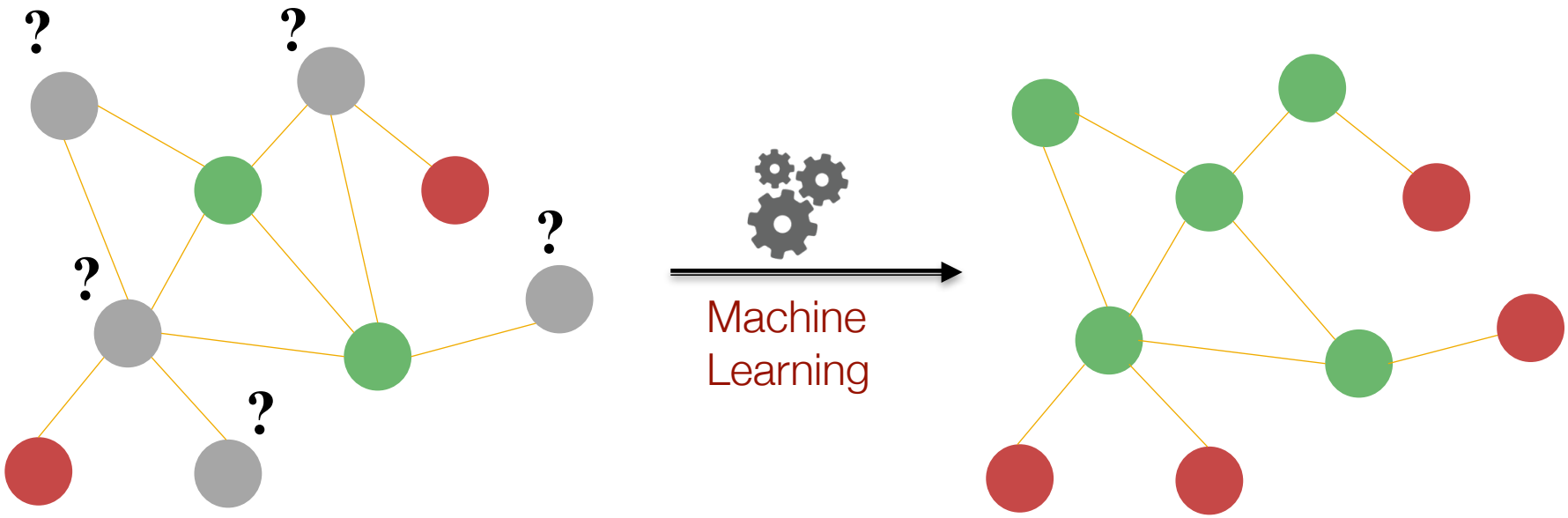
Automatic Feature Learning In Graphs

CS224W: Analysis of Networks
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

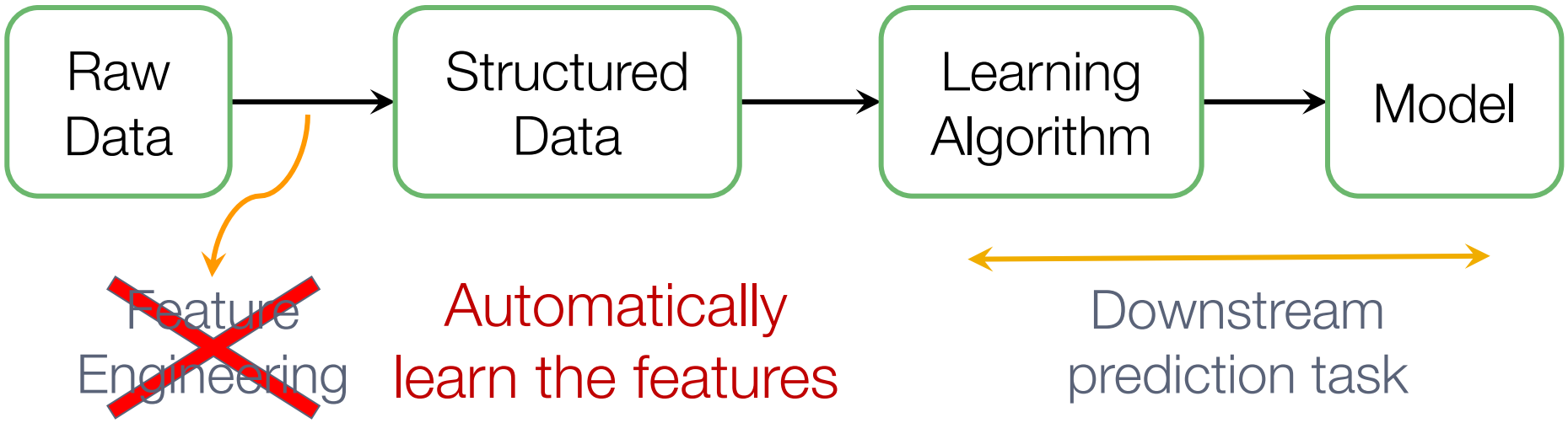
Machine Learning in Networks



Node classification

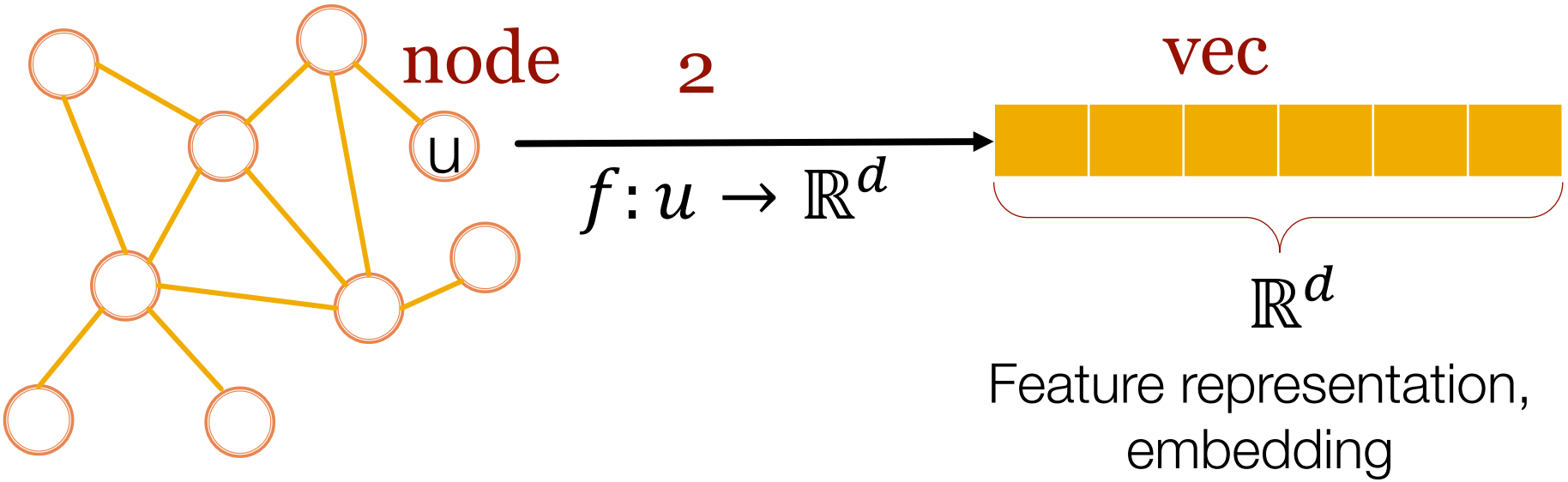
Machine Learning Lifecycle

- **(Supervised) Machine Learning Lifecycle: This feature, that feature. Every single time!**



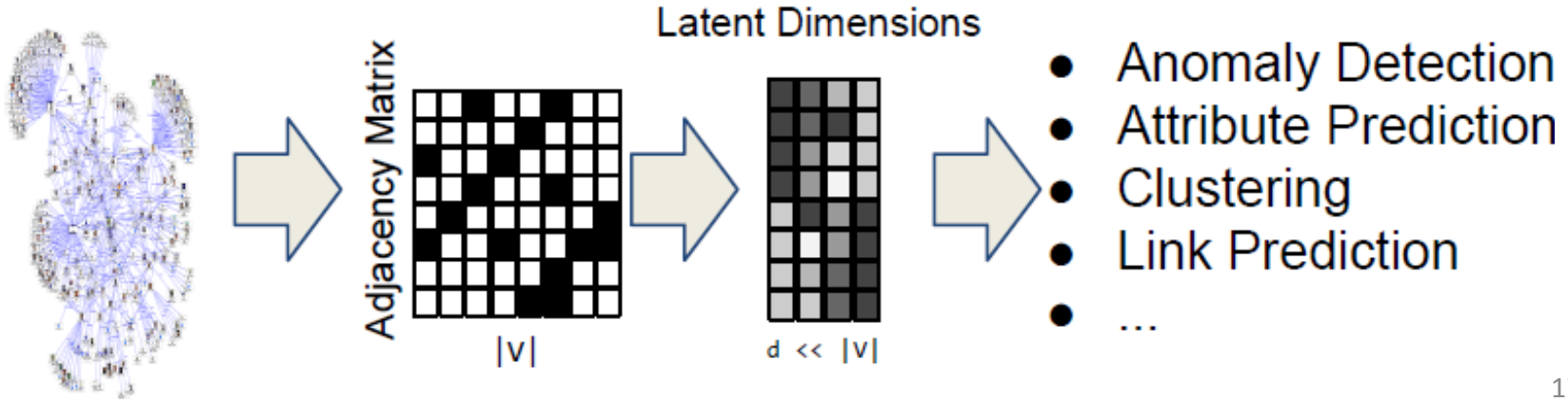
Feature Learning in Graphs

Goal: Efficient task-independent feature learning for machine learning in networks!



