$: $\hat{A}_{i,j} = \sigma((\mathbf{h}_i^{(1)}, \ldots, \mathbf{h}_i^{(K)})^T \mathbf{W_F} (\mathbf{h}_j^{(1)}, \ldots, \mathbf{h}_j^{(K)}))$
  - Loss function: $L = (\hat{A}_{i,j})^{A_{i,j}}(1 - \hat{A}_{i,j})^{1-A_{i,j}}$

- Neural network is trained using backpropagation

Laplacian Diffusion Kernel



$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

From A we construct the graph Lapacian matrix (Normalized Laplacian)

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

where

$$T^{-1/2} = \begin{bmatrix} -\frac{1}{\sqrt{d_1}} & 0 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{d_2}} & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{d_3}} & 0 \\ 0 & 0 & 0 & -\frac{1}{\sqrt{d_4}} \end{bmatrix}$$

If $d_v = 0$, $T^{-1/2}(v,v) = 0$

$\mathbf{d_v}$ – degree of vertex $\mathbf{v}$

Graph Laplacian

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

$$\mathcal{L}(u,v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0, \\ -\dfrac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

Graph Laplacian: applied over vertex values

For any real values g defined over the vertices

$$\langle g, \mathcal{L}g \rangle = \sum_{(u,v)\in E} \left(\frac{g(u)}{\sqrt{d_u}} - \frac{g(v)}{\sqrt{d_v}} \right)^2 \quad , \text{ where } g = \begin{pmatrix} g(v_1) \\ g(v_2) \\ g(v_3) \\ g(v_4) \end{pmatrix}$$

Inner product

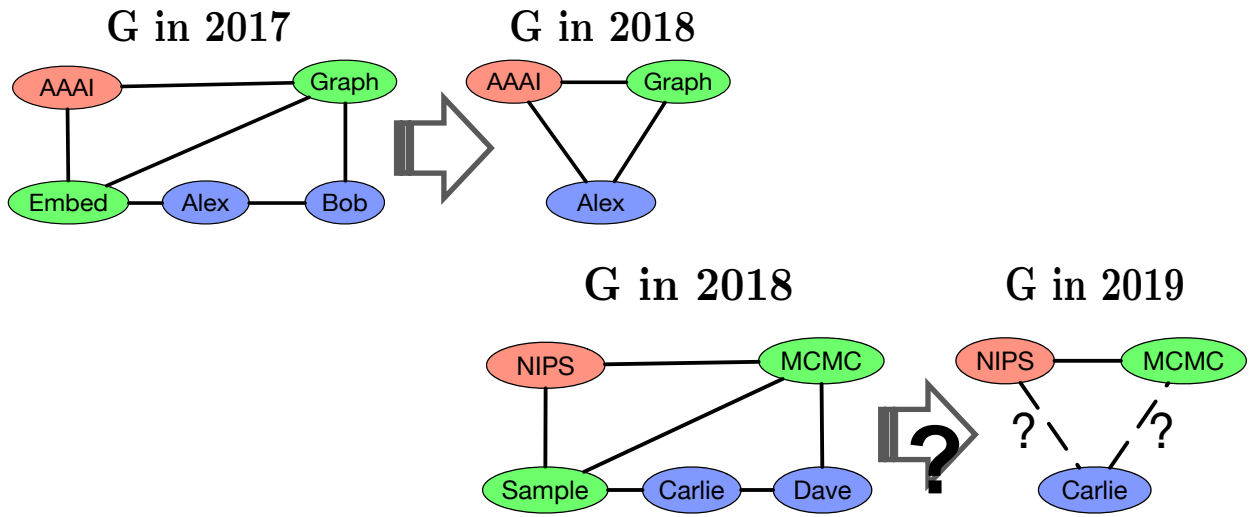
- The quantity $\frac{g(u)}{\sqrt{d_u}} - \frac{g(v)}{\sqrt{d_v}}$ measures flow imbalance between neighbors u and v
  - How much of the quantity g(u) node u can transfer to v minus the amount the v transfers to u
  - This imbalance tells us how well nodes communicate in the graph
  - Node embedding at step t is:

$$\mathbf{h}_i^{(t+1)} = \sigma \left(\sum_{j\in\mathcal{N}_i} \frac{1}{c_{ij}} x_j (\mathcal{L}^t)^T_{i,\star} \right)$$

