source code: https://github.com/ermongroup/graphite
powerpoint by guy who wrote the paper: https://www.youtube.com/watch?v=rxAwhwMCR4g

Graphite: Iterative Generative Modeling of Graphs

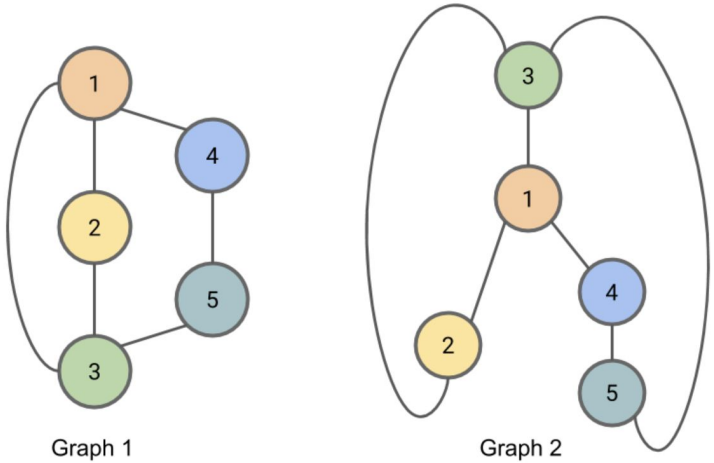
Tameez Latib, Mia Levy, Hengda Shi

Introduction

- Motivation
  - Most data is unlabeled, so let’s generate our own labels
    - Generative model for graph-structured data
  - Representation learning on large graphs
    - Graphite is particularly suited for representation learning on large graphs
  - Decoding low-dimensional latent variables into graphs is difficult
    - VAE for Images, text, etc are far easier because of their inherent structure
    - With graphs we want:
      - Permutation invariance (hard!)
      - Dynamic resizing
      - Local (to node) structure

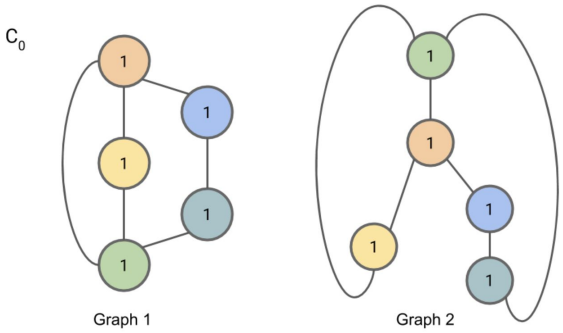
Background

- Weisfeiler-Lehman (WL) Algorithm
  - Test of graph isomorphism
    - Two graphs are considered isomorphic if there is a mapping between the nodes of the graphs that preserves node adjacencies

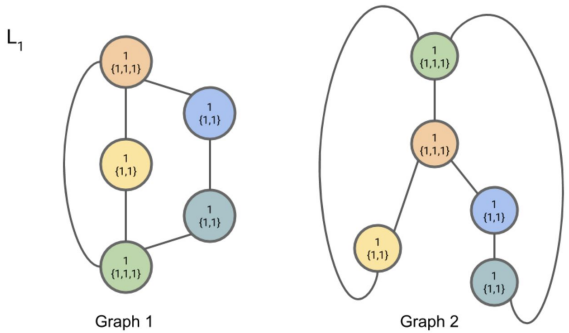


Background

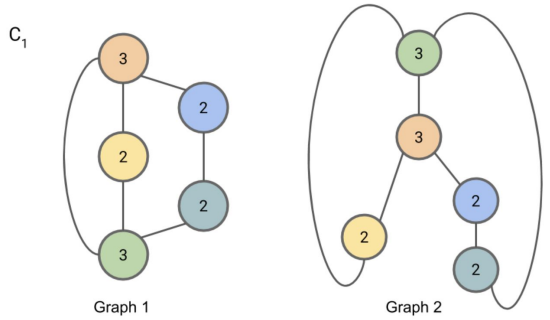
- Weisfeiler-Lehman (WL) Algorithm



Initialize all nodes to 1



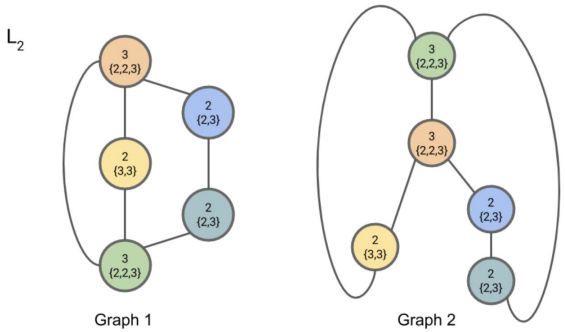
Collect values of neighboring nodes in a vector



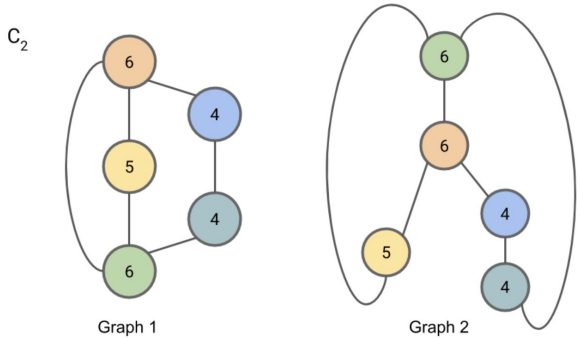
New node value = sum of the neighbor nodes