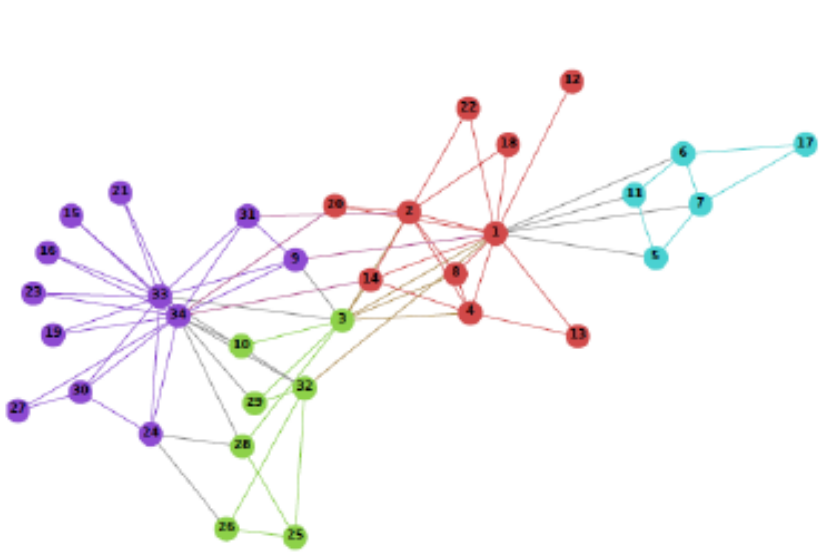
Why network embedding?

- We map each node in a network into a low-dimensional space
  - Distributed representation for nodes
  - Similarity between nodes indicates link strength
  - Encode network information and generate node representation

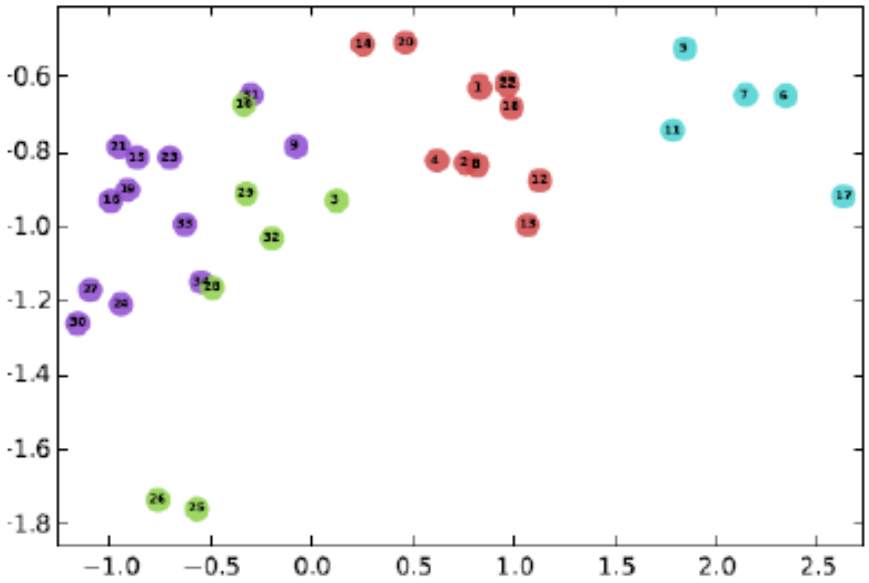


Example

- **Zachary's Karate Club network:**



Input

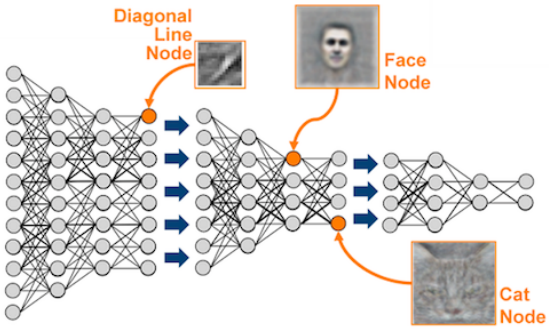


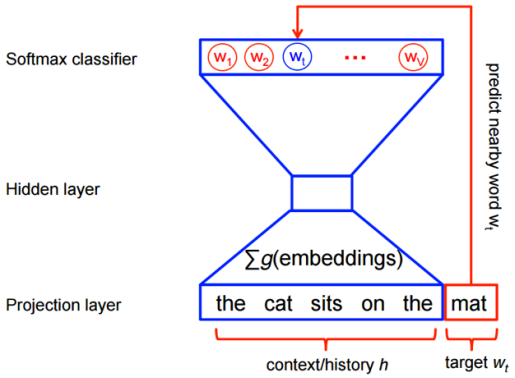
Output

Why Is It Hard?

Graph representation learning is hard:

- Images are fixed size
  - Convolutions (CNNs)
- Text is linear
  - Sliding window (word2vec)
- **Graphs are neither of these!**
  - Node numbering is arbitrary (node isomorphism problem)
  - Much more complicated structure





node2vec: Random Walk Based (Unsupervised) Feature Learning

node2vec: Scalable Feature Learning for Networks
A. Grover, J. Leskovec. KDD 2016.

Overview of node2vec

- **Goal:** Embed nodes with similar network neighborhoods close in the feature space.
- We frame this goal as prediction-task independent maximum likelihood optimization problem.
- Key observation: Flexible notion of network neighborhood $N_S(u)$ of node u leads to rich features.
- Develop biased 2nd order random walk procedure S to generate network neighborhood $N_S(u)$ of node u.

Unsupervised Feature Learning

- Intuition: Find embedding of nodes to d-dimensions that preserves similarity
- Idea: Learn node embedding such that nearby nodes are close together
- Given a node *u*, how do we define nearby nodes?
  - $N_S(u)$ … neighbourhood of *u* obtained by some strategy *S*

Feature learning as optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$.

- Log-likelihood objective:
  $\max_f \sum_{u \in V} \log \Pr(N_S(u) | f(u))$
  - where $N_S(u)$ is neighborhood of node u.

- Given node u, we want to learn feature representations predictive of nodes in its neighborhood $N_S(u)$.

Feature learning as optimization

$$\max_f \sum_{u \in V} \log \Pr(N_S(u) | f(u))$$

- **Assumption:** Conditional likelihood factorizes over the set of neighbors.

$$\log \Pr(N_S(u | f(u)) = \sum_{n_i \in N_S(u)} \log \Pr(f(n_i) | f(u))$$

- Softmax parametrization:

$$\Pr(f(n_i) | f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u)))}$$

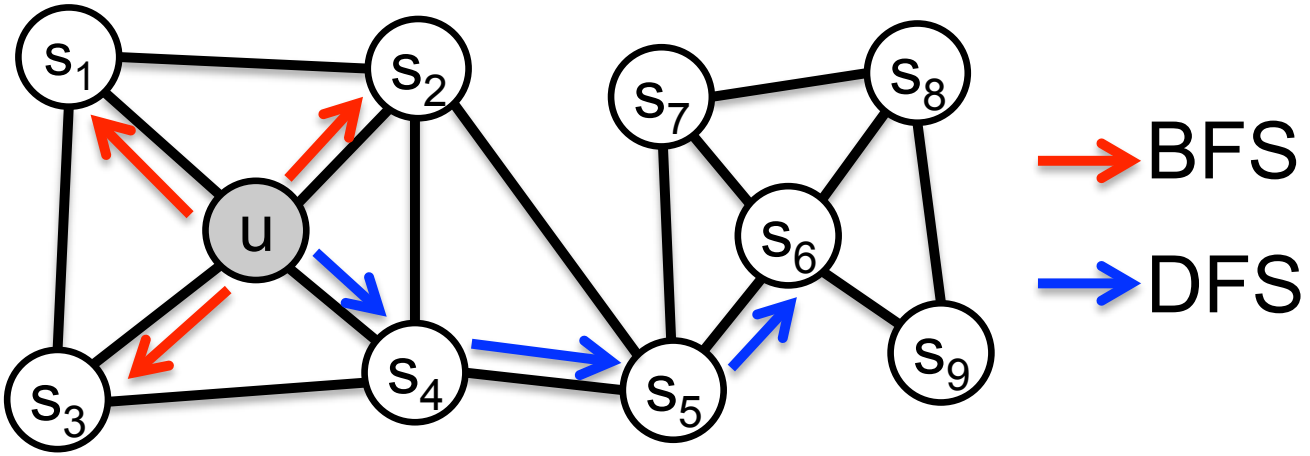
Negative Sampling

$$\max_f \sum_{u \in V} \sum_{n \in N_S(u)} \log \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u)))}$$

- Maximize the objective using Stochastic Gradient descent with **negative sampling**.
  - Computing the summation is expensive
  - **Idea:** Just sample a couple of "negative nodes"
  - This means at each iteration only embeddings of a few nodes will be updated at a time
  - Much faster training of embeddings

How to determine $N_S(u)$

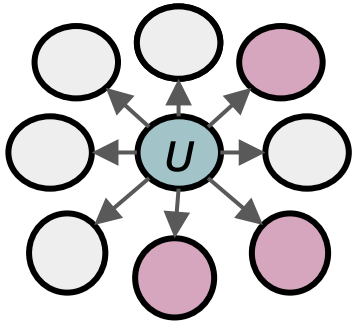
Two classic strategies to define a neighborhood $N_S(u)$ of a given node u:



