Stat 236 Final Project: Graph Neural Networks

Romil Sirohi, Franklyn Wang, Annie Zhu

December 12, 2020

Table of Contents

Graph Prediction

Many objects of study in the modern era are on graphs:

- Social Networks (e.g. coauthorship, Facebook, friends)
- Biology (e.g. proteins, drugs)
- Encapsulating relationships (e.g. objects in scenes, parse trees, link networks on the web)

Graphs provide a particularly nice illustration of sparsity. Many times even the quantity n^2 is too large to fit in memory, but the number of edges, m, does. This has led to a lot of work on sparse graphs, e.g. [Spielman and Teng, 2004].

Measuring understanding

- As explained previously, modelling the structure of graphs is a task of significant importance.
- However creating a task that can test for this is nontrivial because graphs don't usually lend themselves to bulk prediction tasks the same way images do, as it's difficult to find many graphs.
- To deal with this, the field has proposed a few interesting tasks to measure "graph understanding".

Commonly used tasks

- **Node Classification**, or predicting the labels of various nodes
- This is a specific instance of community detection, which we explored in class.

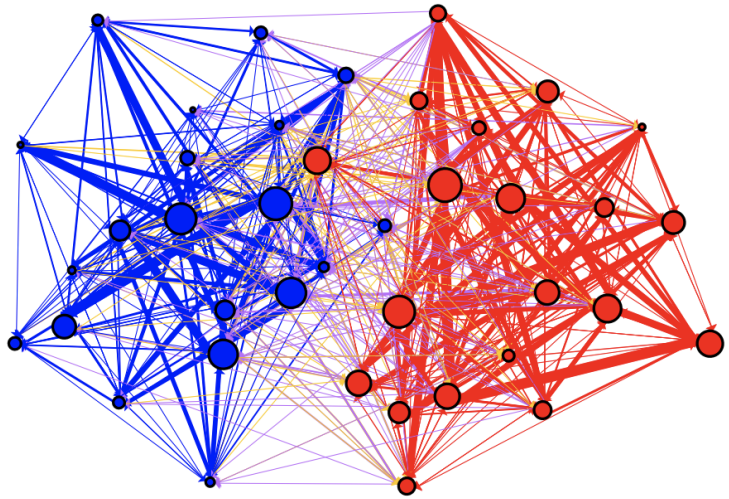


Figure: Political Blogging Network [Adamic and Glance, 2005]

Commonly used tasks

- **Link prediction**, or predicting links in a graph, is a very nice task. How do we test it though?
- One way is to remove information about links, and then attempt to predict that from the remaining links. However, the resulting networks are fundamentally *incomplete*.
- In a task originally introduced by [Liben-Nowell and Kleinberg, 2007] where the network (in this case the coauthorship network) evolves over time. This induces a particularly natural task, because we can attempt to predict the future from the present, and these are all complete networks.

Commonly used tasks

- **Graph Classification / Regression** is the task which bears the most similarity to standard supervised learning applications – it involves learning a function f so that

$$f(G_i) \approx y_i.$$

- An example is the recent advance by AlphaFold, which predicts protein folding structures.