Subgraph Prediction Tasks

Inductive Prediction of Complex Relationships

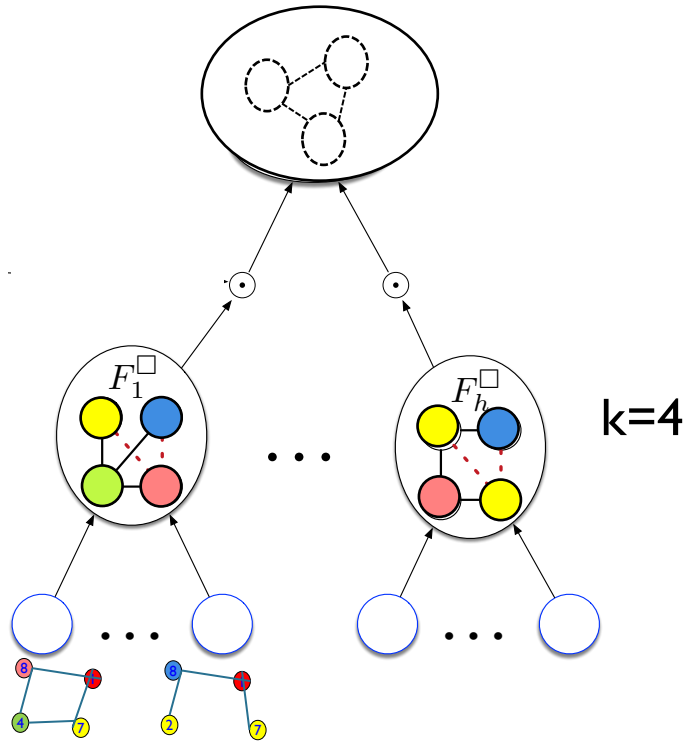
- Heterogeneous Relationships Evolving over Time



- Goal: Predict Relationship Evolution
  - More Accurate Multi-entity Predictions
    - Existing methods tend to perform no better than random guesses
  - Only existing technique: Subgraph Pattern Neural Network

Subgraph Pattern Convolutional Neural Network

neural network
structure represents
relationships
between subgraphs
in the larger network



To train this model we need to sample k-node subgraphs from a very large graph.
Using sampling it takes about 4 min in a 100,000 node graph with k=4

J. Meng, C.M. Sekar, B. Ribeiro, J. Neville, *Predicting Subgraph Evolution in Heterogeneous Dynamic Networks*, *AAAI 2018*

SPNN: Node Embeddings Bad for Subgraph Tasks

PURDUE UNIVERSITY

Results: Better subgraph prediction accuracy in 7 tasks (showing 2 here)
(b) Activity Level Prediction, **(c)** Group dissolution

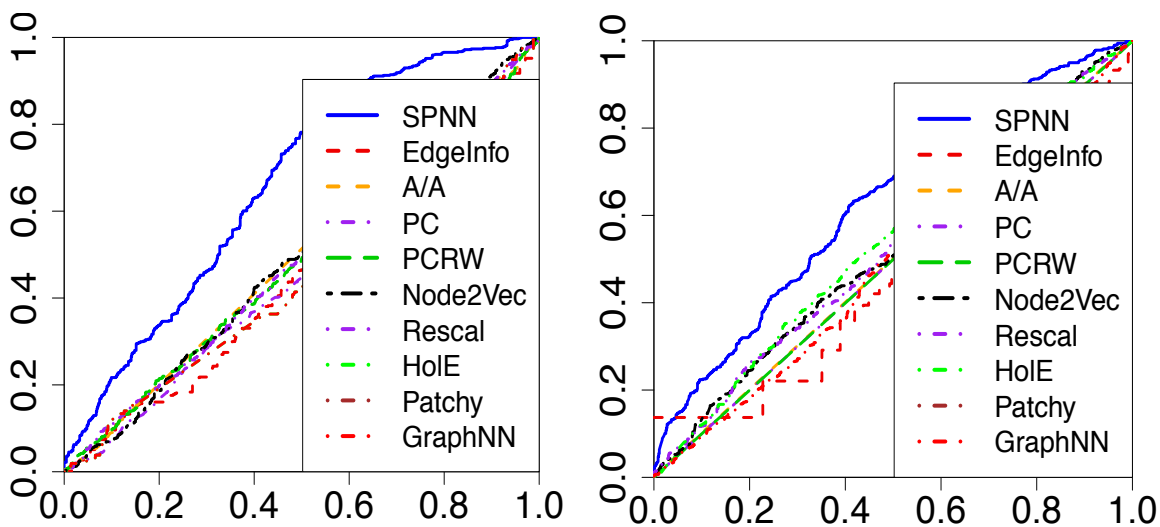
Baselines: Adamic–Adar ,EdgeInfo, PC, PCRW, Node2Vec, Rescal, HolE, Patchy, GraphNN