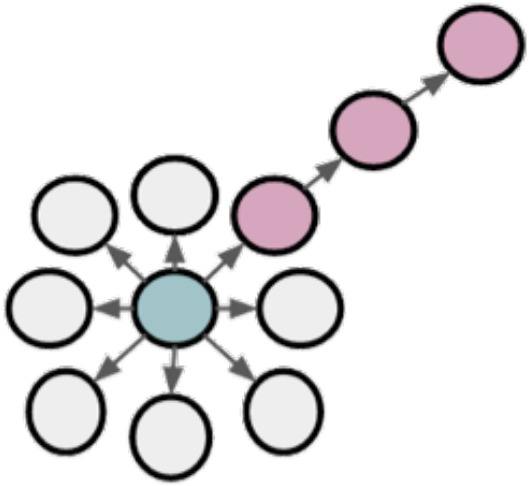
$N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view

$N_{DFS}(u) = \{s_4, s_5, s_6\}$ Global macroscopic view

BFS vs. DFS



BFS:
Micro-view of neighbourhood



DFS:
Macro-view of neighbourhood

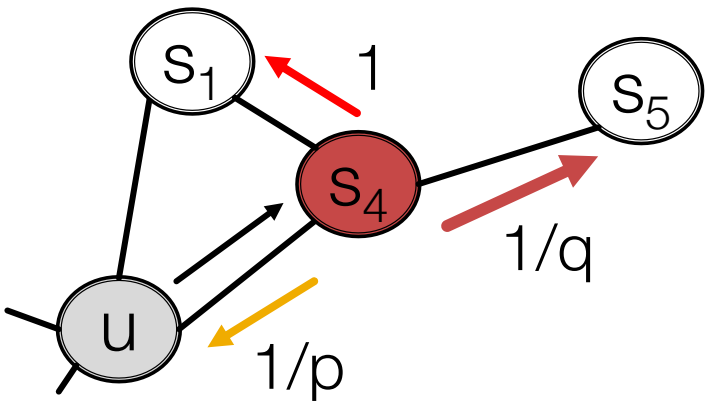
Interpolating BFS and DFS

Biased random walk S that given a node u generates neighborhood $N_S(u)$

- Two parameters:
  - Return parameter p:
    - Return back to the previous node
  - In-out parameter q:
    - Moving outwards (DFS) vs. inwards (BFS)

Biased Random Walks

$N_S(u)$: Biased 2nd-order random walks explore network neighborhoods:



$u \rightarrow s_4 \rightarrow ?$

u
s_1
s_5

- BFS-like: low value of p
- DFS-like: low value of q

p, q can learned in a semi-supervised way

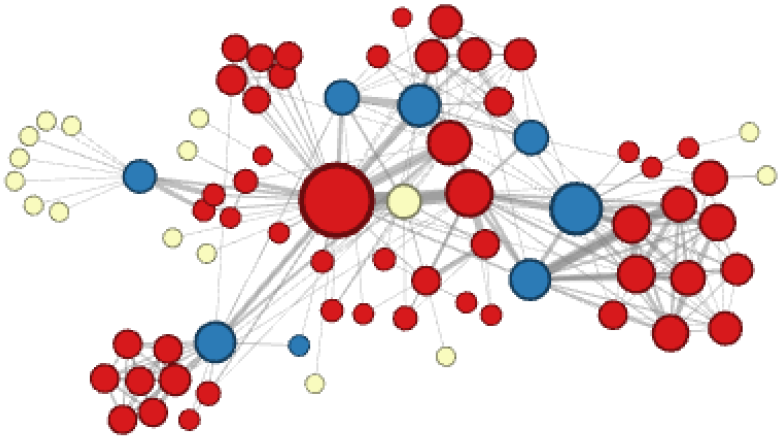
node2vec algorithm

- 1) Compute random walk probs.
- 2) Simulate r random walks of length l starting from each node u
- 3) Optimize the node2vec objective using Stochastic Gradient Descent

Linear-time complexity.
All 3 steps are individually parallelizable

Experiments: Micro vs. Macro

Interactions of characters in a novel:



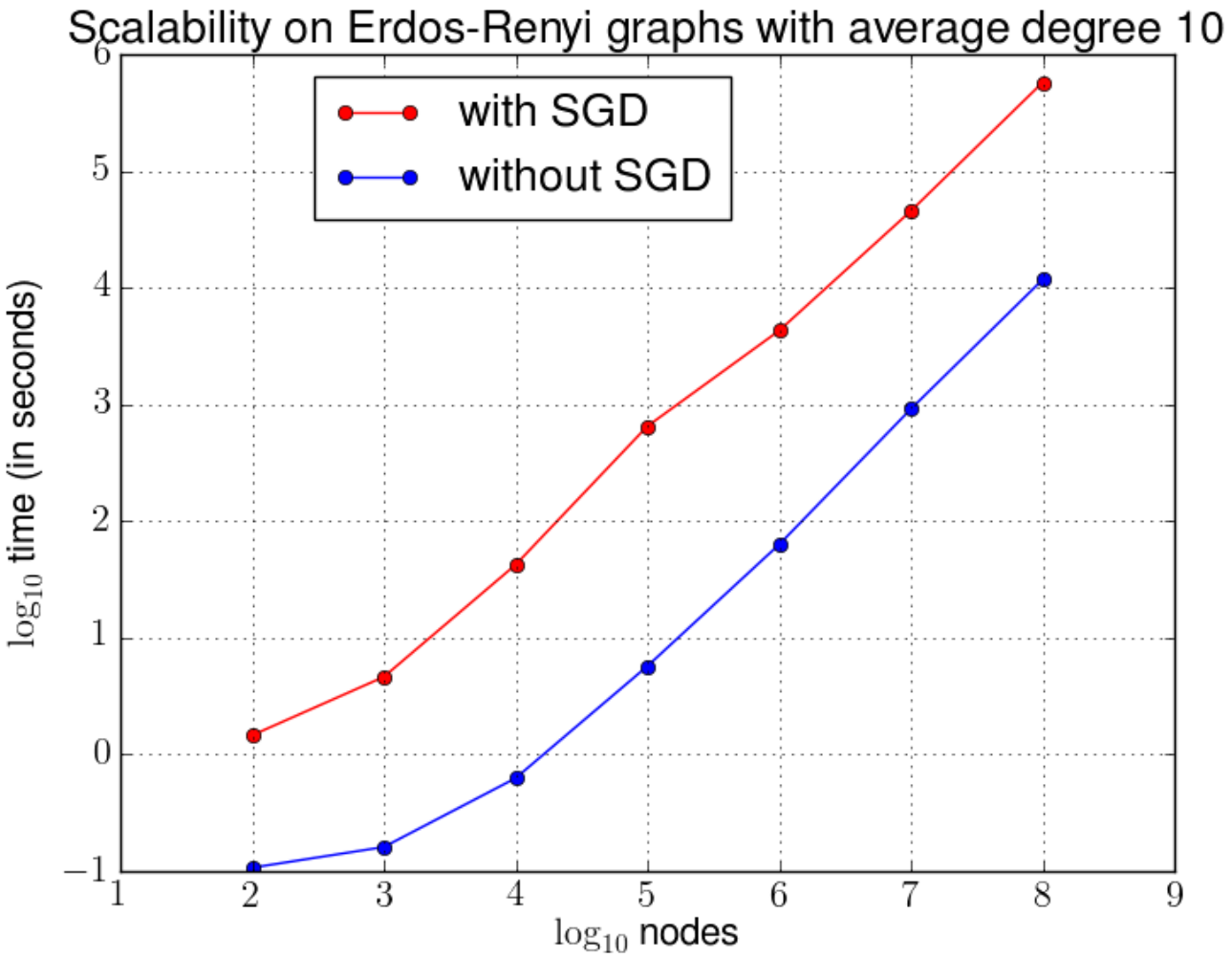
p=1, q=2

Microscopic view of the network neighbourhood

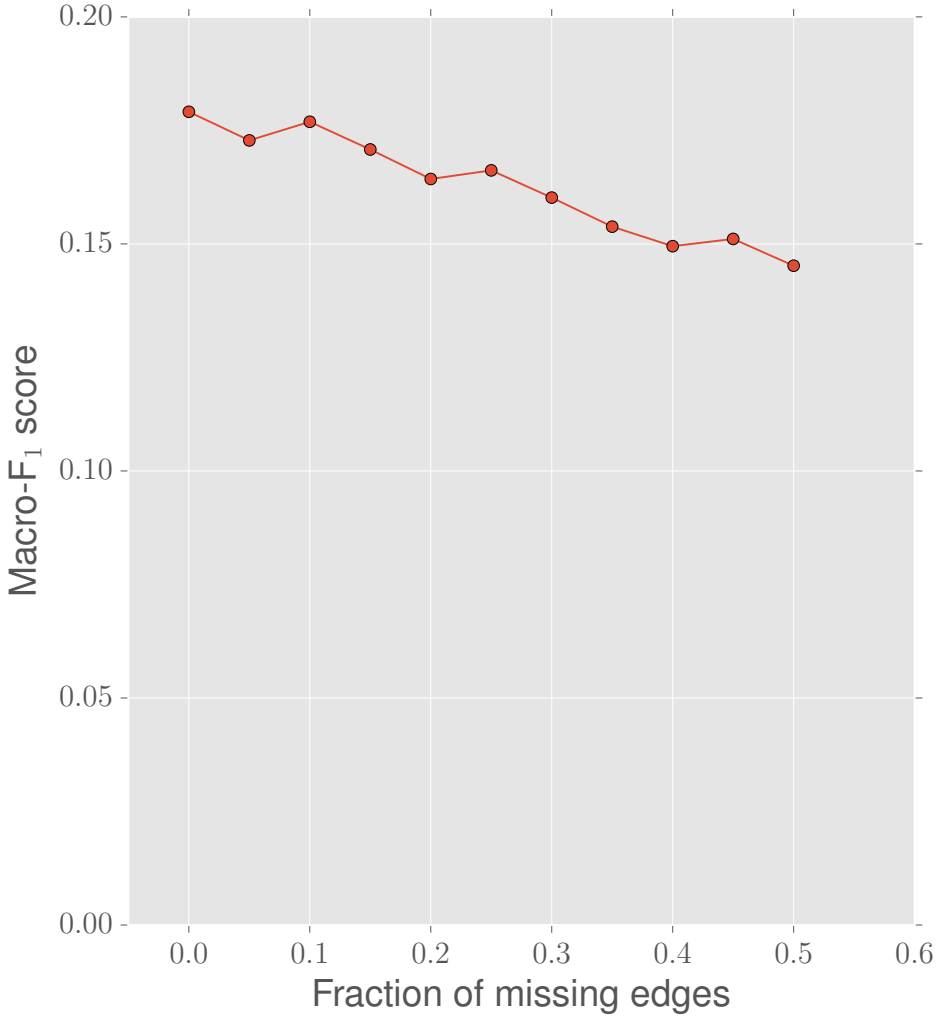
p=1, q=0.5

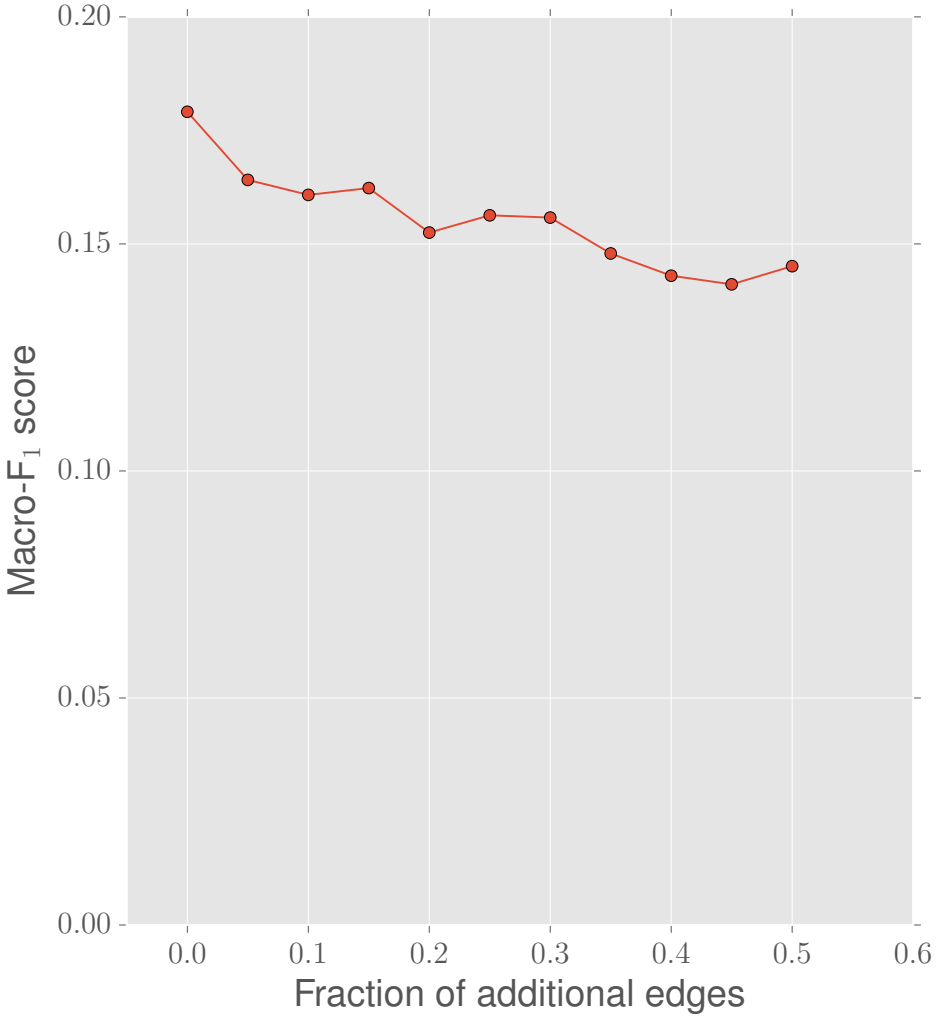
Macroscopic view of the network neighbourhood

Scalability of node2vec



Incomplete Network Data (PPI)





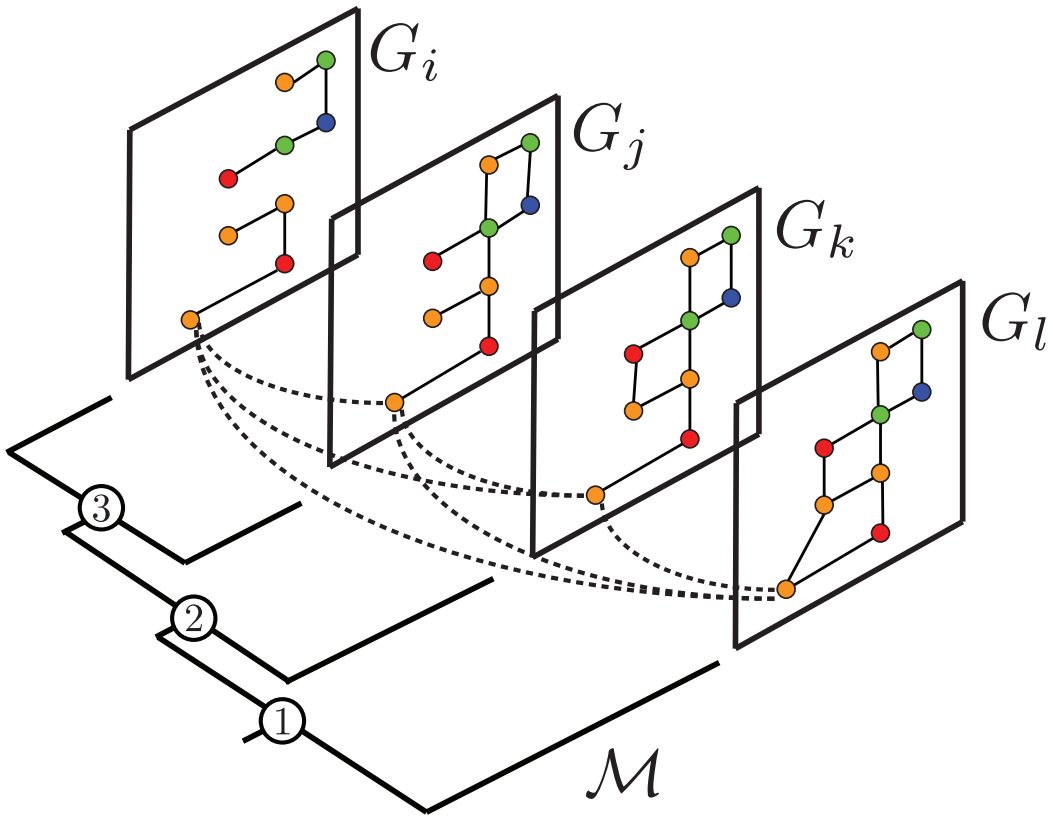
node2vec: Discussion

General-purpose feature learning in networks:

- An explicit locality preserving objective for feature learning.
- Biased random walks capture diversity of network patterns.
- Scalable and robust algorithm with excellent empirical performance.
- Future extensions would involve designing random walk strategies entailed to network with specific structure such as heterogeneous networks and signed networks.

OhmNet: Extension to Hierarchical Networks

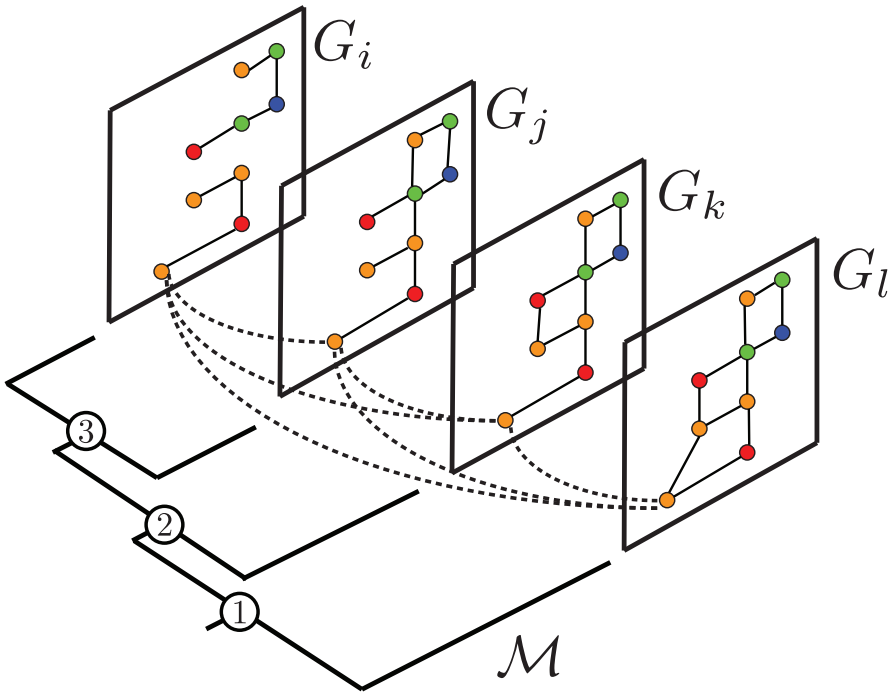
Multilayer Networks



Let's generalize node2vec to multilayer networks!