AlphaFold

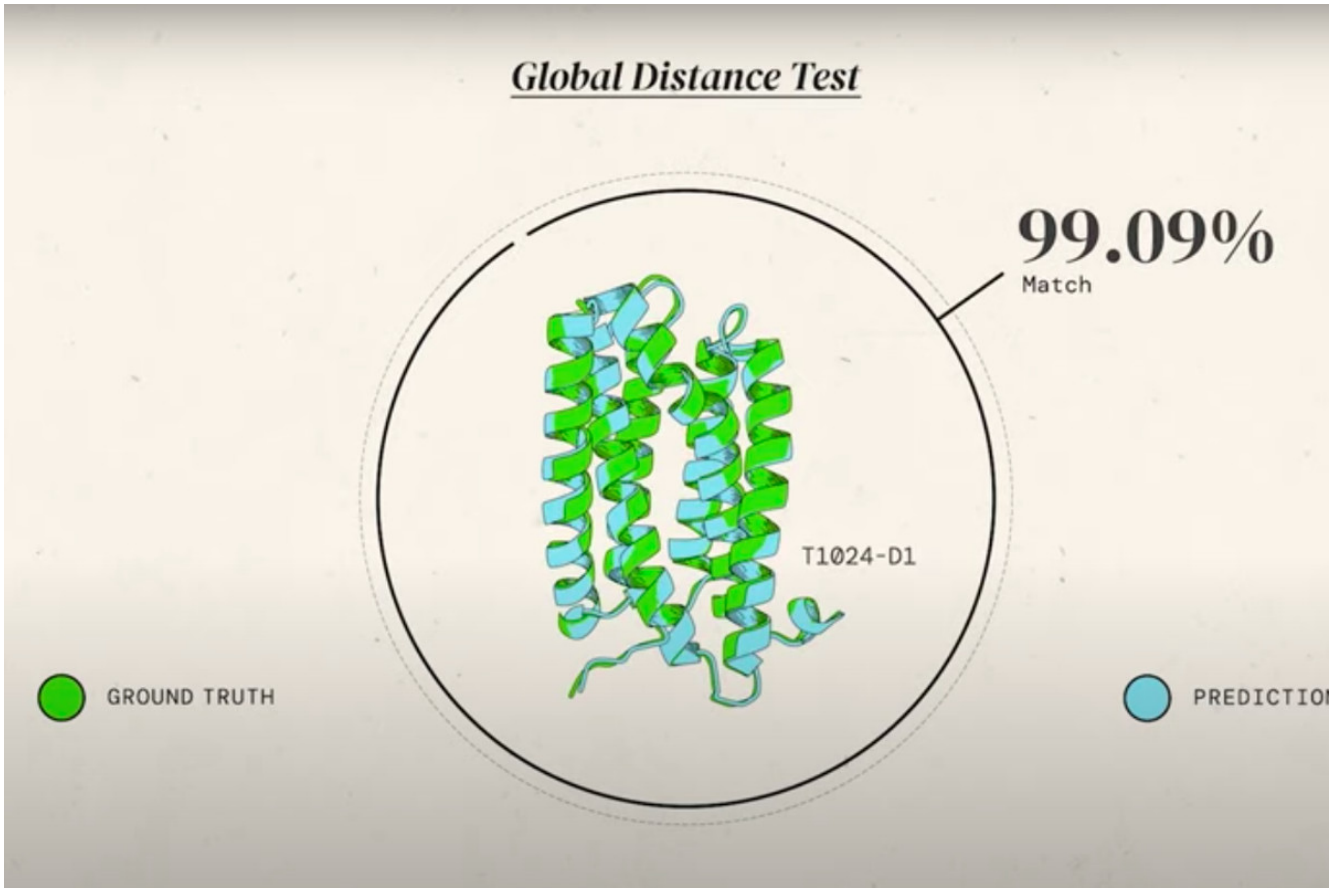
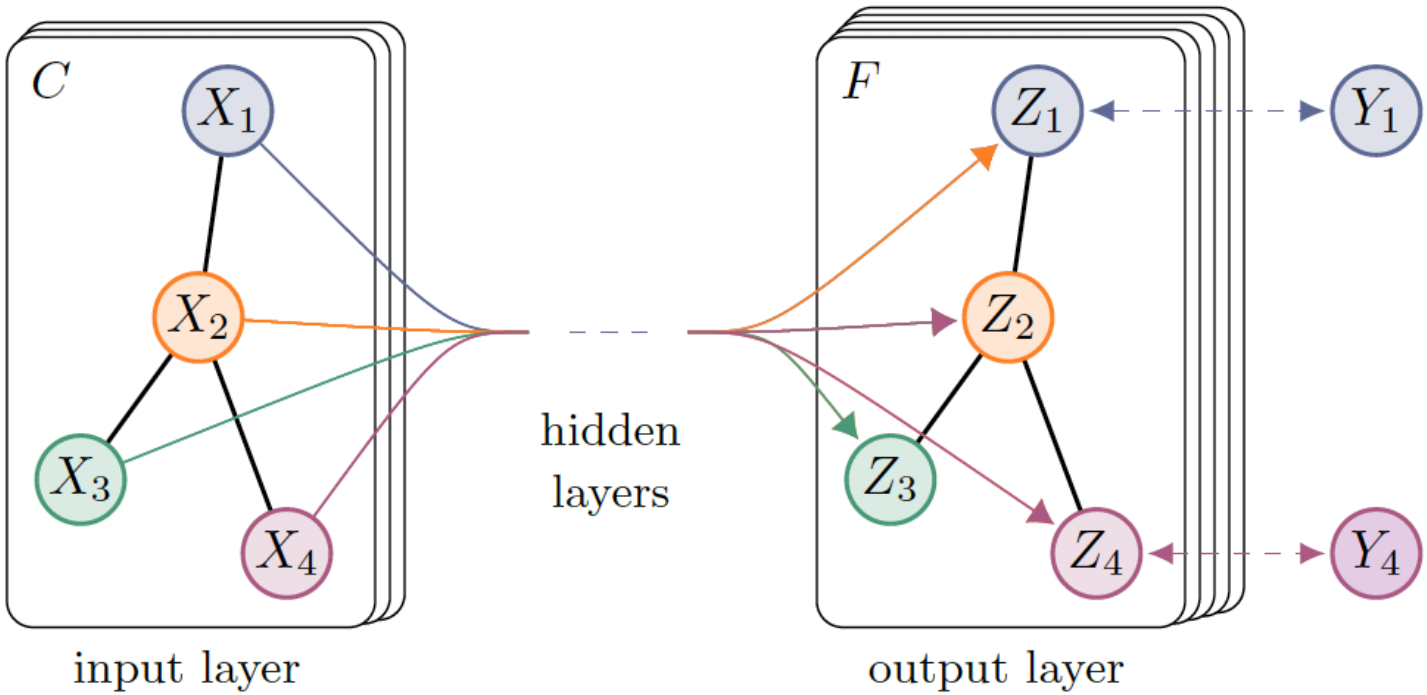


