A Discussion on "Graphite: Iterative Generative Modeling of Graphs"

Discussant: Roman Kouznetsov

October 19, 2020

Review of VAE

VAE was introduced as a way to turn the bottleneck of autoencoders into a generative process from a latent space.

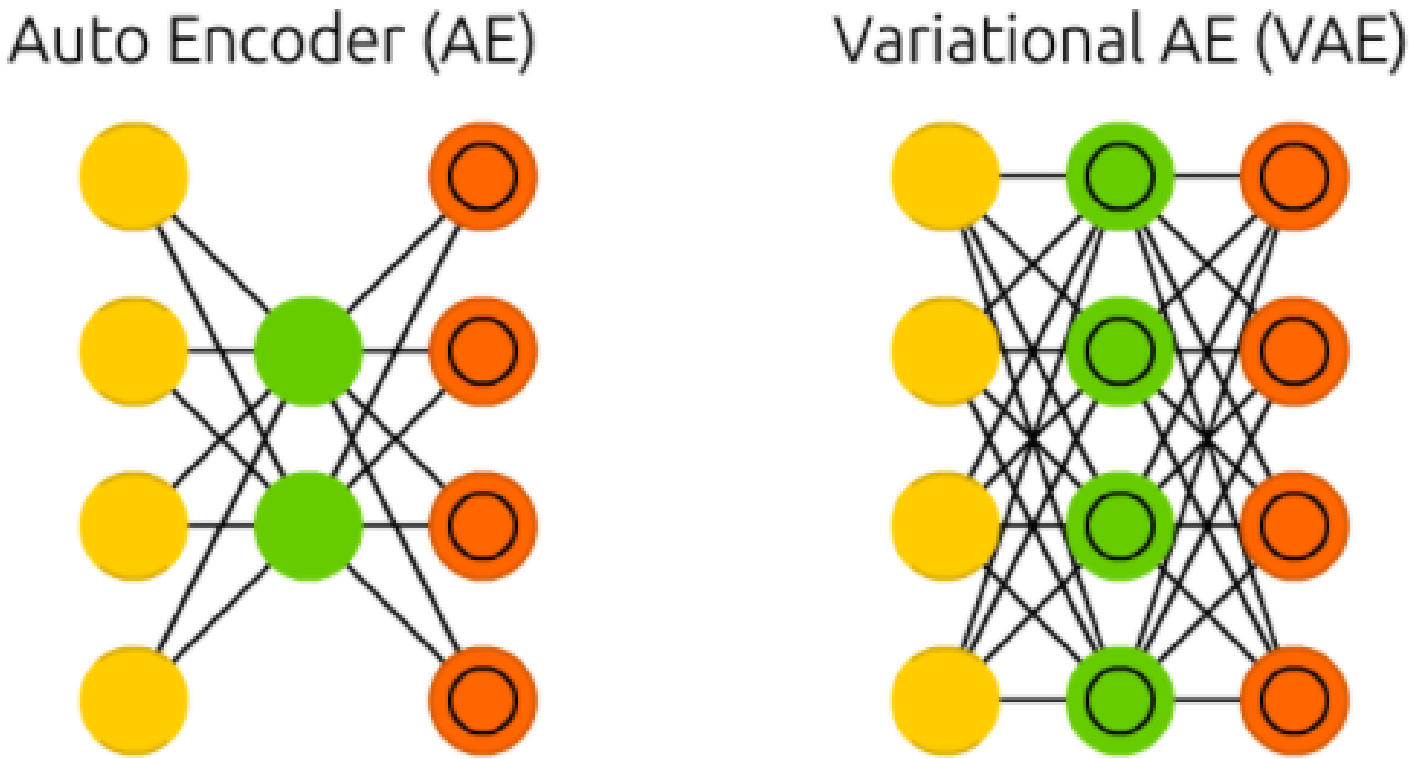


Figure: AE v. VAE

Image Credit:

Graphs

- Graphite works with undirected, unweighted graphs so that is the kind of graph I will introduce here.
- Graphs are objects that contain nodes and edges that connect them.
- Unweighted graphs means that all edges are treated identically storing the same information.
- Undirected graphs means that edges are bidirectional (i.e. messages can be passed both ways). This means that the adjacency matrix is symmetrical (more on this later).
- Graphs can have node labels, node features, and edge features that better describe the structure of a graph.