AUC score

Jointly Trained Multi-Link Task

	EdgeInfo	PCRW	PC	N2V	Rescal	HolE	Patchy	GraphNN	SPNN
DBLP	0.830 ±0.007	0.782 ±0.007	0.788 ±0.014	0.582 ±0.007	0.611 ±0.025	0.690 ±0.024	0.627 ±0.003	0.571 ±0.021	**0.846** ±0.011
Friendster (Activity)	0.502 ±0.007	0.516 ±0.012	0.515 ±0.012	0.524 ±0.018	0.502 ±0.012	0.506 ±0.013	0.519 ±0.010	0.521 ±0.023	**0.690** ±0.008
Friendster (Structure)	0.501 ±0.004	0.502 ±0.002	0.552 ±0.019	0.540 ±0.017	0.521 ±0.017	0.530 ± 0.021	0.547 ± 0.025	0.523 ±0.019	**0.607** ±0.017

ROC curve



(b) Friendster Activity (c) Friendster Structure

References

- Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.
- Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." *Advances in Neural Information Processing Systems*. 2016.
- Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs." *International conference on machine learning*. 2016.
- Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." NIPS (2016).
- Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. Spectral networks and locally connected networks on graphs. ICLR 2014.
- Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in Neural Information Processing Systems. 2017.
- J. Meng, C.M. Sekar, B. Ribeiro, J. Neville, Predicting Subgraph Evolution in Heterogeneous Dynamic Networks, AAAI 2018
- J.-Y. Cai, M. Furer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
- L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Proceedings Symposium on Foundations of Computer Science*, pages 39–46, 1979.
- B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, Ser. 2*, 9, 1968.

Extra: Graph Laplacian and Clustering

Graph Laplacian: applied over vertex values

For any real values g defined over the vertices

$$\langle g, \mathcal{L}g \rangle = \sum_{(u,v)\in E} \left(\frac{g(u)}{\sqrt{d_u}} - \frac{g(v)}{\sqrt{d_v}} \right)^2 \quad , \text{ where } g = \begin{pmatrix} g(v_1) \\ g(v_2) \\ g(v_3) \\ g(v_4) \end{pmatrix}$$

Inner product

Special g functions: The k-th eigenvector is a special function

$$\lambda_k g_k = \mathcal{L} g_k$$

And

$$\langle g_k, \mathcal{L}g_k \rangle = \langle g_k, \lambda_k g_k \rangle = \lambda_k \langle g_k, g_k \rangle$$

From which we have

$$\lambda_k = \frac{\sum_{(u,v)\in E} \left(\frac{g_k(u)}{\sqrt{d_u}} - \frac{g_k(v)}{\sqrt{d_v}} \right)^2}{\sum_v g_k(v)^2}$$

Graph Laplacian eigenvalues

$$\lambda_k = \frac{\sum_{(u,v)\in E} \left(\frac{g_k(u)}{\sqrt{d_u}} - \frac{g_k(v)}{\sqrt{d_v}} \right)^2}{\sum_v g_k(v)^2}$$

Assume $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1}$ and $\|g_k\| = 1$