Figure: Protein Folding

Table of Contents

GNN formulation

In general, a convolutional GNN inputs a C features x_v per node. Through many layers of conv filters, we output a F-dim h_v *latent feature* per node, which we can use to perform the desired task.



Spectral Methods

- Spectral methods focus on the spectral properties of the graph.
- The principal object of study in spectral graph theory is the Laplacian matrix, which is

$$L = D - A$$

  where D is the degree matrix and A is the adjacency matrix. For example, the commute time heuristic can be naturally represented in terms of the spectrum of $D - A$. The eigenvectors of L have great significance in many ways.
- What if we could use this systematically?

Spectral Methods

- The standard convolutional network architecture

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} \Theta_{i,j}^{(k)} h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

Spectral Methods

- The standard convolutional network architecture

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} \Theta_{i,j}^{(k)} h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

- However, this formula does not incorporate any elements of a graph. Thus, we consider the modification

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)} U^{>} h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

where U is the Laplacian eigenbasis of the graph G. We can now make a sparsity assumption on Θ – it must be diagonal!

Algorithmic Difficulties

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)}U^{>}h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

- This product is difficult to evaluate quickly - note that calculating U of the graph requires $O(n^3)$ times. Thus, there are approximations based on Chebyshev polynomials to calculate this expression faster - this is known as ChebNet.
- When one uses Chebyshev polynomials, one can make an approximation with two steps - as it turns out, this is equivalent to the graph convolutional network.

Why spectral?

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)}U^{>}h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

- Consider feature vector $h \in \mathrm{R}^n$.
- Project into Fourier basis $\mathrm{Col}(U)$ to get $U^{>}h$.

Why spectral?

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)}U^{>}h_{:,i}^{(k\ \ 1)}\right), 1 \leq j \leq f_k$$

- Consider feature vector $h \in \mathrm{R}^n$.
- Project into Fourier basis $\mathrm{Col}(U)$ to get $U^{>}h$.
- Apply convolution with $g \in \mathrm{R}^n$

$$h *_G g = F^{\ \ 1}(F(h) \odot F(g)) = U(U^{>}h \odot U^{>}g)$$

Why spectral?

$$h_{:,j}^{(k)} = \tau\left(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)} U^{\top} h_{:,i}^{(k-1)}\right), 1 \leq j \leq f_k$$

- Consider feature vector $h \in \mathbb{R}^n$.
- Project into Fourier basis $\mathrm{Col}(U)$ to get $U^{\top}h$.
- Apply convolution with $g \in \mathbb{R}^n$

$$h *_G g = F^{-1}(F(h) \odot F(g)) = U(U^{\top}h \odot U^{\top}g)$$

- Reparametrize with $G_\theta = \mathrm{diag}(U^{\top}g)$

$$h *_G g = UG_\theta U^{\top}h$$

- G_θ is the diagonal matrix that parametrizes the convolutional filter.