Graph Examples

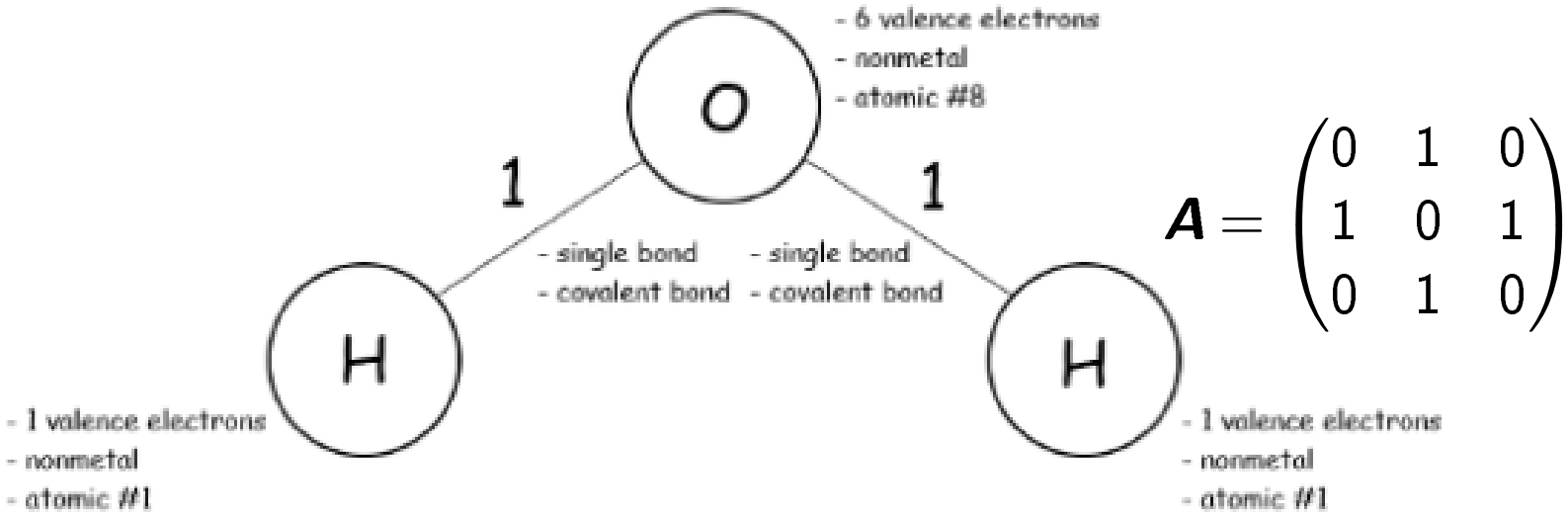


Figure: A Water Molecule Graph w/ Corresponding Adjacency Matrix

Image Credit: `https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b`

Some Graph Notation

- $G = (V, E)$: undirected graph with nodes V and edges E
- $X \in \mathbb{R}^{n \times m}$: optional feature matrix where each node has an m-dimensional attribute
- $A \in \mathbb{R}^{n \times n}$: adjacency matrix s.t. $A_{i,j}$ represents the weight of the edge between node i and j
- $H = GNN_{\langle \boldsymbol{W}, \boldsymbol{B} \rangle}(\boldsymbol{A}, \boldsymbol{X})$: graph neural network of adjacency matrix A and feature matrix X on learnable weight and bias parameters W and B (where W and B contain all of the weights and biases across layers)

Takeaway: A graph is defined by its adjacency and feature matrices. This will be an important concept later.

Introduction to Graph Neural Networks (GNNs)

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

$\boldsymbol{B}_l$: bias at current layer

$\boldsymbol{A}$: adjacency matrix

$f \in \mathscr{F}_l$: a set of linear transformations that acts on $\boldsymbol{A}$

$\boldsymbol{H}^{(l-1)}$: activations of the the previous layer

$\boldsymbol{W}_l$: weights at current layer

Introduction to Graphite

- **Research Objective:** GNNs do the simple encoding while Graphite focuses on the challenges of creating the decoder.
- **Previous Work**: Previous work in this field has been done, but none have used iterative graph building with learnable parameters.
- **Laymen's Terms**: Graphite has a superior performance because of its parameterization.