Background

- Weisfeiler-Lehman (WL) Algorithm



Collect values of neighboring nodes in a vector

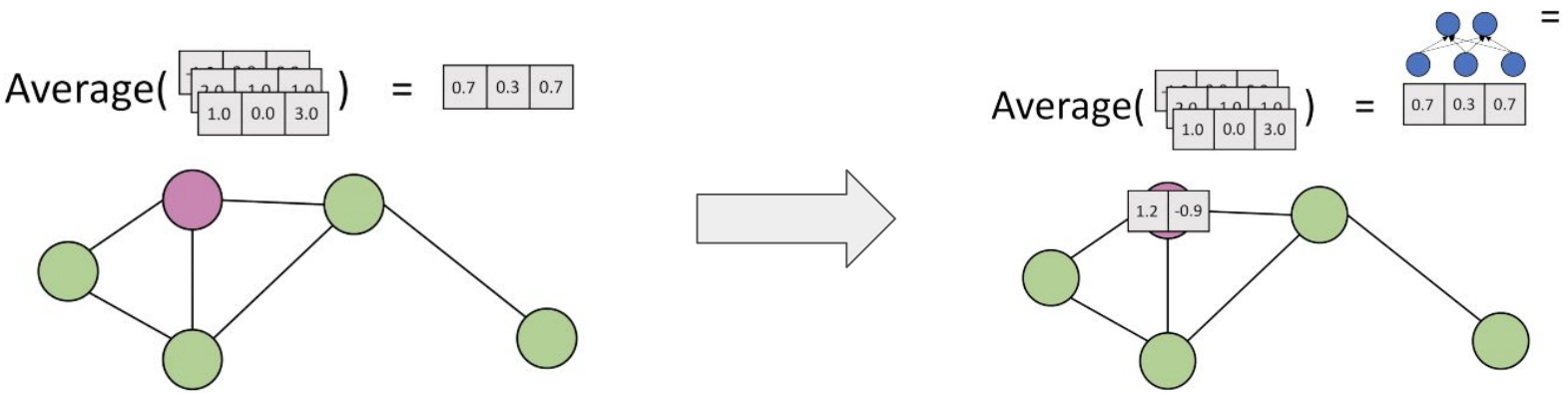


New node value = sum of the neighbor nodes

… Repeat until grouping of nodes, or label sets (i.e. “6” and “6” nodes change to “9” and “9”, but no more 9’s are added) stops changing

Background

- Graph Convolutional Networks (GCN)
  - Aggregates neighbors (this example uses average)
  - Passes aggregated neighbors through a neural network
  - Output of NN is the new node assignment
  - Repeat for every node



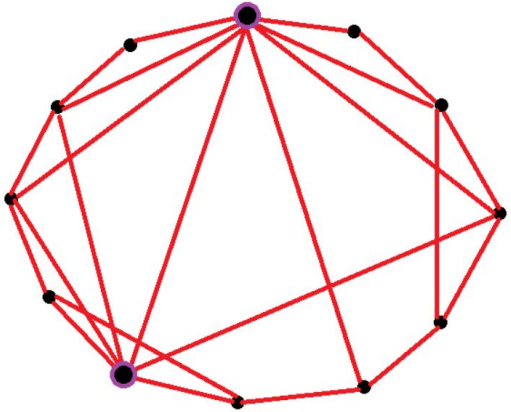
Background

- Graph Convolutional Networks (GCN) update rule
  - $H^{(l)}$: lth hidden layer
  - η_l: lth activation
  - B_l: Bias parameters
  - $H^{(l-1)}$: Previous hidden layer
  - W_l: Learnable weight parameters
  - $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$
    - where D is the degree matrix, A is the adjacency matrix

$$\mathbf{H}^{(l)} \leftarrow \eta_l \left(\mathbf{B}_l + \tilde{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}_l \right)$$

Related Work - probabilistic modeling (on graphs)

- Create edge in graph with constant probability (Erdos & Renyi, 1959).
  - This is too restrictive!
- Small-world model (Watts & Strogatz, 1998)
  - Has local clustering
  - Sparse adjacency matrix
  - Two neighbors of node are highly likely to also be neighbors



- Barabasi-Albert models (Barabasi & Albert, 1999)
  - Nodes with high degree more likely to form edges with new nodes
- However, these all generate graphs without learning!
  - Do not learn from data, so graphs cannot have structure of data

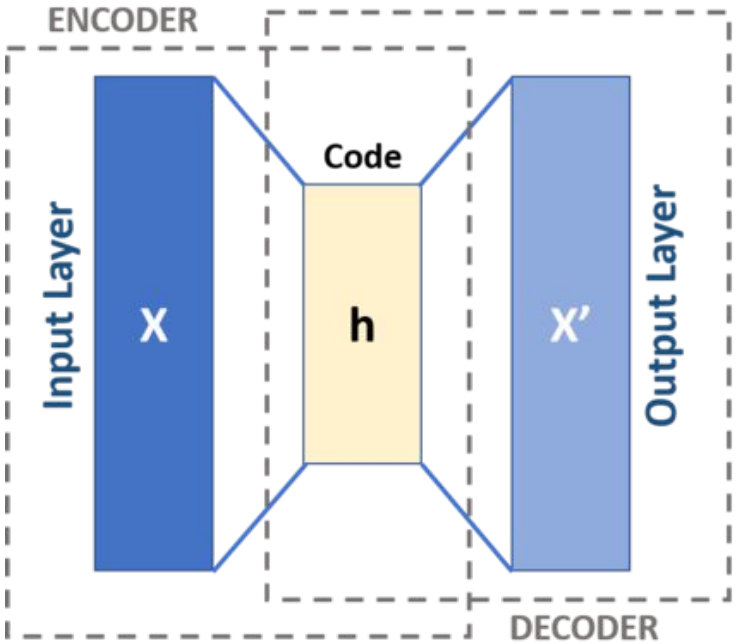
Related Work - representation learning (on graphs)

- Matrix factorization
  - Use graph laplacians and other tools
  - Computationally expensive for larger graphs
- Random walk approaches
  - Linearize graph via a random walk, objective function similar to skip-gram (e.g. node2vec)
  - Good at semi-supervised node classification
- graph neural networks
  - In this paper, Graph Conv Nets used
    - Similar to GAE/VGAE for more direct comparison

Related Work - latent variable models (for graphs)

- Bayesian models with deep neural nets (Hu et al., 2017; Wang et al., 2017).
  - Very restrictive and do not generalize well
    - Task specific; large inference cost; etc
- Adversarial generative approaches (Bojchevski et al. 2018; Li et al. 2018)
  - Generate graphs, but do not have encoding feature
- (Jin et al., 2018; Samanta et al., 2018; Simonovsky & Komodakis, 2018) have latent variable model, but does not scale well
  - Expensive decoding procedure
- GAE/VGAE
  - Graphite has new iterative decoding mechanism

Autoencoder



- An autoencoder is a structure that aims to reconstruct the original input via transformation.
- There are many applications for the autoencoder (dimensionality reduction, denoising).
- However, normal autoencoder is not a generative model which cannot generate fake data.
- Instead of using a normal autoencoder, we will talk about **variational autoencoder** which is a generative model.