Multi-Layer Networks

- Each network is a layer $G_i = (V_i, E_i)$
- Similarities between layers are given in hierarchy $\mathcal{M}$, map π encodes parent-child relationships

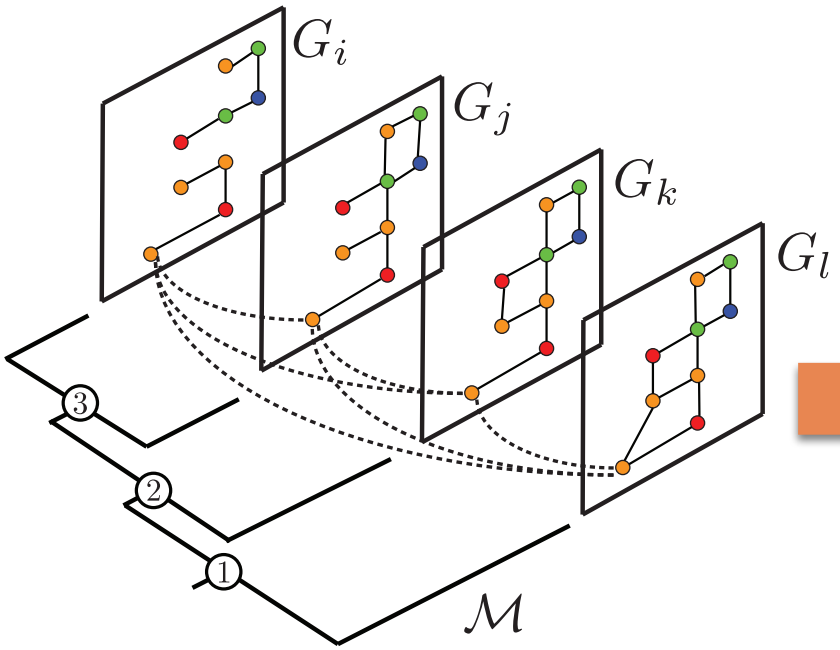


The Approach

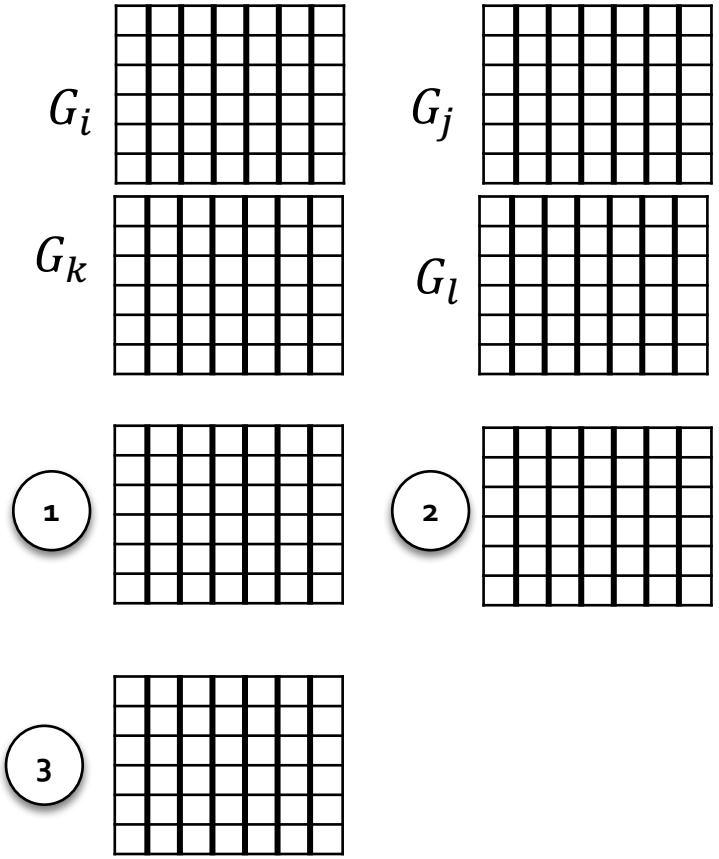
- Computational framework that learns features of every node and at every scale based on:
  - Edges within each layer
  - Inter-layer relationships between nodes active on different layers

OhmNet

Input



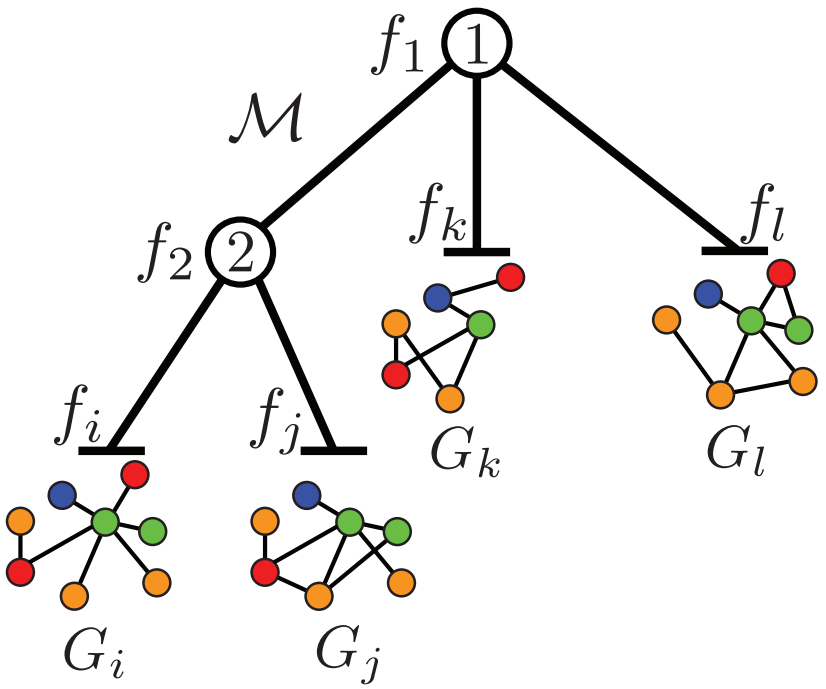
Output: embeddings of nodes in layers as well as internal levels of the hierarchy



OhmNet

- **OhmNet**: Given layers G_i and hierarchy M, learn node features captured by functions f_i

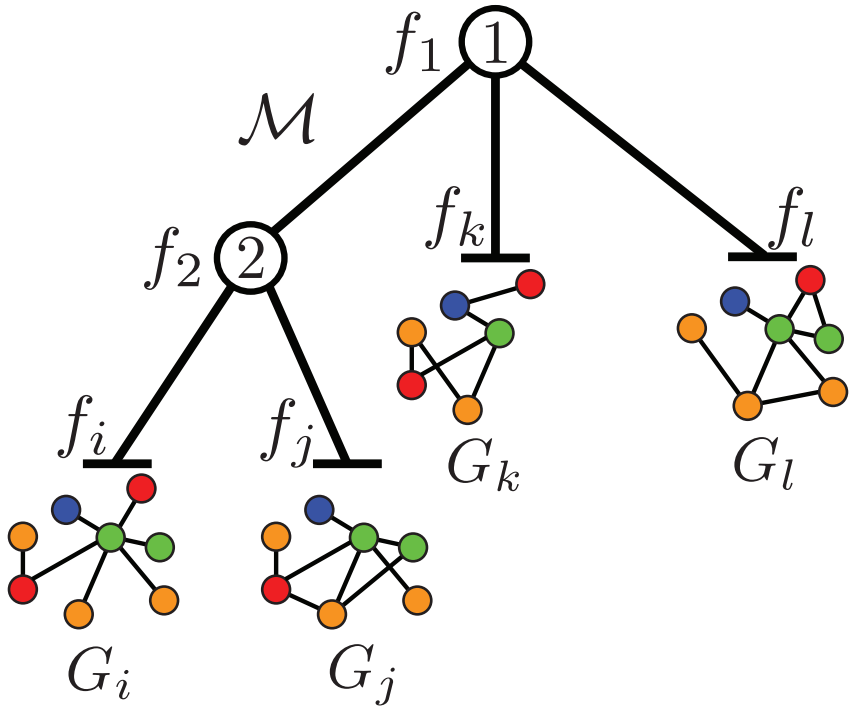
- Functions f_i embed every node in a d-dimensional feature space



A multi-layer network with four layers and a two-level hierarchy M

Features in Multi-Layer Network

- Given: Layers $\{G_i\}$, hierarchy $\mathcal{M}$
  - Layers $\{G_i\}_{i=1..T}$ are in leaves of $\mathcal{M}$
- Goal: Learn functions: $f_i: V_i \rightarrow \mathbb{R}^d$



Features in Multi-Layer Network

- Approach has two components:
  - Per-layer objectives: Nodes with similar network neighborhoods in each layer are embedded close together
  - Hierarchical dependency objectives: Nodes in nearby layers in hierarchy are encouraged to share similar features

Per-Layer Objective: node2vec

- Intuition: For each layer, find a mapping of nodes to d-dimensions that preserves node similarity
- Approach: Similarity of nodes u and v is defined based on similarity of their network neighborhoods
- Given node u in layer i we define nearby nodes $N_i(u)$ based on random walks starting at node u

Per-Layer Objective: node2vec

- Given node u in layer i, learn u's representation such that it predicts nearby nodes $N_i(u)$:

$$\omega_i(u) = \log Pr(N_i(u)|f_i(u))$$

- Given T layers, maximize:

$$\Omega_i = \sum_{u \in V_i} \omega_i(u), \quad \text{for } i = 1, 2, \ldots, T$$

- Notice: Nodes in different networks representing the same entity have different features

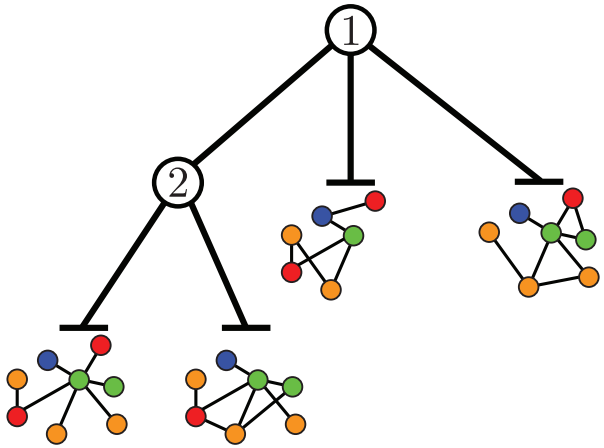
Interdependent Layers

- So far, we did not consider hierarchy $\mathcal{M}$
- Node representations in different layers are learned independently of each other

How to model dependencies between layers when learning node features?