Describing the Graphite Model

Graphite introduces:

- Latent Variable Vectors: $\boldsymbol{Z_i} \in \mathbb{R}^k$
- Individual Node Fature Vectors: $\boldsymbol{X}_i \in \mathbb{R}^m$

These matrices will prove to be instrumental to how an iterative generative model of graphs.

Graphite Latent Variable Model

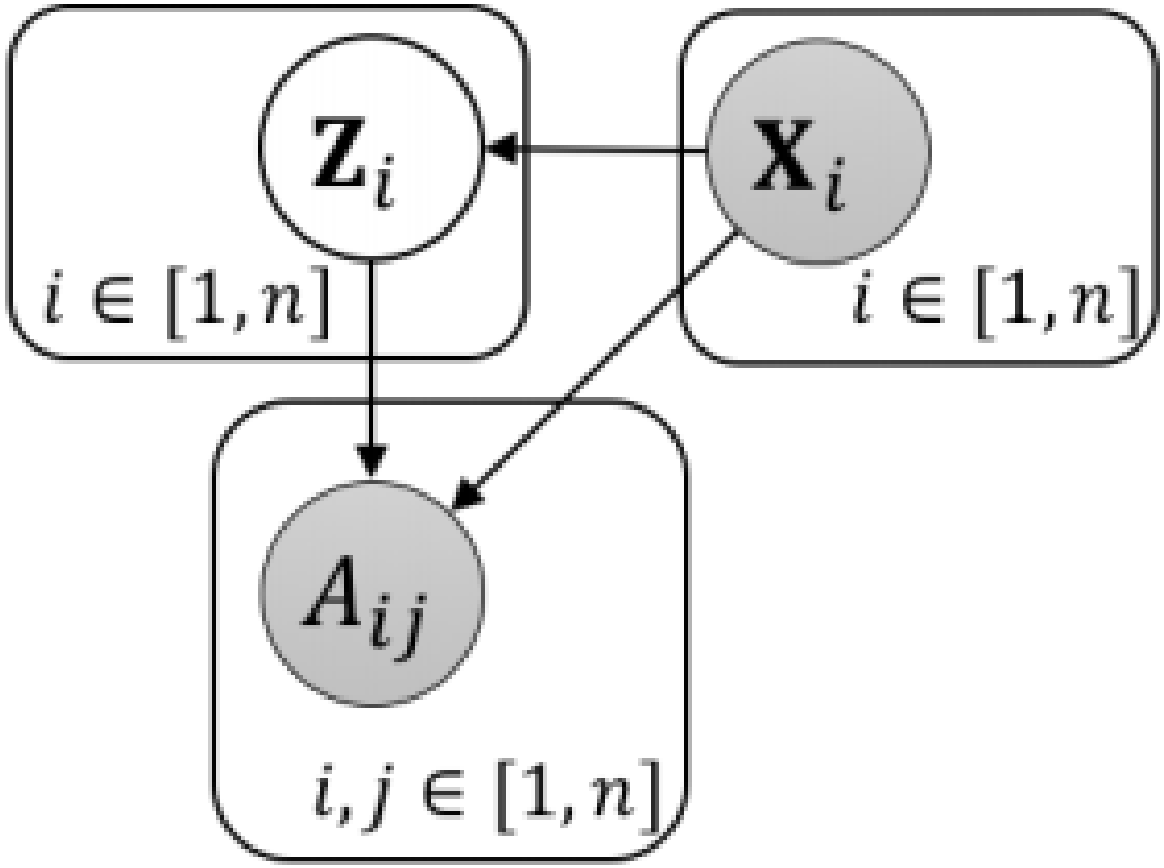


Figure 1. Latent variable model for Graphite. Observed evidence variables in gray.