KL Divergence

- In information theory, the amount of information is defined as:

$$I = -\log p(x)$$

- The expectation of information is also known as entropy:

$$H = -\sum p(x) \log p(x)$$

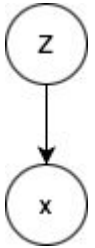
- The KL divergence between two distributions p and q is then:

$$\begin{aligned} D_{KL}(p||q) &= -\sum p(x) \log q(x) + \sum p(x) \log p(x) \\ &= \sum p(x) \log \frac{p(x)}{q(x)} \\ &= -\sum p(x) \log \frac{q(x)}{p(x)} \end{aligned}$$

Note 1: $D_{KL}(p||q) \geq 0$

Note 2: $D_{KL}(p||q) \neq D_{KL}(q||p)$

Variational Inference



- We often want to compute the **conditional density** of **latent variables z** given **observed variables x** (i.e. $p(\mathbf{z}|\mathbf{x})$).
- We know that from bayes rule, $p(\mathbf{z}|\mathbf{x})$ can be expressed as:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- The main obstacle in computing $p(\mathbf{z}|\mathbf{x})$ is that the marginal density of the observation $p(\mathbf{x})$ is intractable to compute:

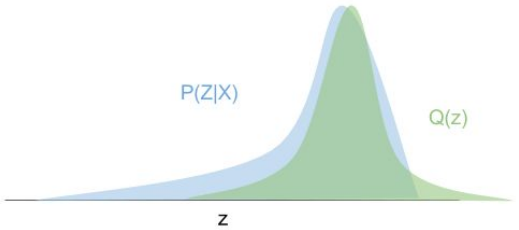
$$p(\mathbf{x}) = \int p(\mathbf{z}, \mathbf{x}) d\mathbf{z}$$

Note: Since **z** is a set of latent variables, the integral can be nested many times when **z** is in high dimension.

Variational Inference

- We cannot compute $p(\mathbf{z}|\mathbf{x})$ directly because it is intractable. Approximating the conditional density seems to be the only option.
- There are two main approaches in approximating it:
  - Monte Carlo (Gibbs Sampling)
  - Variational Inference
- Variational Inference turns the computation of $p(\mathbf{z}|\mathbf{x})$ into an optimization problem.
- Specifically, we would imagine an arbitrary distribution $q(\mathbf{z}|\mathbf{x})$, and then minimize the KL divergence between $p(\mathbf{z}|\mathbf{x})$ and $q(\mathbf{z}|\mathbf{x})$:

$$D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$$



Variational Inference

- Even though we know we could minimize $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$, we still seem to have no clue how to optimize this equation. Let's break it down with the definition of KL divergence.

$$
\begin{aligned}
D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}) &= -\sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} \\
&= -\sum q(\mathbf{z}|\mathbf{x}) \log \frac{\frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}}{q(\mathbf{z}|\mathbf{x})} \\
&= -\sum q(\mathbf{z}|\mathbf{x}) [\log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} + \log \frac{1}{p(\mathbf{x})}] \\
&= -\sum q(\mathbf{z}|\mathbf{x}) [\log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} - \log p(\mathbf{x})] \\
&= -\sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} + \sum q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}) \\
&= -\sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} + \log p(\mathbf{x})
\end{aligned}
$$

ELBO (Evidence Lower BOund)

- Rearranging the terms, we can get the following equation:

$$\log p(\mathbf{x}) = D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))+\sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}$$

- Note that **x** is observed, $\log p(\mathbf{x})$ is then a constant.
- $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x}))$ is always positive by definition.
- The lower bound of the evidence $p(\mathbf{x})$ is then:

$$ELBO = \sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}$$

- Instead of minimizing the KL divergence, we will maximize the evidence lower bound (ELBO) because $\log p(\mathbf{x})$ is a constant.

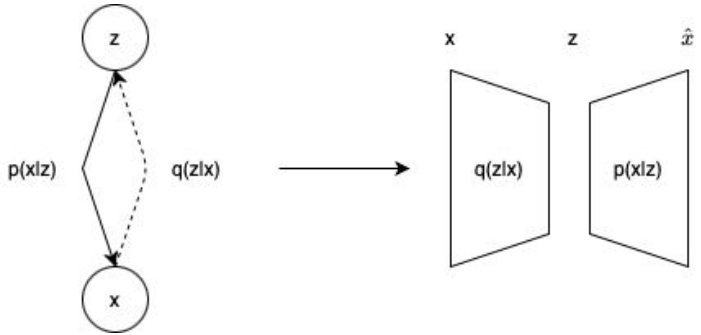
ELBO (Evidence Lower BOund)

- Keep deriving the ELBO, we can get the following objective:

$$
\begin{aligned}
ELBO &= \sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \\
&= \sum q(\mathbf{z}|\mathbf{x}) [\log p(\mathbf{x}|\mathbf{z}) + \log \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}] \\
&= \sum q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}|\mathbf{z}) + \sum q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}) - D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))
\end{aligned}
$$