Hierarchical regularization

- We use regularization to share information across the hierarchy
- We want to enforce similarity between feature representations of networks that are located nearby in the hierarchy



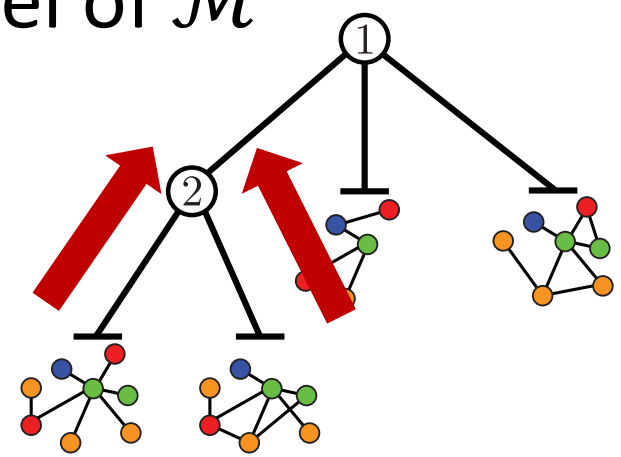
Interdependent Layers

- Given node u, learn u's representation in layer i to be close to u's representation in parent $\pi(i)$:

$$c_i(u) = \frac{1}{2}\|f_i(u) - f_{\pi(i)}(u)\|_2^2$$

- Multi-scale: Repeat at every level of $\mathcal{M}$

$$C_i = \sum_{u \in L_i} c_i(u)$$



L_i has all layers appearing in sub-hierarchy rooted at i

Implications

- Nodes in different layers representing the same entity have the same features in hierarchy ancestors
- We learn feature representations at multiple scales:
  - features of nodes in the layers
  - features of nodes in non-leaves in the hierarchy
- This model is more efficient than the fully pairwise model, where dependencies between layers are modeled by pairwise comparisons of nodes across all pairs of layers

OhmNet: Final Model

Learning node features in multi-layer networks

Solve maximum likelihood problem:

$$\max_{f_1, f_2, \ldots, f_{|M|}} \sum_{i \in \mathcal{T}} \Omega_i - \lambda \sum_{j \in \mathcal{M}} C_j$$

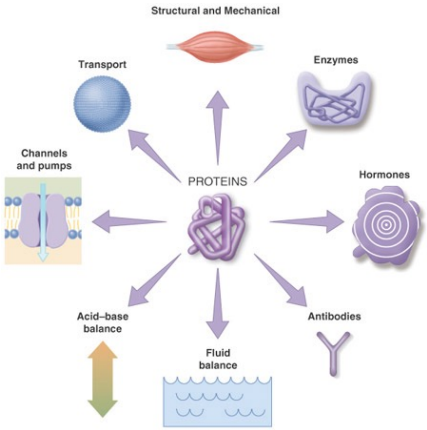
Per-layer network objectives

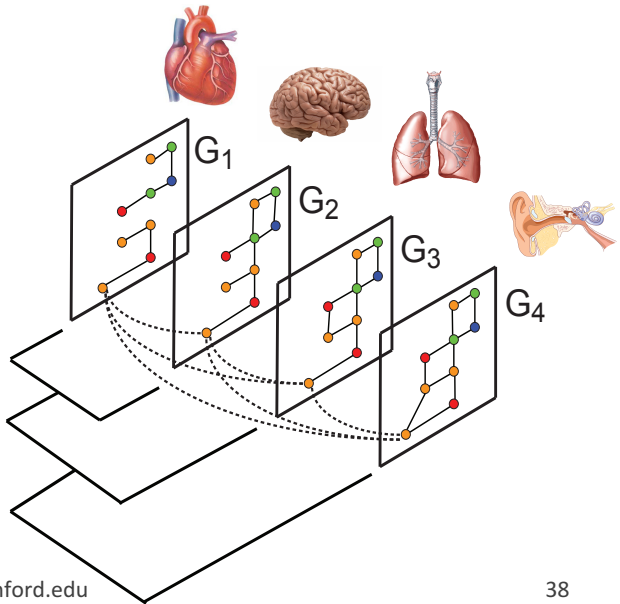
Hierarchical dependency objectives

Application: Protein function

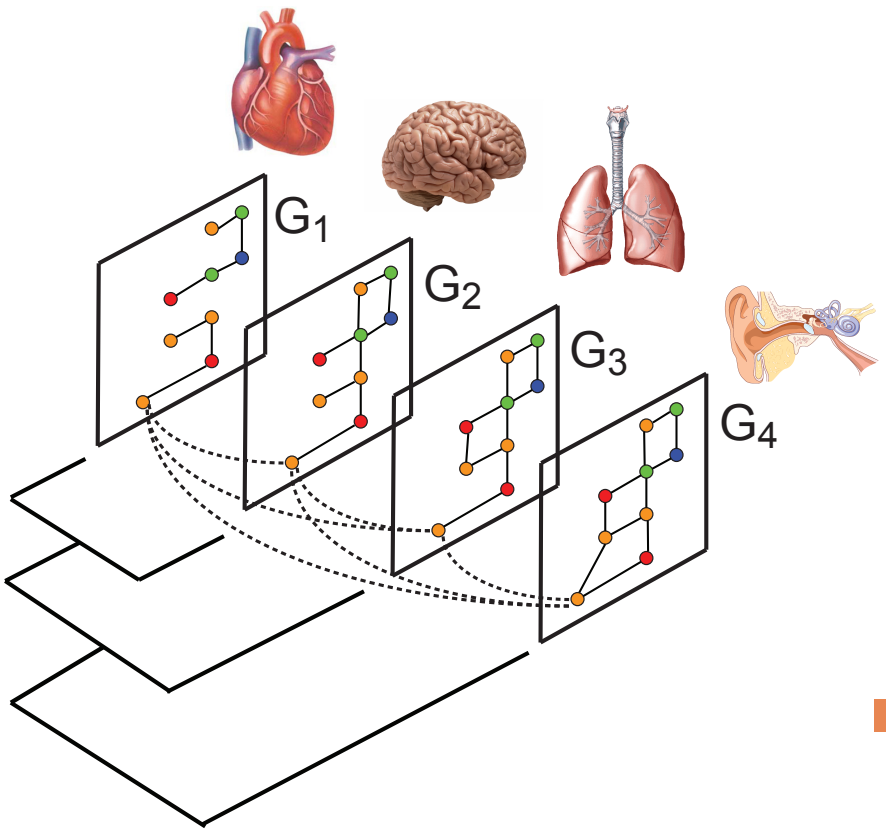
- Proteins are worker molecules
  - Understanding protein function has great biomedical and pharmaceutical implications
- Function of proteins depends on their tissue context

  [Greene et al., Nat Genet '15]





Protein functions are tissue-specific

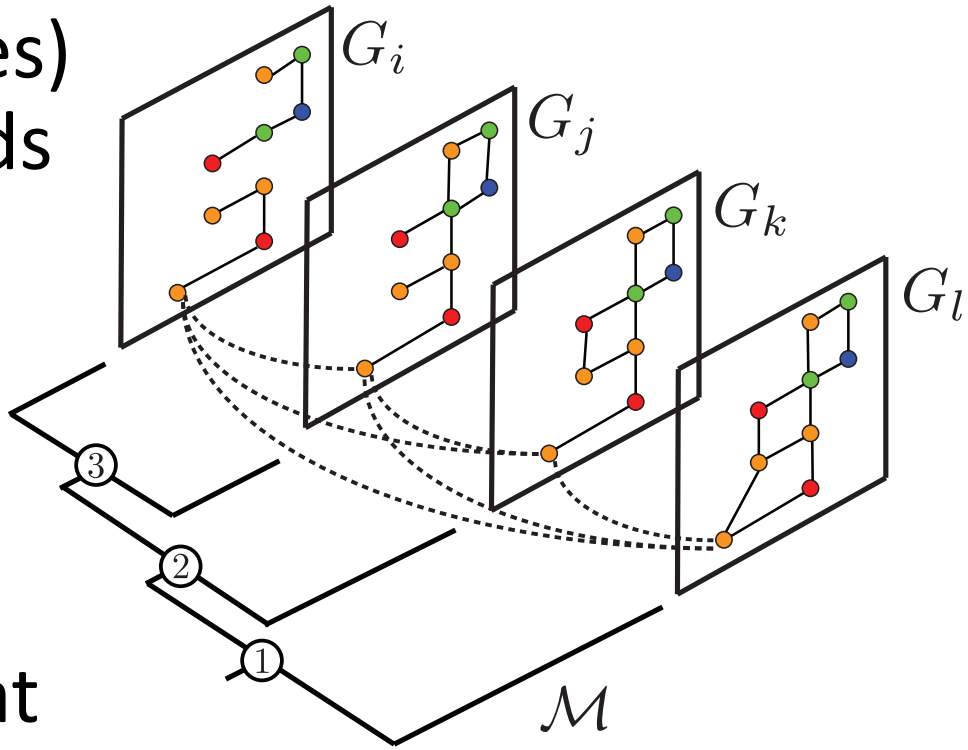


Tissue-specific protein interaction networks

- The precise function of proteins depends on their tissue context (Greene et al., Nat Genet 2015)
- Diseases result from the failure of tissue-specific processes (Hu et al., Nat Rev Genet 2016)
- Current models assume that protein functions are constant across tissues