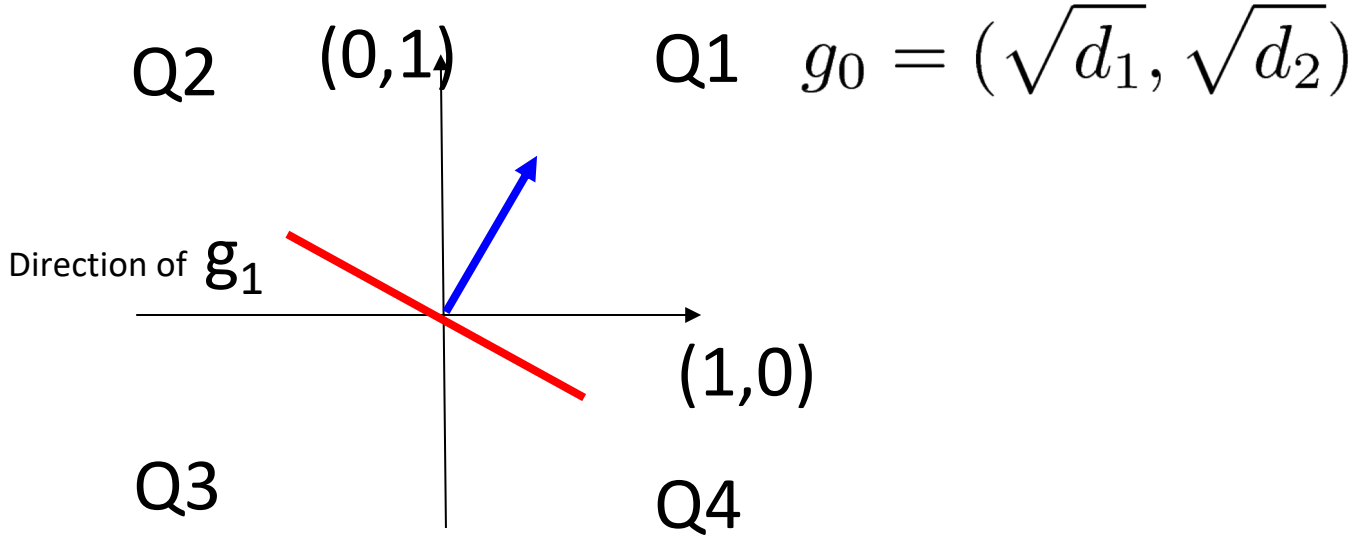
Properties of the eigenvalues

- All non-negative
- $\lambda_0 = ?$
  - Zero
    $g_0(u) = \sqrt{d_u}$
- $g_0 = ?$
  -

Properties of λ_1

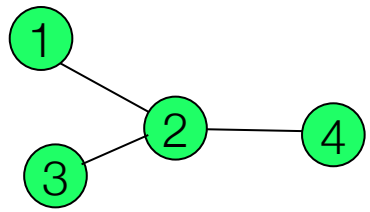
$$\lambda_1 = \inf_{g_1 \perp g_0} \frac{\sum_{(u,v)\in E} \left(\frac{g_1(u)}{\sqrt{d_u}} - \frac{g_1(v)}{\sqrt{d_v}} \right)^2}{\sum_v g_1(v)^2}$$

Geometric interpretation of g_1 (in 2D) [we will use this later]



Properties of zero-valued eigenvalues

- If $\lambda_i = 0$ and $\lambda_{i+1} > 0$ then G has exactly i+1 connected components
  - Why?



$$\lambda_k = \frac{\sum_{(u,v)\in E} \left(\frac{g_k(u)}{\sqrt{d_u}} - \frac{g_k(v)}{\sqrt{d_v}} \right)^2}{\sum_v g_k(v)^2}$$

1) Divide G into i+1 connected graphs G_k, k=1,..,i+1

2) Eigenvectors of G_k have dimension < n and are of the form $g_0^{(k)}(u) = \sqrt{d_u}$

3) G has an eigenvector of dimension *n* : $g_0(u) = \sqrt{d_u}$

4) Can we create i eigenvectors that are perpendicular to g_0 whose eigenvalues are all zero?

$$g_k = \alpha(0, 0, \sqrt{d_3}, 0) + \beta(\sqrt{d_1}, \sqrt{d_2}, 0, \sqrt{d_4})$$

$$\langle g_0, g_k \rangle = \alpha d_3 + \beta(d_1 + d_2 + d_4) = 0$$

Smallest non-zero eigenvalue

Assume G is connected

$$\lambda_1 = \inf_{g_1 \perp g_0} \frac{\sum_{(u,v)\in E} \left(\frac{g_1(u)}{\sqrt{d_u}} - \frac{g_1(v)}{\sqrt{d_v}} \right)^2}{\sum_v g_1(v)^2}$$

Another formulation is

$$\lambda_1 = \min \sum_{(u,v)\in E} \left(\frac{x_u}{d_u} - \frac{x_v}{d_v} \right)^2$$

$$\text{s.t.} \sum_i x_i = 0 \ , \quad \text{and}$$

$$\|x\| = |V|$$

where

$$g_1(i) = \frac{x_i}{\sqrt{d_i}}$$

Recall our geometric remark about g_1?

Minimum cut problem

$$\Phi_G = \min \sum_{(u,v)\in E} (x_u - x_v)^2$$

$$\text{s.t.} \sum_i x_i = 0, \quad \text{and}$$

$$x_i \in \{-1, 1\}$$

Semi-definite relaxation

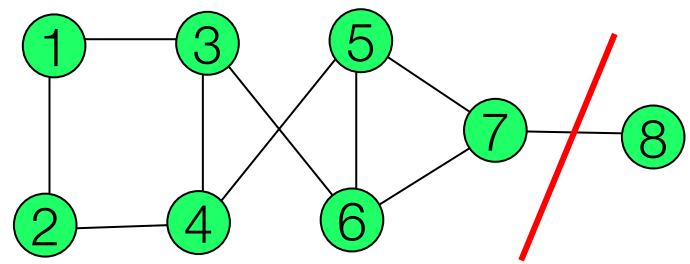
$$\min \sum_{(u,v)\in E} (x_u - x_v)^2$$

$$\text{s.t.} \sum_i x_i = 0, \quad \text{and}$$

$$x_i \in [-1, 1]$$

Clustering and the MinCut problem

- Clustering can be seen as a MinCut problem



- MinCut is easy to solve
  - What is the problem with MinCut?