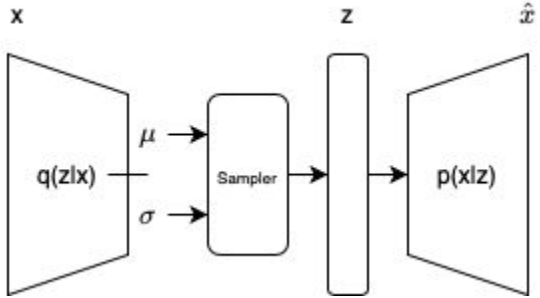
Variational Autoencoder

- Now we are trying to relate everything to autoencoder. In the original graphical model, we are interested in $p(\mathbf{z}|\mathbf{x})$. Now that we fake it with an arbitrary distribution $q(\mathbf{z}|\mathbf{x})$.
- This graphical model can then be turned into an autoencoder. Given the input data **x**, we try to compute **z** using a function $q(\mathbf{z}|\mathbf{x})$, and given the latent representation **z**, we try to reconstruct **x** using a function $p(\mathbf{z}|\mathbf{x})$.



Variational Autoencoder

- One key difference between the autoencoder and variational autoencoder is that the **actual latent representation z** is **sampled** from a certain distribution.
- i.e. we model our neural network output as the parameter of the distribution, and then sample the latent representation **z** from that distribution.
- Assuming our choice of distribution is isotropic Gaussian distribution meaning that the covariance matrix is diagonal, our neural network design will be:



Variational Autoencoder

- Going back to our objective function:

$$
\begin{aligned}
J &= -ELBO \\
&= -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}) + D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))
\end{aligned}
$$

- The first part is actually the L2 loss of original data and reconstructed data under Gaussian distribution or it would be similar to cross entropy loss under Bernoulli distribution.

$$
\begin{aligned}
\log P(\mathbf{x}|\mathbf{z}) &= \log P(\mathbf{x}|\hat{\mathbf{x}}) \\
&= \log e^{-|\mathbf{x}-\hat{\mathbf{x}}|^2} \\
&= -|\mathbf{x}-\hat{\mathbf{x}}|^2
\end{aligned}
$$

Variational Autoencoder

- The second part $D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$ has a closed form assuming $p(\mathbf{z})$ is a unit Gaussian distribution $N(0,1)$.
- Other than the special construction of neural network, the loss function of variational autoencoder is not simply $\mathcal{L} = |\mathbf{x} - \hat{\mathbf{x}}|$, but:

$$\mathcal{L} = -\sum_{j=1}^{J} \frac{1}{2}\left[1 + \log\left(\sigma_i^2\right) - \sigma_i^2 - \mu_i^2\right] - \frac{1}{L}\sum_{l} E_{\sim q_\theta(z|x_i)}\left[\log p(x_i|z^{(i,l)})\right]$$

- The first term is the simplified KL divergence and second term is the reconstruction loss
- The KL divergence term can also be interpreted as a regularization term, to encourage the approximate posterior to be close to the prior $p(\mathbf{z})$.

Check here for a full derivation.

Variational Autoencoder

- Note that when we model $q(\mathbf{z}|\mathbf{x})$, we often assume that $q(\mathbf{z}|\mathbf{x})$ can be factorized as:

$$q(\mathbf{z}|\mathbf{x}) \approx \prod_i q(\mathbf{z}_i|\mathbf{x})$$

- This assumption makes each $\mathbf{z}_i$ independent of each other. It also restricts the expressive power of the model because it becomes incapable of modeling the correlation between different $\mathbf{z}_i$.

Variational Graph Autoencoder

- An important prior work of this paper is the Variational Graph Autoencoder paper (the original VAE is also a work from their group).
- In graph data, things can be a little different. For an undirected, unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined to model the connection among edges, a node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$ is defined to keep the features of each node.
- The encoder uses mean field approximation and assumes the distribution to be isotropic Gaussian.

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \operatorname{diag}(\boldsymbol{\sigma}_i^2))$$

Variational Graph Autoencoder

- The encoder or inference model is constructed as a two layer GCN, on the first layer, weights are shared for the mean GCN and stddev GCN. On the second layer, two parallel GCN blocks would output the mean and stddev.

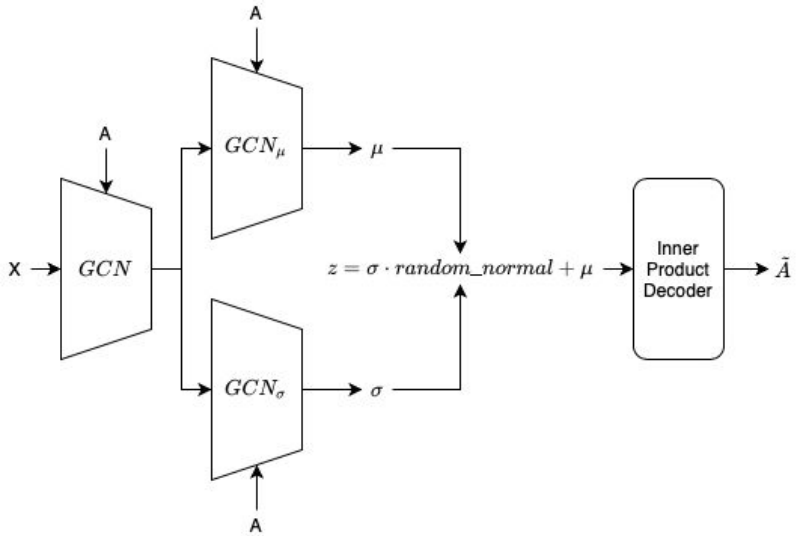
$$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A}) \qquad \log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$$

- The decoder or the generative model however is a simple **inner product** decoder.
- i.e. the decoder does not have any learnable parameters.

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j) \ , \quad \text{with} \quad p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^{\top} \mathbf{z}_j)$$

Variational Graph Autoencoder

- VGAE is only capable of generating model structure. The reconstructed matrix is only a reconstructed adjacency matrix. It is incapable of generating node feature matrix.