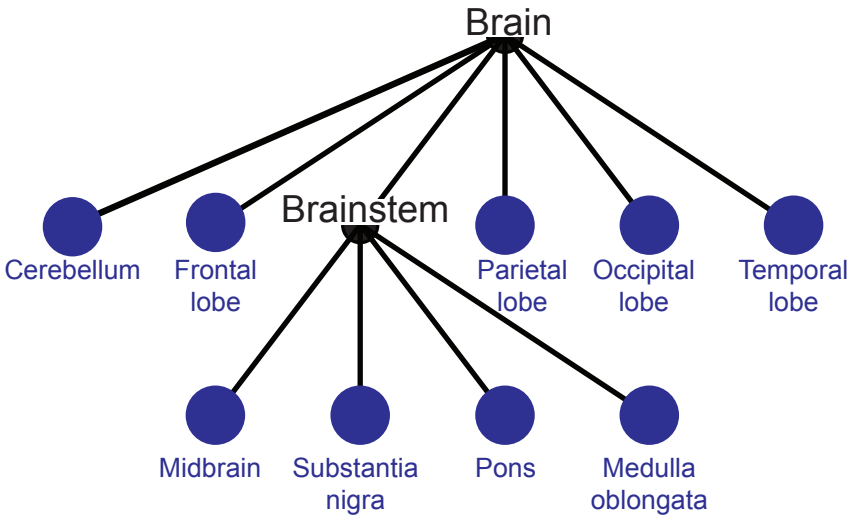
Multi-layer tissue network

- A multi-layer tissue network has many network layers (tissues)
- Each layer corresponds to one tissue-specific protein interaction network
- Hierarchy M encodes biological similarities between the tissues at multiple scales



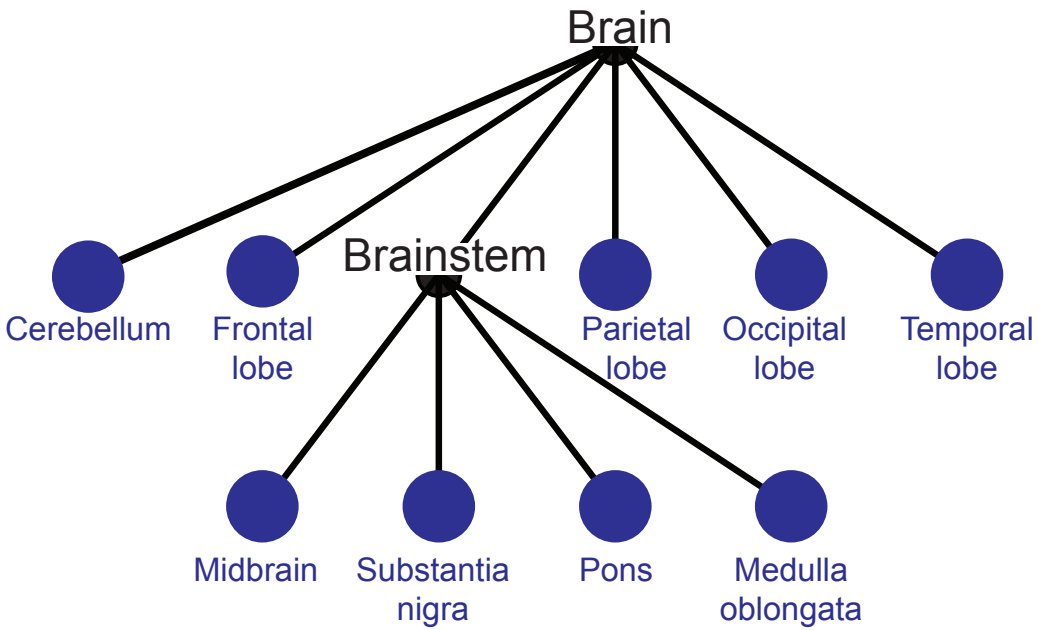
Experiments: Biological Nets

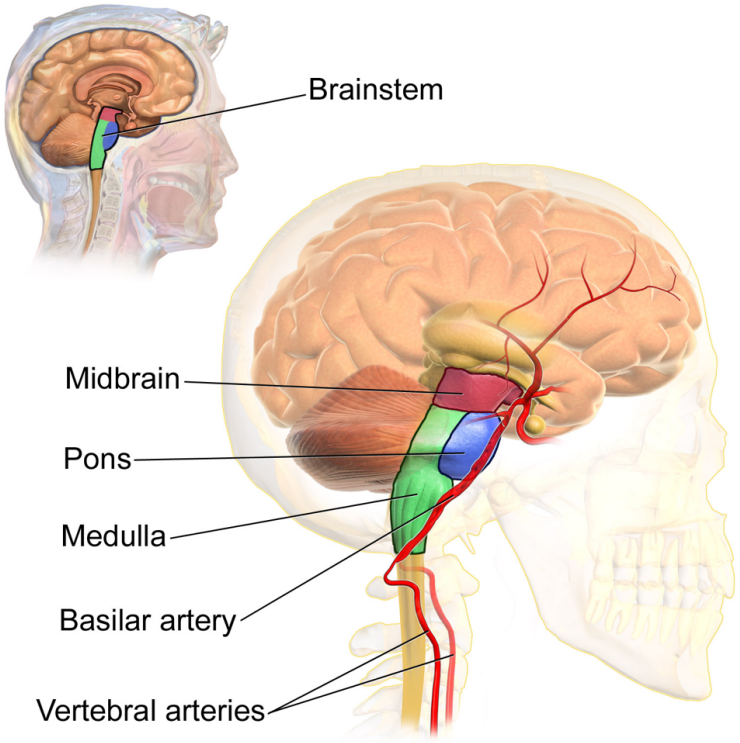
107 genome-wide tissue-specific protein interaction networks



- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - (renal cortex, cortex development)
  - (artery, pulmonary artery morphogenesis)

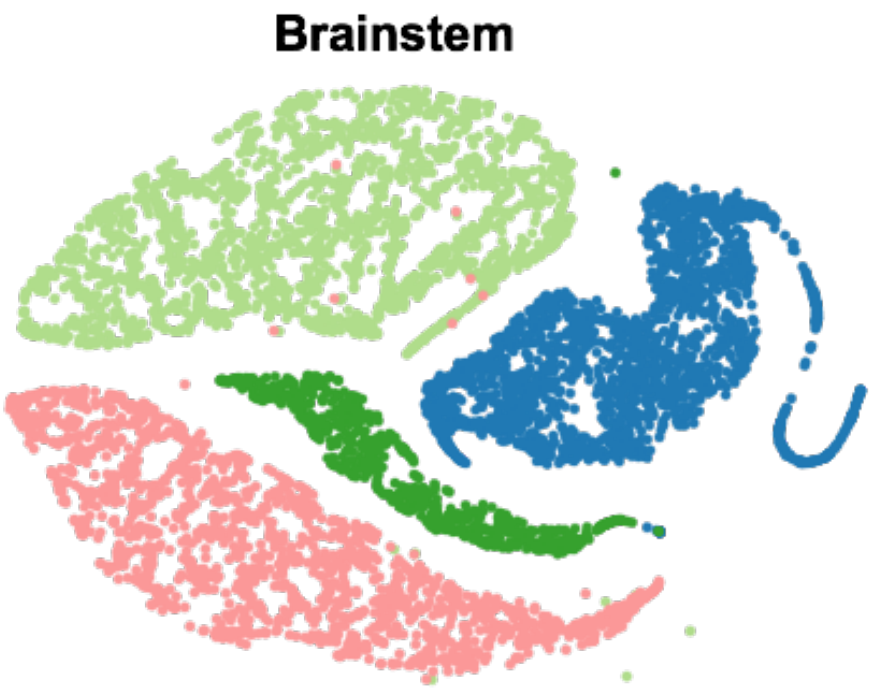
Brain Tissues

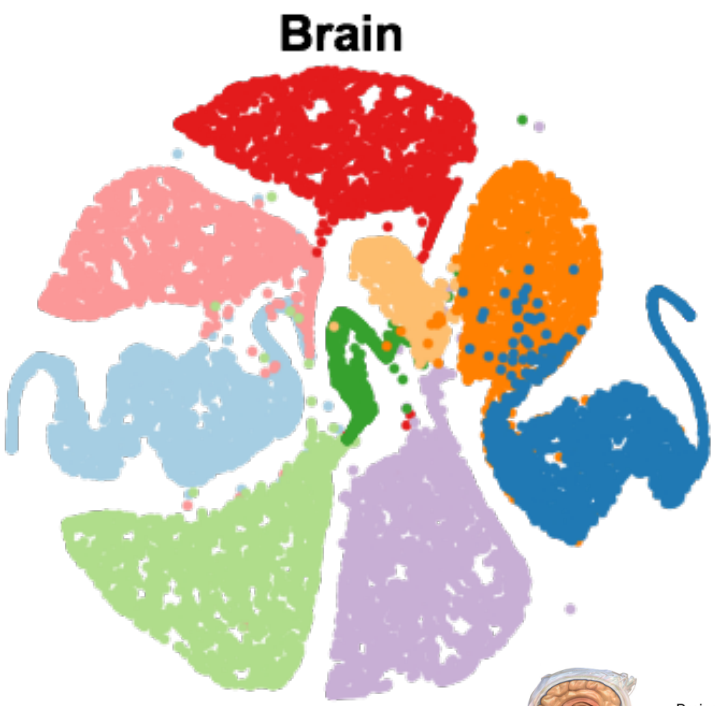




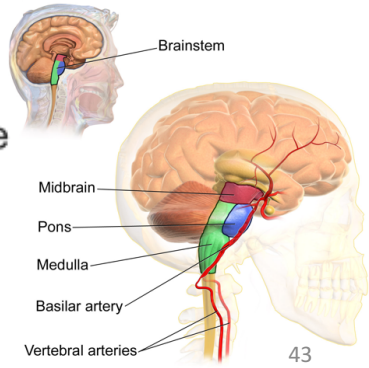
9 brain tissue PPI networks in two-level hierarchy