From fully connected to graph neural networks

- In a fully connected neural network, we have that each element of layer $i+1$ - $x_{i+1,y}$ is a function of all $x_{i,y'}$.

From fully connected to graph neural networks

- In a fully connected neural network, we have that each element of layer $i+1$ - $x_{i+1,y}$ is a function of all $x_{i,y'}$.
- In a convolutional neural network, we have that $x_{i+1,y}$ is a function of a smaller set of $x_{i,y'}$, namely only y^0 which are close to y.

From fully connected to graph neural networks

- In a fully connected neural network, we have that each element of layer $i+1$ - $x_{i+1,y}$ is a function of all $x_{i,y'}$.
- In a convolutional neural network, we have that $x_{i+1,y}$ is a function of a smaller set of $x_{i,y'}$, namely only y^0 which are close to y.
- In a graph neural network, $x_{i+1,y}$ is a again a function of a smaller set of $x_{i,y'}$, except it has a particular structure: the nearby y^0 are those that are adjacent in the graph.

From fully connected to graph neural networks

- In a fully connected neural network, we have that each element of layer $i+1$ - $x_{i+1,y}$ is a function of all $x_{i,y'}$.
- In a convolutional neural network, we have that $x_{i+1,y}$ is a function of a smaller set of $x_{i,y'}$, namely only y^0 which are close to y.
- In a graph neural network, $x_{i+1,y}$ is a again a function of a smaller set of $x_{i,y'}$, except it has a particular structure: the nearby y^0 are those that are adjacent in the graph.
- Spectral methods are in fact also localized to a k-hop neighborhood, though in a less obvious way.

Spatial Graph Neural Network Definition

- Recall that for each node v in a graph, a GNN learns a latent representation h_v.

Spatial Graph Neural Network Definition

- Recall that for each node v in a graph, a GNN learns a latent representation h_v.
- Each h_v is initialized as x_v (which is possibly empty). In a spatial GNN, this is then updated according to the values of its neighbors at each iteration.

Spatial Graph Neural Network Definition

- Recall that for each node v in a graph, a GNN learns a latent representation h_v.
- Each h_v is initialized as x_v (which is possibly empty). In a spatial GNN, this is then updated according to the values of its neighbors at each iteration.
- For example, at the k-th iteration, we update it as

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k\ \ 1)} : u \in \mathcal{N}(v)\right\}\right),$$
$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k\ \ 1)}, a_v^{(k)}\right).$$

WL Test

We now present a test for isomorphism, the WL-test, which bears a strong relationship to GNNs.