Iterative Generative Modeling via Graphite

- In Graphite, the encoding step is identical to the VGAE (even the code implementation for the encoding step is inherited from the VGAE implementation).
- The major contribution is in the decoding step.
- In the original VGAE, GCN cannot be used because A is no longer available in the decoding step.
- In Graphite, the author tries to construct an **intermediate** adjacency matrix $\hat{\mathbf{A}}$ from latent representation $\mathbf{Z}$, and use GCN to generate a refined latent representation $\mathbf{Z}^*$.
- The final reconstructed adjacency matrix $\tilde{\mathbf{A}}$ is then decoded using inner product of the refined latent representation $\mathbf{Z}^*$.

Iterative Generative Modeling via Graphite

- Specifically, the intermediate adjacency matrix is constructed with the following equation:

$$\widehat{\mathbf{A}} = \frac{\mathbf{Z}\mathbf{Z}^T}{\|\mathbf{Z}\|^2} + \mathbf{1}\mathbf{1}^T$$

- The first part is the normalized inner product that forms the basic form of the intermediate adjacency matrix. The 11^T is to ensure the adjacency matrix is nonnegative.
- The intermediate adjacency matrix will then be sent to the GCN along with the concatenation of latent representation $\mathbf{Z}$ and the node feature matrix $\mathbf{X}$:

$$\mathbf{Z}^* = \mathrm{GNN}_\theta(\widehat{\mathbf{A}}, [\mathbf{Z}|\mathbf{X}])$$

Iterative Generative Modeling via Graphite

- Further assume the generative model can be factorized as follows:

$$p_\theta(\mathbf{A}|\mathbf{Z},\mathbf{X}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p_\theta^{(i,j)}(\mathbf{A}_{ij}|\mathbf{Z}^*).$$

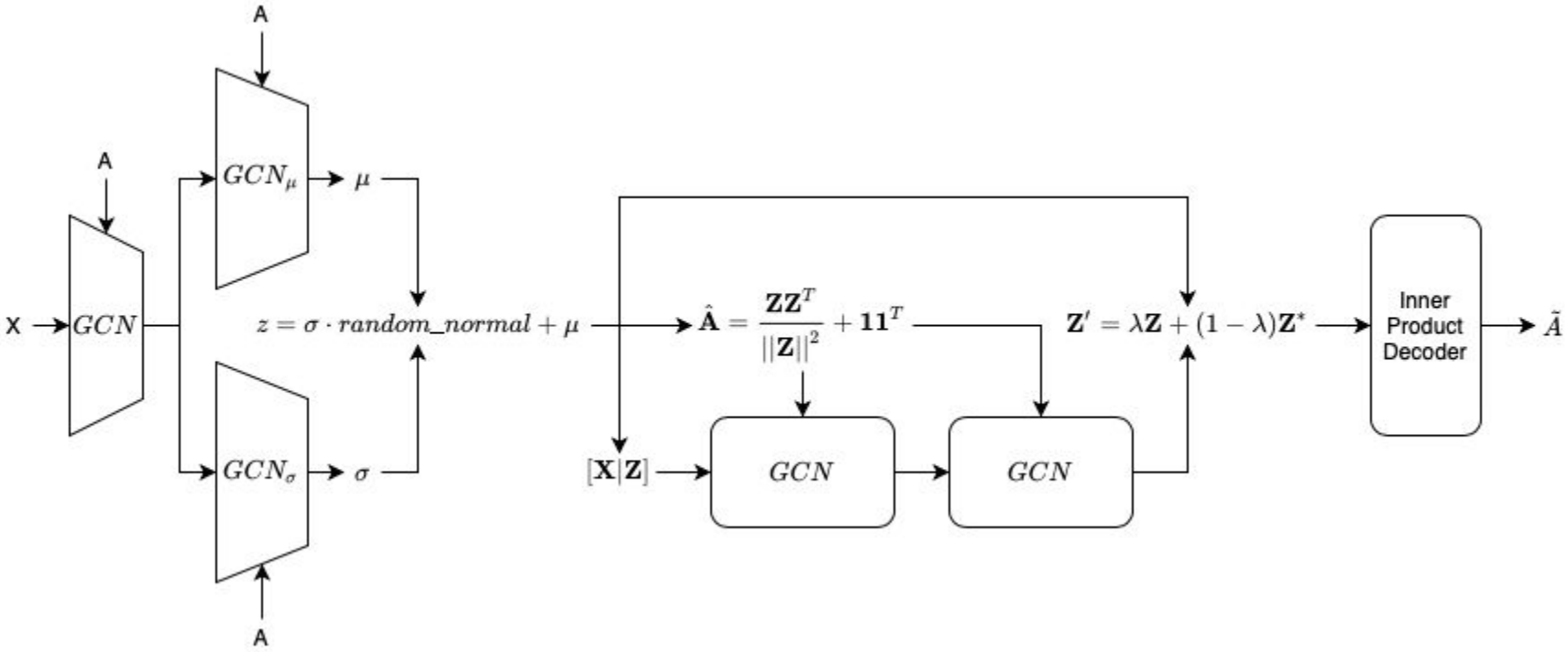
- For distribution on edge:
  - If edges are binary, use Bernoulli
  - Else if weighted, use Gaussian
- E.g. we can obtain the edge probabilities of an unweighted graph with:

$$\text{sigmoid}(\mathbf{Z}^*\mathbf{Z}^{*^T})$$

- One empirical result that the author did not show in the paper is that the final latent representation is a weighted sum of the original latent rep and the refined rep:

$$\mathbf{Z}' = \lambda\mathbf{Z} + (1-\lambda)\mathbf{Z}^*$$

General Structure of Graphite



Scalable learning and inference using Graphite

- In the actual implementation, they also made two modification to speed up the training and inference.
- The graph propagation rule in the decoding stage is:

$$\begin{aligned}\mathbf{H}^{(l)} &\leftarrow \eta_l(\hat{\mathbf{A}}\mathbf{H}^{l-1}) \\ &\leftarrow \eta_l((\frac{\mathbf{Z}\mathbf{Z}^T}{||\mathbf{Z}||^2}) + \mathbf{1}\mathbf{1}^T)\mathbf{H}^{l-1} \\ &\leftarrow \eta_l(\mathbf{Z}\frac{\mathbf{Z}^T\mathbf{H}^{l-1}}{||\mathbf{Z}||^2} + \mathbf{1}\mathbf{1}^T\mathbf{H}^{l-1})\end{aligned}$$