Meaningful Node Embeddings





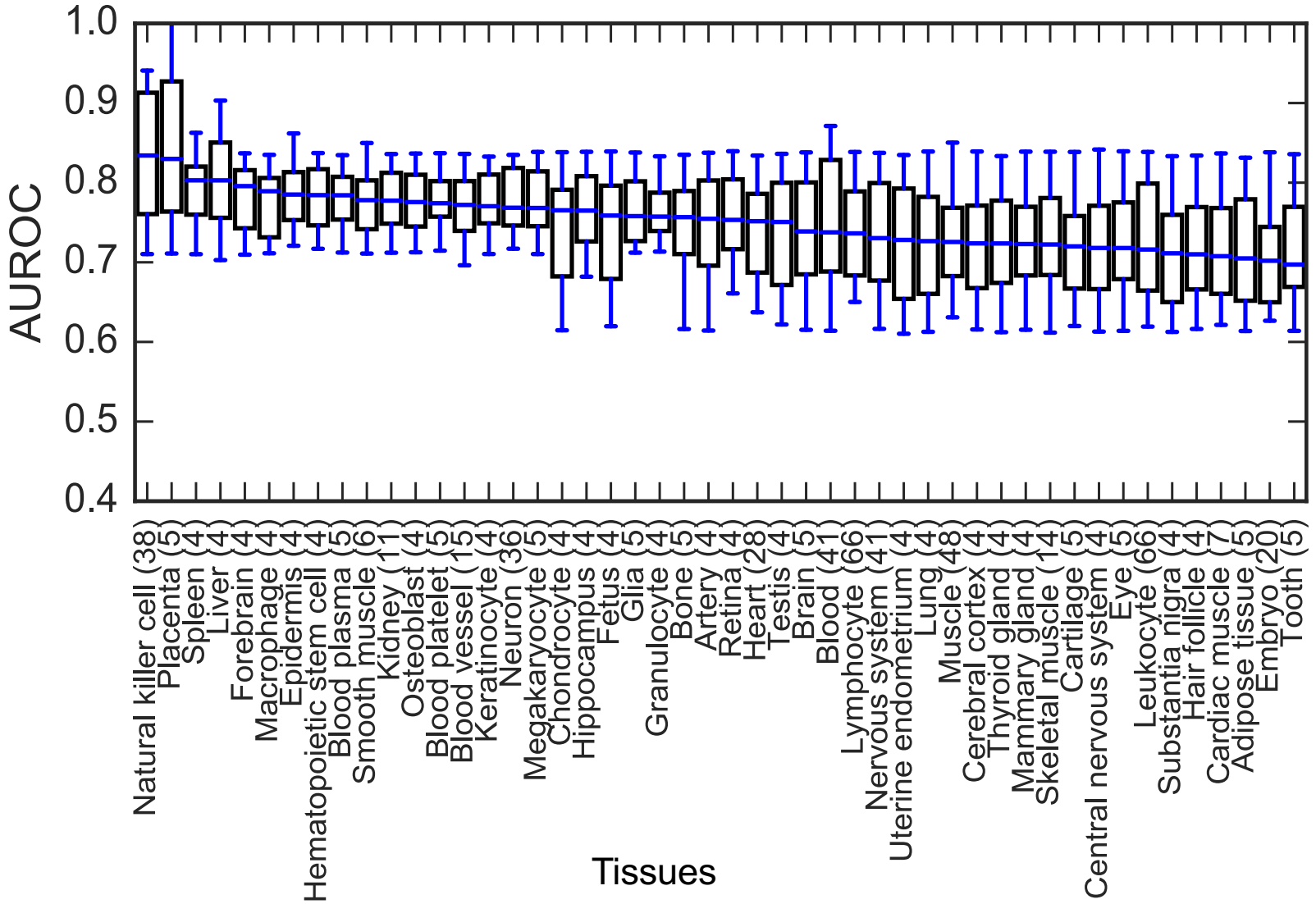
Cerebellum
Medulla oblongata
Substantia nigra
Frontal lobe
Temporal lobe
Pons
Parietal lobe
Occipital lobe
Midbrain



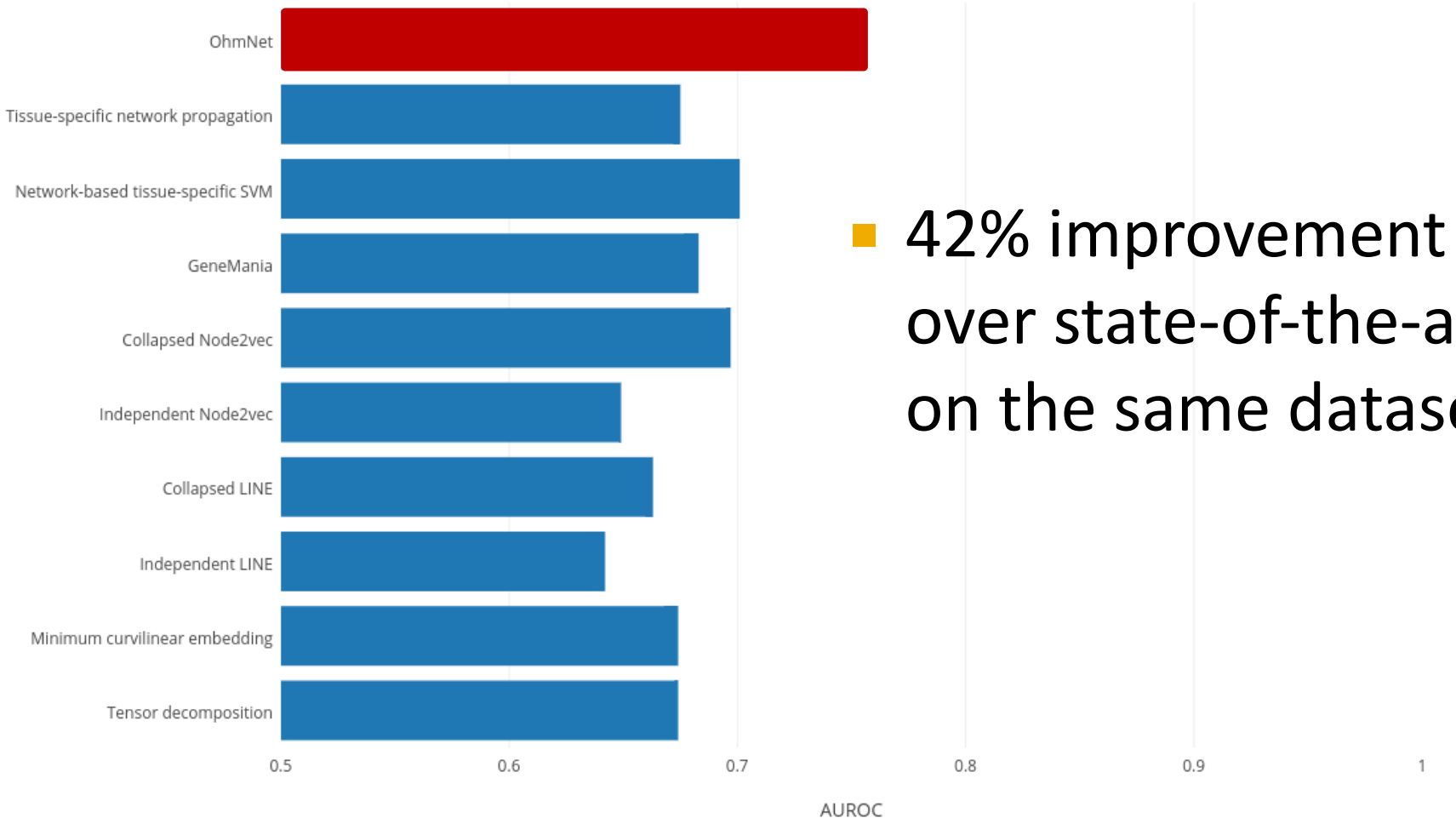
Experimental setup

- Cellular function prediction is a multi-label node classification task
- Every node (protein) is assigned one or more labels (cellular functions)
- Setup:
  - We apply OhmNet, which for every node in every layer learns a separate feature vector in an unsupervised way.
  - For every layer and every function we then train a separate one-vs-all regularized linear classifier using the modified Huber loss
  - During the training phase, we observe only a certain fraction of proteins and all their cellular functions across the layers
  - The task is then to predict the tissue-specific functions for the remaining proteins

Protein Function Prediction

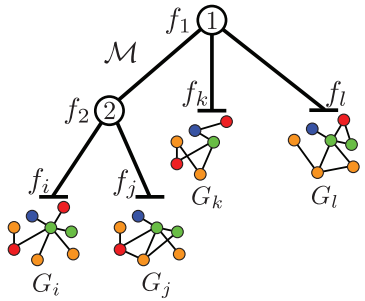


Protein Function Prediction



- 42% improvement over state-of-the-art on the same dataset

Transfer Learning



Transfer functions to unannotated tissues

- Task: Predict functions in target tissue without access to any annotation/label in that tissue

Target tissue	OhmNet	Tissue non-specific	Improvement
Placenta	0.758	0.684	**11%**
Spleen	0.779	0.712	**10%**
Liver	0.741	0.553	**34%**
Forebrain	0.755	0.632	**20%**
Blood plasma	0.703	0.540	**40%**
Smooth muscle	0.729	0.583	**25%**
Average	0.746	0.617	**21%**

Reported are AUC values