Figure: Graphite Latent Variable Model

Image Credit: *Graphite: Iterative Generative Modeling of Graphs*

Graphite Achievements

Graphite has been able to achieve a superior performance to its predecessor in the following problems:

- Reconstruction & Density Estimation
- Link Prediction
- Semi-Supervised Node Classification

Encoder

Let's compare Graphite to a standard VAE.

$$(\mu, \log \sigma) = \text{EncoderNeuralNet}_\phi(x)$$
$$q_\phi(z|x) = N(z; \mu, \text{diag}\sigma)$$

$$\mu, \sigma = \text{GNN}_\phi(\mathbf{A}, \mathbf{X})$$

where μ and σ denote the vector of means and stand
deviations for the variational marginals $\{q_{\phi_i}(\mathbf{Z}_i|\mathbf{A}, \mathbf{X})\}$

Figure: Comparing Encoders

Image Credit: Ismael Mendoza (from this reading group :-)) (left) and *Graphite: Iterative Generative Modeling of Graphs* (right)

Decoder

In VAE, $\boldsymbol{x}$ and $\boldsymbol{z}$ are vectors and so message passing between them can be established with simple NNs. However, in the context of Graphite, our input for learning $p_\theta(x|z)$ is a graph but a forward pass through the encoder yields a latent matrix. How do we create a decoder that actually has learnable parameters? **This question is the motivation for Graphite.**

$$(\boldsymbol{\mu}_l, \boldsymbol{\sigma}_l) = \text{DecoderNN}_{\boldsymbol{\theta}}(\boldsymbol{z}) \qquad \widehat{\mathbf{A}} = \frac{\mathbf{Z}\mathbf{Z}^T}{\|\mathbf{Z}\|^2} + \mathbf{1}\mathbf{1}^T,$$
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_l, \text{diag}(\boldsymbol{\sigma}_l)) \qquad \mathbf{Z}^* = \text{GNN}_\theta(\widehat{\mathbf{A}}, [\mathbf{Z}|\mathbf{X}])$$

Figure: Comparing Decoders

Image Credit: Ismael Mendoza (from this reading group :-)) (left) and *Graphite: Iterative Generative Modeling of Graphs* (right)

Derivation of ELBO for GNNs

$$
\begin{aligned}
log p_\theta(\boldsymbol{A}|\boldsymbol{X}) &= E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}[log p_\theta(\boldsymbol{A}|\boldsymbol{X})] \\
&= E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{p_\theta(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] \\
&= E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X}) q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X}) p_\theta(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] \\
&= E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})} + log \frac{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}{p_\theta(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] \\
&= E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] + E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}{p_\theta(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] \\
&\geq E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log \frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right]
\end{aligned}
$$