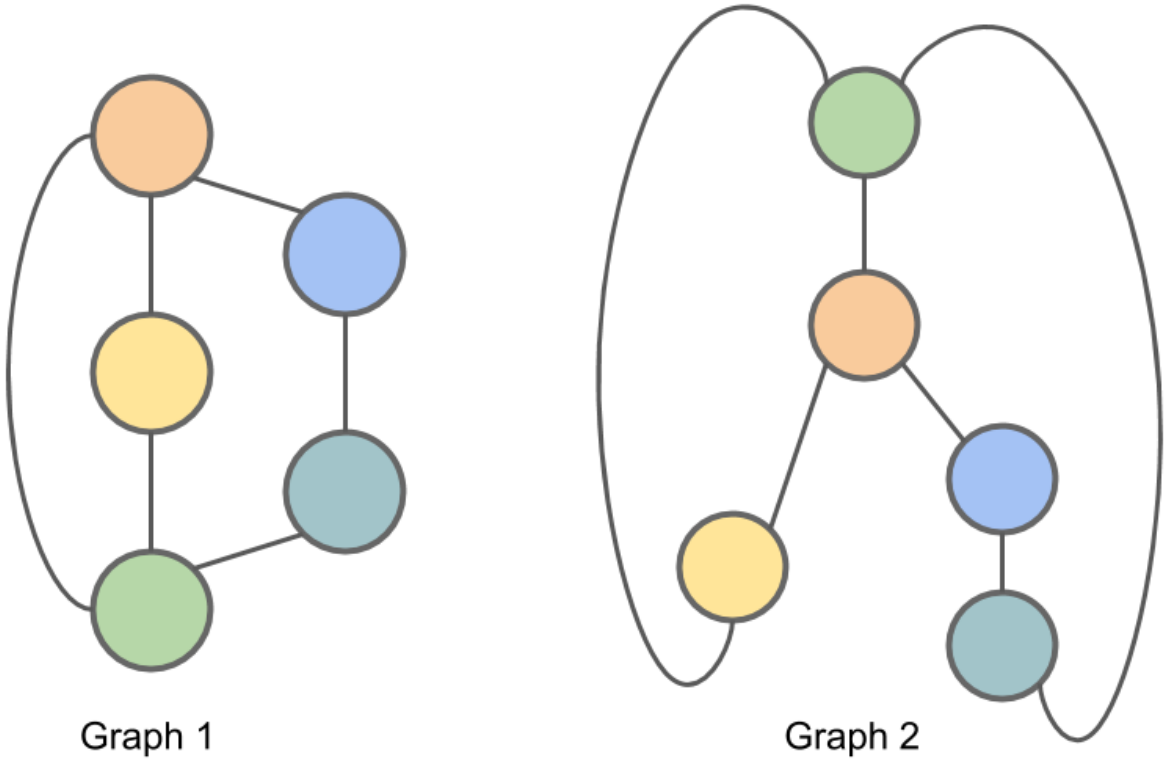
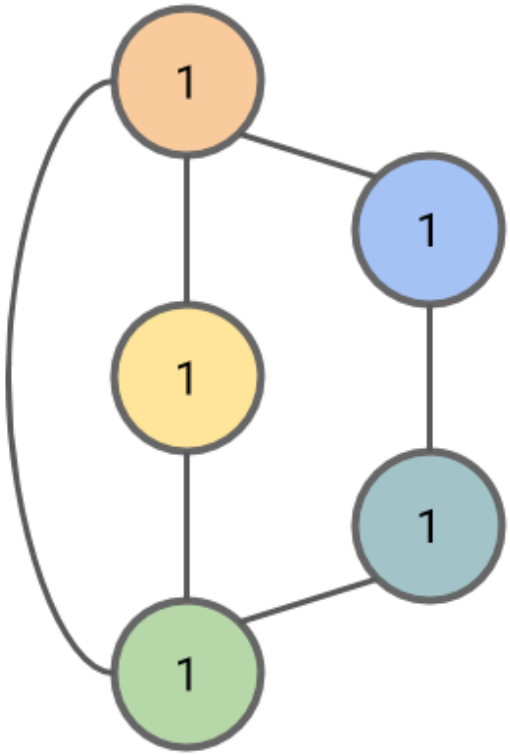


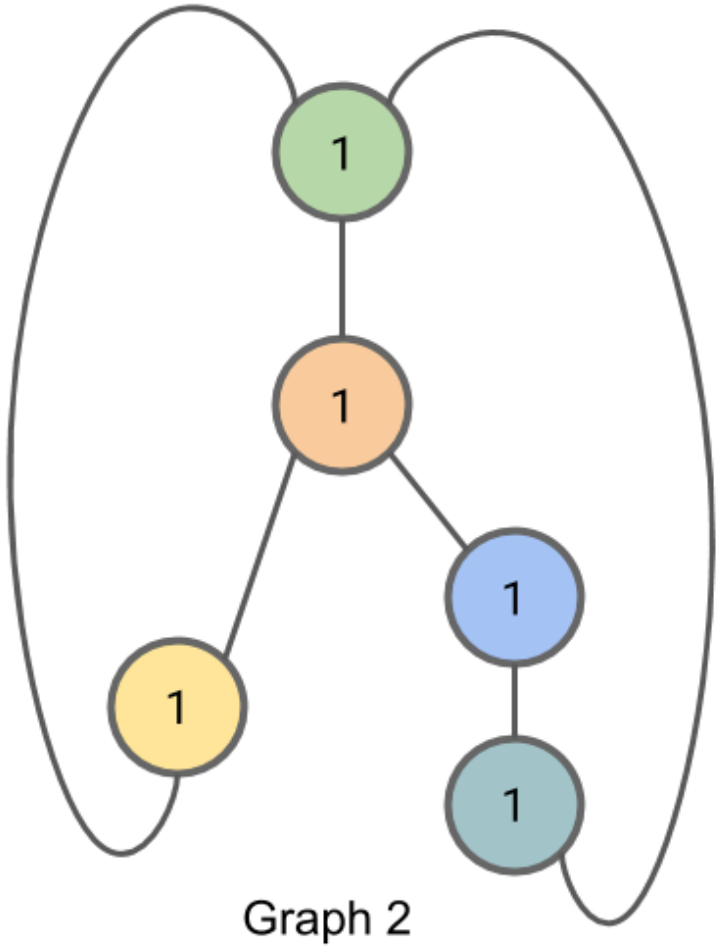
Figure: Two Graphs. Figures taken from `https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/`