- It can then avoid the heavy inner product computation by first multiplying $\mathbf{Z}^T\mathbf{H}^{l-1}$. The time complexity is reduced from $O(n^2k)$ to $O(nkd_{l-1} + nd_{l-1}d_l)$.
- The second trick is to subsample the edges in loss computation to trade-off statistical accuracy with efficiency.

Experimental Evaluation

- Overview
  - 2 models tested, Graphite-VAE and Graphite-AE
  - For unweighted graphs
    - Minimize negative cross entropy between input adjacency matrix and reconstructed adjacency matrix
  - For weighted graphs
    - Minimize mean square error between input adjacency matrix and reconstructed adjacency matrix
  - Graphite outperforms most competing approaches for graph representation learning empirically for the tasks of density estimation (over entire graphs), link prediction, and semi-supervised node classification on synthetic and benchmark datasets

Experimental Evaluation

- Reconstruction and density estimation
  - Want to use standard reconstruction loss terms (like those in image based VAE)
  - However, graphs cannot be reduced to fixed number of vertices for input to GNN
  - To simplify, consider largest connected component to evaluate + optimize objective function
    - Set dimension for Z* (k*) to be max number of vertices
- Create dataset from 6 families
  - Erdos-Renyi, ego-nets, random regular graphs, random geometric graphs, random Power Law Tree and Barabasi-Albert.
  - For each family, have 300 graphs with 10-20 nodes. Evenly split for train/val/test sets
  - Benchmark is GAE/VGAE
    - Key difference: GAE/VGAE Decoder has no learnable parameters, so reconstruction is done through an inner product operation

Experimental Evaluation

- Reconstruction and density estimation: Results
  - Demonstrates usefulness of learned decoders in Graphite

Table 1. Mean reconstruction errors and negative log-likelihood estimates (in nats) for autoencoders and variational autoencoders respectively on test instances from six different generative families. Lower is better.

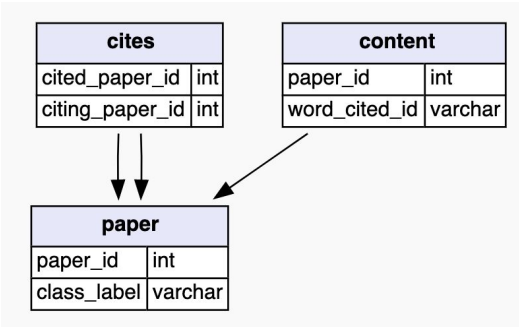
	Erdos-Renyi	Ego	Regular	Geometric	Power Law	Barabasi-Albert
GAE	221.79 ± 7.58	197.3 ± 1.99	198.5 ± 4.78	514.26 ± 41.58	519.44 ± 36.30	236.29 ± 15.13
Graphite-AE	**195.56** ± 1.49	**182.79** ± 1.45	**191.41** ± 1.99	**181.14** ± 4.48	**201.22** ± 2.42	**192.38** ± 1.61
VGAE	273.82 ± 0.07	273.76 ± 0.06	275.29 ± 0.08	274.09 ± 0.06	278.86 ± 0.12	274.4 ± 0.08
Graphite-VAE	**270.22** ± 0.15	**270.70** ± 0.32	**266.54** ± 0.12	**269.71** ± 0.08	**263.92** ± 0.14	**268.73** ± 0.09

Experimental Evaluation

- Link prediction
  - *“Given two nodes in a graph, does an edge exist between the nodes?”*
  - To evaluate: Feed in an incomplete graph, holding out 5% of the edges for validation and 10% for testing, train on the remaining subgraph
  - Datasets:
    - Cora, Citeseer, Pubmed
      - For Cora: “Each publication… is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary
      - Given node data, which other papers should be cited?

Table 2. Citation network statistics

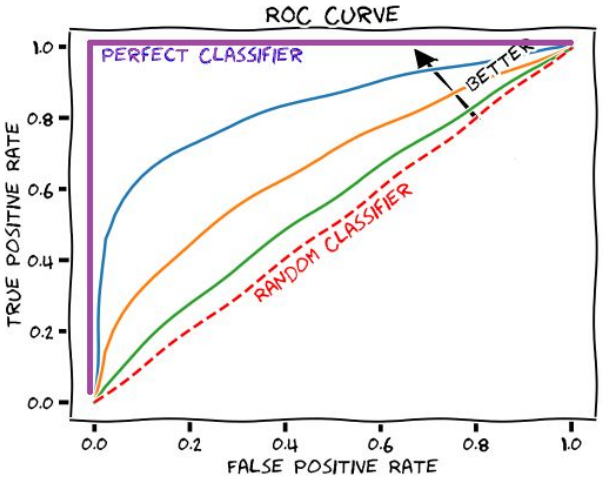
	Nodes	Edges	Node Features	Labels
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19717	44338	500	3



Structure of Cora Dataset

Experimental Evaluation

- Link prediction: Evaluation metrics
  - Area under the ROC Curve
  - Average precision
- Baseline link prediction methods:
  - Spectral clustering, DeepWalk, node2vec, GAE/VGAE



$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$$

P_n = precision at n^{th} threshold
R_n = recall at the n^{th} threshold

Experimental Evaluation

- Link prediction results
  - Graphite-VAE gives the best performance overall, and Graphite-AE also does well
  - Note: Table is empty for SC, DeepWalk, and node2vec for the datasets with features because they are not able to incorporate node features while learning embeddings

Table 3. Area Under the ROC Curve (AUC) for link prediction (* denotes dataset with features). Higher is better.