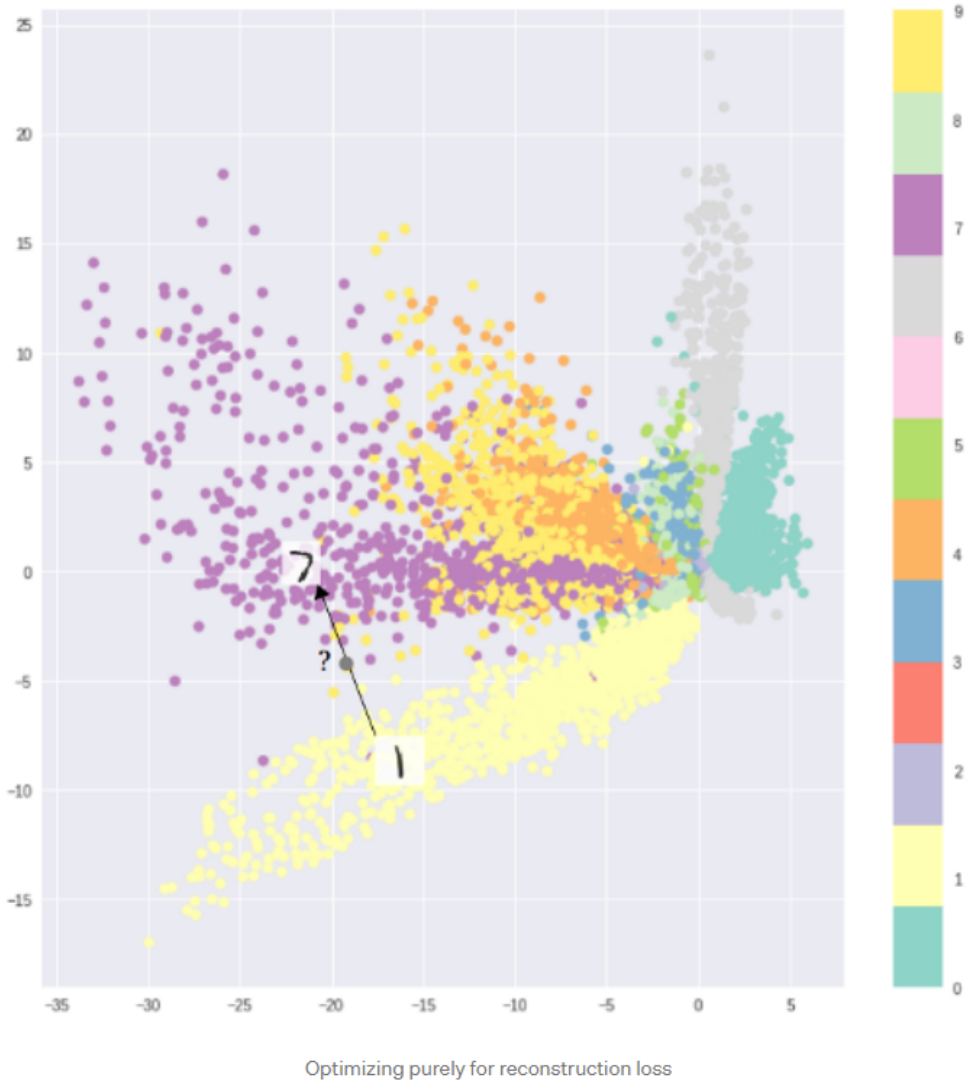
Interpretation of ELBO

$$E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log\frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] = logp_\theta(\boldsymbol{A}|\boldsymbol{X}) - E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log\frac{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}{p_\theta(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right]$$

1. Learn correct adjacency matrix.
2. Make sure the learned posterior distribution is close to the (unknown) true posterior distribution.

These 2 terms working in tandem is critical to understanding why ELBO works as such a great optimizer.

Showcase of Optimization on $E(logp_\theta(\boldsymbol{A}|\boldsymbol{Z},\boldsymbol{X}))$