Clusters should be “reasonably large”

- Addressing the problem $\mathrm{Ncut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A_i})}{\sum_{v \in A_i} d_v}$

Minimum Ncut partition

Let $\mathrm{vol}(A) = \sum_{v \in A} d_v$

$$\mathrm{Ncut}(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{\mathrm{cut}(A_i, \overline{A_i})}{\mathrm{vol}(A_i)}$$

With two partitions

$$\min_{A} \left(\frac{\mathrm{cut}(A, \overline{A})}{\mathrm{vol}(A)} + \frac{\mathrm{cut}(A, \overline{A})}{\mathrm{vol}(\overline{A})} \right)$$

Ncut is the conductance of the graph Φ_G

Conductance & Laplacian

Let $\alpha = \frac{\mathrm{vol}(\bar{A})}{\mathrm{vol}(A)}$ and

$$g'(v) = \begin{cases} \sqrt{d_v \alpha} & \text{if } v \in A \\ -\sqrt{d_v \alpha^{-1}} & \text{if } v \in \bar{A} \end{cases}$$

$$\langle g', \mathcal{L} g' \rangle = \sum_{(u,v) \in E} \left(\frac{g'(u)}{\sqrt{d_u}} - \frac{g'(v)}{\sqrt{d_v}} \right)^2$$

Conductance & Laplacian

$$\langle g', \mathcal{L} g' \rangle = \sum_{(u,v)\in E} \left(\frac{g'(u)}{\sqrt{d_u}} - \frac{g'(v)}{\sqrt{d_v}} \right)^2 =$$

$$= \sum_{(u,v)\in E(A)} \left(\alpha^{1/2} - \alpha^{1/2} \right)^2 + \sum_{(u,v)\in E(\bar{A})} \left(-\alpha^{-1/2} + \alpha^{-1/2} \right)^2 +$$

$$\sum_{(u,v)\in E(A,\bar{A})} \left(\alpha^{1/2} + \alpha^{-1/2} \right)^2 =$$

$$= \sum_{(u,v)\in E(A,\bar{A})} (\alpha + 2 + \alpha^{-1}) = \delta(A, \bar{A})(\alpha + 2 + \alpha^{-1}) =$$

(cont...)

$$\mathrm{cut}(A, \overline{A})(\alpha + 2 + \alpha^{-1}) =$$

$$= \mathrm{cut}(A, \overline{A})(\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})} + \frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)} + 2)$$

$$= \mathrm{cut}(A, \overline{A})(\frac{\mathrm{vol}(A) + \mathrm{vol}(\overline{A})}{\mathrm{vol}(\overline{A})} + \frac{\mathrm{vol}(\overline{A}) + \mathrm{vol}(A)}{\mathrm{vol}(A)})$$

$$= \mathrm{vol}(V)\,\mathrm{Ncut}(A, \overline{A})$$

Find two cluster with minimal Ncut

Traditional definition:

$$\min_{A} \left(\frac{\mathrm{cut}(A, \overline{A})}{\mathrm{vol}(A)} + \frac{\mathrm{cut}(A, \overline{A})}{\mathrm{vol}(\overline{A})} \right)$$

Alternate definition:

$$\min_{A} \langle g', \mathcal{L} g' \rangle$$
$$\text{s.t. } g' \perp g_0,$$
$$\|g'\| = \mathrm{vol}(V),$$

where

$$g'(v) = \begin{cases} \sqrt{d_v \alpha} & \text{if } v \in \overline{A} \\ -\sqrt{d_v \alpha^{-1}} & \text{if } v \in A \end{cases} \quad \text{with} \quad \alpha = \frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}$$

Relax minimal Ncut problem

Allowing *g'* to be real valued

$$\min_{g'} \langle g', \mathcal{L} g' \rangle$$
$$\text{s.t. } g' \perp g_0, \text{ and}$$
$$\|g'\| = \text{vol}(V)$$

Warning: There is no guarantee on the approximation ratio

Drineas' CUR technique? Chooses g' from columns of A.