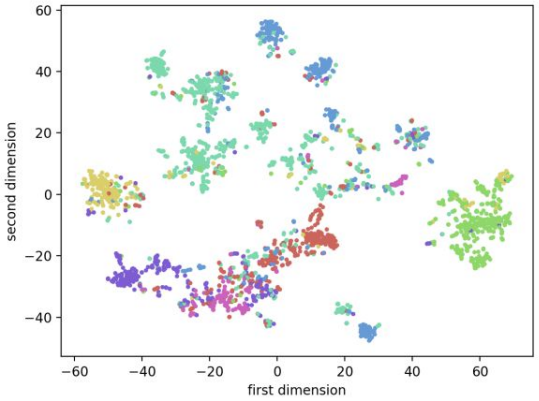
	Cora	Citeseer	Pubmed	Cora*	Citeseer*	Pubmed*
SC	89.9 ± 0.20	91.5 ± 0.17	**94.9** ± 0.04	-	-	-
DeepWalk	85.0 ± 0.17	88.6 ± 0.15	91.5 ± 0.04	-	-	-
node2vec	85.6 ± 0.15	89.4 ± 0.14	91.9 ± 0.04	-	-	-
GAE	90.2 ± 0.16	92.0 ± 0.14	92.5 ± 0.06	93.9 ± 0.11	94.9 ± 0.13	96.8 ± 0.04
VGAE	90.1 ± 0.15	92.0 ± 0.17	92.3 ± 0.06	94.1 ± 0.11	96.7 ± 0.08	95.5 ± 0.13
Graphite-AE	91.0 ± 0.15	92.6 ± 0.16	94.5 ± 0.05	94.2 ± 0.13	96.2 ± 0.10	**97.8** ± 0.03
Graphite-VAE	**91.5** ± 0.15	**93.5** ± 0.13	94.6 ± 0.04	**94.7** ± 0.11	**97.3** ± 0.06	97.4 ± 0.04

Table 4. Average Precision (AP) scores for link prediction (* denotes dataset with features). Higher is better.

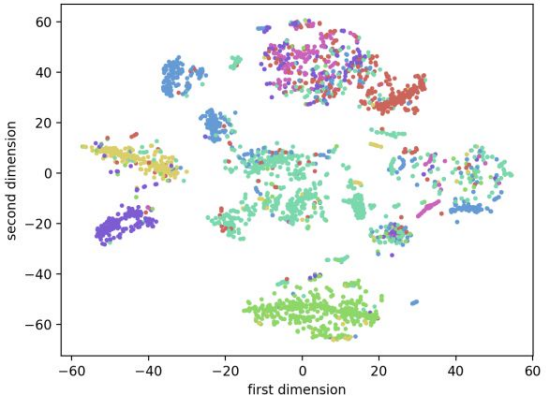
	Cora	Citeseer	Pubmed	Cora*	Citeseer*	Pubmed*
SC	92.8 ± 0.12	94.4 ± 0.11	**96.0** ± 0.03	-	-	-
DeepWalk	86.6 ± 0.17	90.3 ± 0.12	91.9 ± 0.05	-	-	-
node2vec	87.5 ± 0.14	91.3 ± 0.13	92.3 ± 0.05	-	-	-
GAE	92.4 ± 0.12	94.0 ± 0.12	94.3 ± 0.5	94.3 ± 0.12	94.8 ± 0.15	96.8 ± 0.04
VGAE	92.3 ± 0.12	94.2 ± 0.12	94.2 ± 0.04	94.6 ± 0.11	97.0 ± 0.08	95.5 ± 0.12
Graphite-AE	92.8 ± 0.13	94.1 ± 0.14	95.7 ± 0.06	94.5 ± 0.14	96.1 ± 0.12	**97.7** ± 0.03
Graphite-VAE	**93.2** ± 0.13	**95.0** ± 0.10	**96.0** ± 0.03	**94.9** ± 0.13	**97.4** ± 0.06	97.4 ± 0.04

Experimental Evaluation

- Qualitative evaluation
  - Visualizing the latent feature vectors (Z from previous sections) in 2D using 2D t-SNE projection
  - Visualization on the Cora dataset
    - The nodes (papers) are clustered by their labels (paper categories) even without access to label information in training



(a) Graphite-AE



(b) Graphite-VAE

Figure 2. t-SNE embeddings of the latent feature vectors for the Cora dataset. Colors denote labels.

Experimental Evaluation

- Semi-supervised node classification
  - *“Given labels for a subset of nodes in an underlying graph, predict the labels for the remaining nodes.”*
  - Most similar approach: GCN
  - Hybrid Approach: Augment GCN with the Graphite objective, and use a hyperparameter to control the weighting of the two terms in the combined objective

Table 5. Classification accuracies (* denotes dataset with features). Baseline numbers from Kipf & Welling (2017).

	Cora*	Citeseer*	Pubmed*
SemiEmb	59.0	59.6	71.1
DeepWalk	67.2	43.2	65.3
ICA	75.1	69.1	73.9
Planetoid	75.7	64.7	77.2
GCN	81.5	70.3	79.0
Graphite	**82.1** $\pm$ 0.06	**71.0** $\pm$ 0.07	**79.3** $\pm$ 0.03

Theoretical Analysis - Kernel Embedding

- Kernel checks similarity between two objects $K : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}$ Z is object space
- Given K, construct feature map $\psi : \mathcal{Z} \to \mathcal{H}$ where H is some feature space
- One step further, we map a distribution to H $T_\psi : \mathcal{P} \to \mathcal{H}$ P is all distributions of Z
- Formally: $T_\psi(p) := \mathbb{E}_{Z \sim p}[\psi(Z)]$
  - If this is injective (diff inputs result in diff outputs), then functions on distributions can be associated with functions on feature maps: For any $\mathcal{O} : \mathcal{P} \to \mathbb{R}^d$ we can find $\tilde{\mathcal{O}}_\psi : \mathcal{H} \to \mathbb{R}^d$ Such that $\mathcal{O}(p) = \tilde{\mathcal{O}}_\psi(T_\psi(p))$
  -
    - This will be useful later!