WL Test

C_0



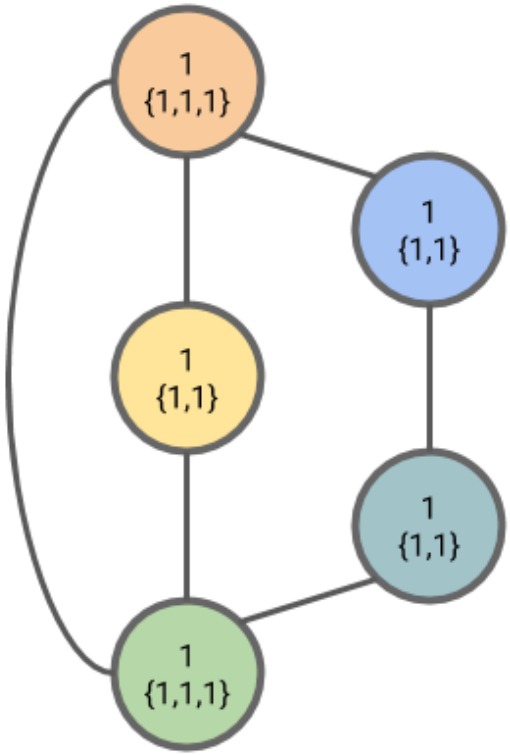
Graph 1



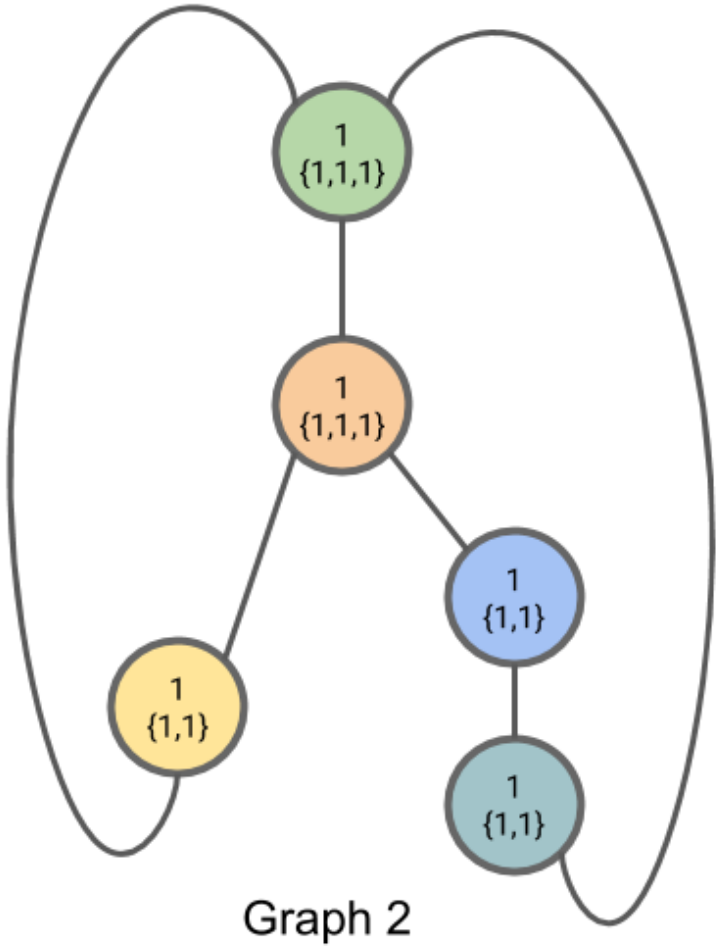
Graph 2

WL Test

L_1



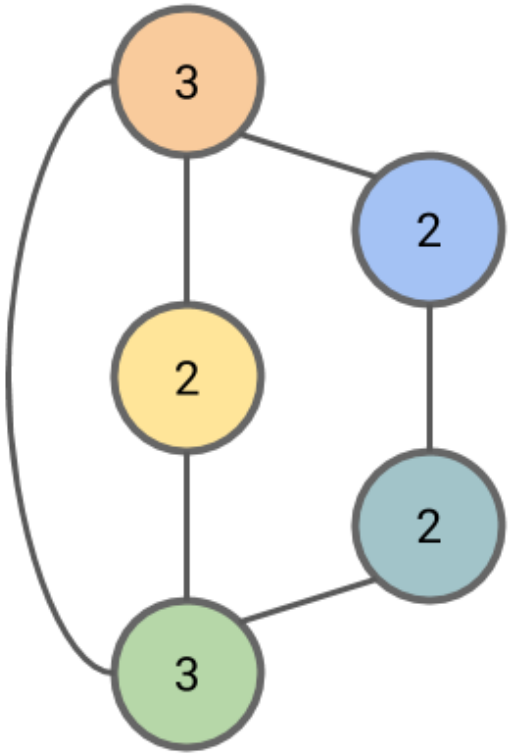
Graph 1



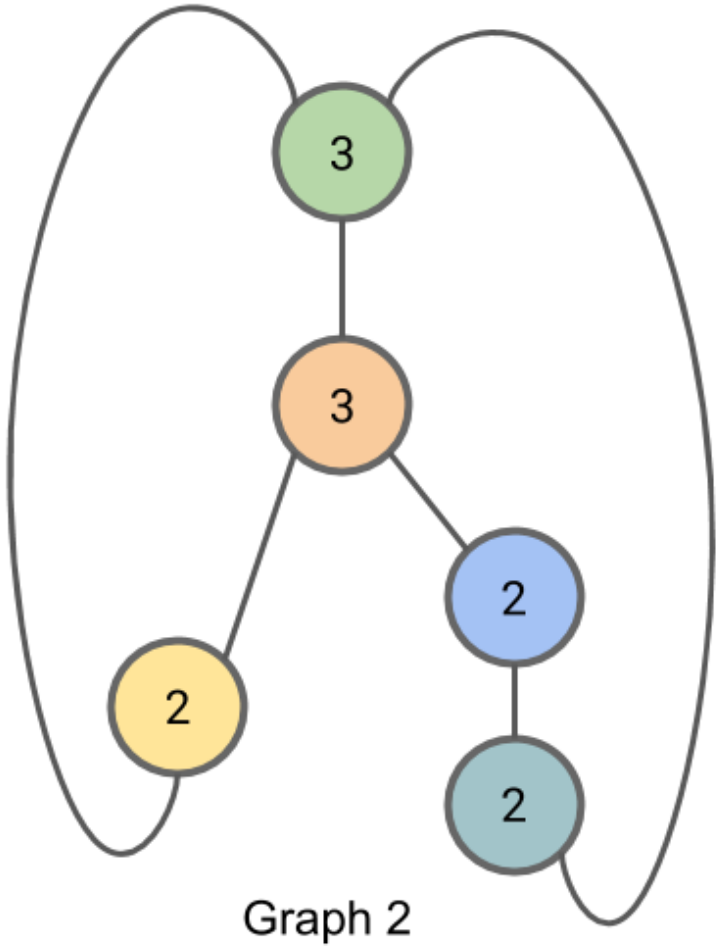
Graph 2

WL Test

C_1



Graph 1



Graph 2

The graph isomorphism network

- In fact, if the local structures of two nodes are the same, the latent representations will be identical. [Xu et al., 2019]

- What could go wrong with this?

A task in link prediction

The naive approach: for (u, v), use CONCAT(h_u, h_v).