Theoretical Analysis - Mean field inference

- By the mean field approximation, we can assume independence of latent variable distributions when conditioned on A, X. So any conditional distribution (r) can be written as

$$r(\mathbf{Z}_1, \cdots, \mathbf{Z}_n | \mathbf{A}, \mathbf{X}) \approx \prod_{i=1}^{n} q_{\phi_i}(\mathbf{Z}_i | \mathbf{A}, \mathbf{X})$$

- And we can then solve for learnable parameters using KL divergence

$$\min_{\phi_1, \cdots, \phi_n} \mathrm{KL}\left(\prod_{i=1}^{n} q_{\phi_i}(\mathbf{Z}_i | \mathbf{A}, \mathbf{X}) \| r(\mathbf{Z}_1, \cdots, \mathbf{Z}_n | \mathbf{A}, \mathbf{X}) \right)$$

Theoretical Analysis - Mean field inference

- From standard techniques, the optimal marginal distribution has the below form, where N(i) denotes the neighbors of node i and O is some function
-
$$q_{\phi_i}(\mathbf{Z}_i|\mathbf{A},\mathbf{X}) = \mathcal{O}_{\mathcal{G}}^{MF}\left(\mathbf{Z}_i, \{q_{\phi_j}\}_{j\in\mathcal{N}(i)}\right)$$
-
- Note that this function only depends on the latent representation and the distribution of its neighbors, implying local consistency
- To solve, we can use an iterative approach via message passing

$$q_{\phi_i}^{(l)}(\mathbf{Z}_i|\mathbf{A},\mathbf{X}) = \mathcal{O}_{\mathcal{G}}^{MF}\left(\mathbf{Z}_i, \{q_{\phi_j}^{(l-1)}\}_{j\in\mathcal{N}(i)}\right)$$

Theoretical Analysis - Mean field inference

- Now assuming injectivity, we solve directly for mean

$$q_{\phi_i}^{(l)}(\mathbf{Z}_i|\mathbf{A},\mathbf{X}) = \mathcal{O}_{\mathcal{G}}^{MF}\left(\mathbf{Z}_i, \{q_{\phi_j}^{(l-1)}\}_{j\in\mathcal{N}(i)}\right)$$

$$\boldsymbol{\mu}_i^{(l)} = \tilde{O}_{\psi,\mathcal{G}}^{MF}\left(\{\boldsymbol{\mu}_j^{(l-1)}\}_{j\in\mathcal{N}(i)}\right)$$

- We can further simplify with an important thm, which states that we can instead use the following update equation* as a 1st order approximation:

*Given suitable choice of

η_l, $\mathcal{F}_l$, $\mathbf{W}_l$, *and* $\mathbf{B}_l$

Fixed — learnt

$$\mathbf{H}^{(l)} \leftarrow \eta_l\left(\mathbf{B}_l + \sum_{f\in\mathcal{F}_l} f(\mathbf{A})\mathbf{H}^{(l-1)}\mathbf{W}_l\right)$$

Therefore it makes sense to use GNN!

Theoretical Analysis - rough proof of thm

Re-write in terms of $N_i^{(l-1)}$

$$\boldsymbol{\mu}_i^{(l)} = \tilde{O}_{\psi,\mathcal{G}}^{MF}\left(\{\boldsymbol{\mu}_j^{(l-1)}\}_{j\in\mathcal{N}(i)}\right) \longrightarrow \mu_i^{(l)} = \tilde{O}_{\psi,\mathcal{G}}\left(\mathbf{N}_i^{(l-1)}\right)$$

Then by Taylor series expansion,

$$\mu_i^{(l)} \approx \tilde{O}_{\psi,\mathcal{G}}(\mathbf{0}) + \mathbf{N}_i^{(l-1)} \cdot \nabla\tilde{O}_{\psi,\mathcal{G}}(\mathbf{0})$$

Which is similar to plugging in the values
For the update equation below

$$H_i^{(l)} = \eta_l\left(B_{l,i} + \sum_{f\in\mathcal{F}_l} f(\mathbf{A}_i)\mathbf{H}^{(l-1)}W_l\right)$$

- $\eta_l \leftarrow \mathcal{I}$ (identity function)
- $B_{l,i} \leftarrow \tilde{O}_{\psi,\mathcal{G}}(\mathbf{0})$
- A family of n transformations $\mathcal{F}_l = \{f_{l,j}\}_{j=1}^n$ where $f_{l,j}(\mathbf{A}_i) = \frac{\partial\tilde{O}_{\psi,\mathcal{G}}}{\partial\mu_j^{(l-1)}}(\mathbf{0})\,A_{ij}$
- $H_i^{(l-1)} \leftarrow \mu_i^{(l-1)}$
- $W_l \leftarrow 1.$

Conclusion

- Scalable, generative model
- Encoder/decoder with GNN
- Performs well on different datasets
- Future work
  - Can use different GNN (instead of graph conv net)
  - Explore permutation invariance
  - Expand to other graph structure like time-varying
  - Use knowledge of domain to create synthesizer, parser, or other

References

ROC Image: https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/
AP Score Equation: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html#rcdf8f32d7f9d-1
GNN Image: https://www.youtube.com/watch?v=2KRAOZIULzw
WL Algorithm Images: https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

Quiz

1. (True/False) Graphite can model both graph structure and node features.
2. (True/False) Graphite differs from GVAE/GAE by having learnable parameters for decoding.
3. (Multi-Choice) Graphite outperforms most competing approaches in:

a) Density Estimation b) Link Prediction c) Semi-supervised Node Classification