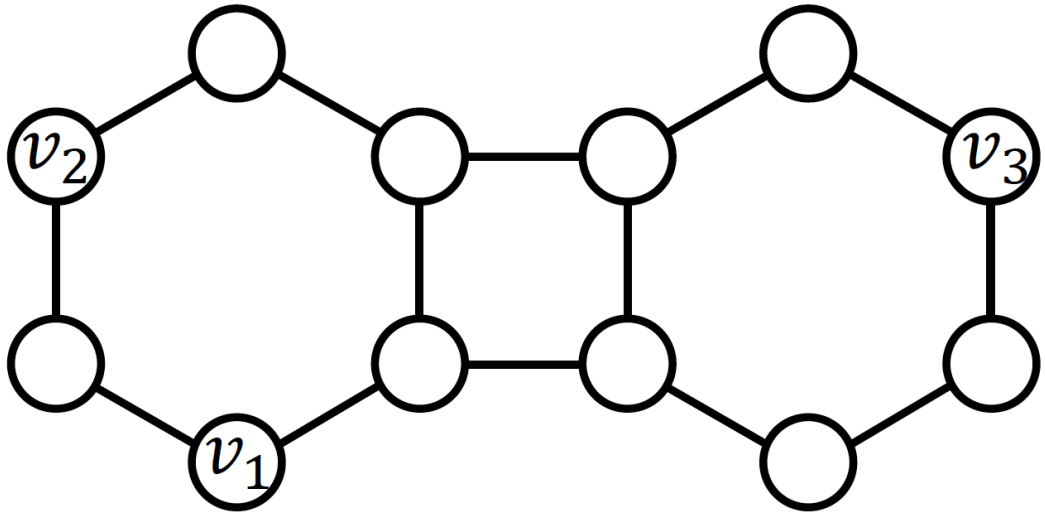


Figure: In this graph, $h_u = h_v$ by symmetry, so the naive link prediction of (v_2, v_1) is identical to that of (v_3, v_1)

SEAL: A method for link prediction

- Latent node representation $\rightarrow$ latent link representation.

SEAL: A method for link prediction

- Latent node representation $\rightarrow$ latent link representation.
- $L_{u,v} : V \rightarrow \mathrm{N}$ encodes *relative* location of nodes to (u, v).

SEAL: A method for link prediction

- Latent node representation $\rightarrow$ latent link representation.
- $L_{u,v} : V \rightarrow \mathrm{N}$ encodes *relative* location of nodes to (u, v).
- Concatenate $L_{u,v}(w)$ to GNN input as an add'l feature per node.
- Run GNN as usual.

Table of Contents

Correct and Smooth

- A recent work [Huang et al., 2020] combines classical methods as well as graph neural networks in a particularly nice framework. It starts with a set of base predictions on the training set, and then uses local smoothness to figure out the correct values.

Correct and Smooth

- A recent work [Huang et al., 2020] combines classical methods as well as graph neural networks in a particularly nice framework. It starts with a set of base predictions on the training set, and then uses local smoothness to figure out the correct values.

- Step 1: Generate base predictions
- Step 2: Correct these predictions by propagating errors along edges
- Step 3: Construct our best guesses by appealing to the training data again

Correct

- Initial error is $E_{L_t} = Z_{L_t} - Y_{L_t}$, $E_{fvg} = 0$ for other v.
- Intuition: The errors of models at similar nodes should be correlated.

- We can propagate the errors with

$$\hat{E} = \arg\min_{E'} \mathrm{tr}(E'^{T} \underbrace{(I - D^{-1/2} A D^{-1/2})}_{N_L} E^0) + \mu |E^0 - E|_F^2$$

- Our corrected predictions are $Z^c = Z + \sigma \hat{E} / |\hat{E}|_1$, where σ is the average error.

Smooth

- Intuition: The labels of at adjacent nodes should be correlated.
- Thus, we smooth our predictions with

$$\hat{Z} = \arg\min_{Z'} \mathrm{tr}(Z'^{T} \underbrace{(I - D^{-1/2} A D^{-1/2})}_{N_L} Z^{0}) + \mu |Z^{0} - Z^{c}|_F^2.$$

Table of Contents

Conclusion

- While Graph Neural Networks have shown great promise in recent years, their mechanisms for working are unclear. Some gains, like those in node classification, may be due to other things than simply "stronger representations".
- Graph Neural Networks often need assistance from other modules to "format" its output. When they do, however, the results can be quite strong. See [Karalias and Loukas, 2020] for an example.

References I

Adamic, L. A. and Glance, N. (2005).
The political blogosphere and the 2004 us election: divided they blog.
In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43.

Huang, Q., He, H., Singh, A., Lim, S.-N., and Benson, A. R. (2020).
Combining label propagation and simple models out-performs graph neural networks.
arXiv preprint arXiv:2010.13993.

Karalias, N. and Loukas, A. (2020).
Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs.
Advances in Neural Information Processing Systems, 33.

References II


Liben-Nowell, D. and Kleinberg, J. (2007).
The link-prediction problem for social networks.
Journal of the American society for information science and technology, 58(7):1019–1031.


Spielman, D. A. and Teng, S.-H. (2004).
Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.
In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90.


Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019).
How powerful are graph neural networks?
In *International Conference on Learning Representations*.