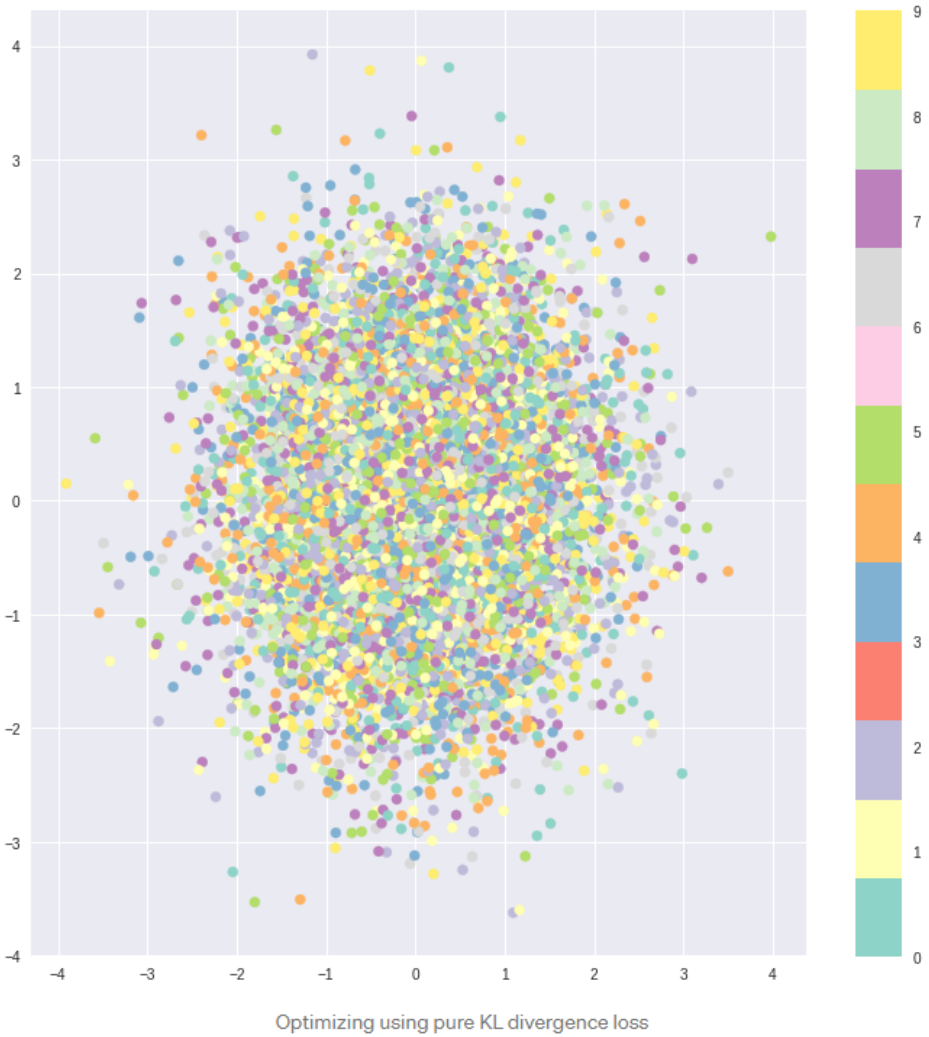


Classification ✓

Complete Support X

Image Credit: `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`

Showcase of Optimization on $\mathrm{KL}(q_\phi(z|x)||p_\theta(z|x))$

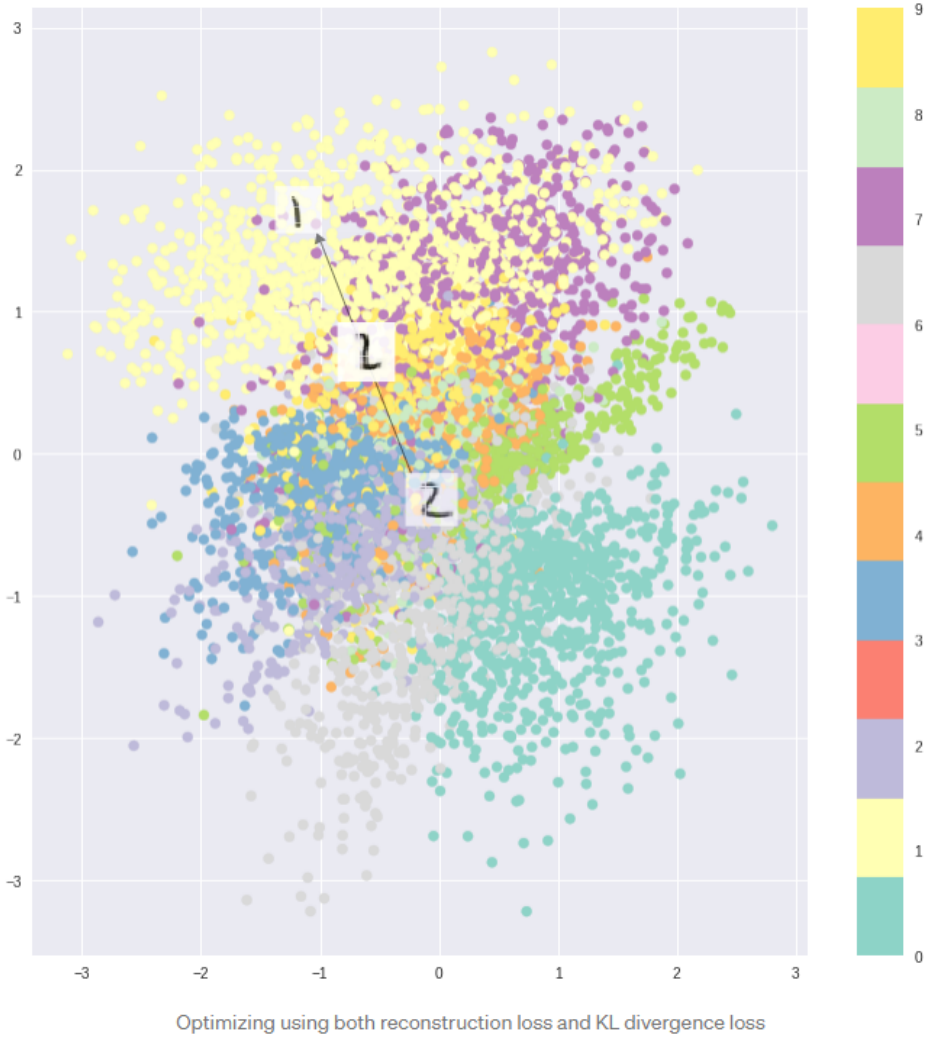


Classification X

Complete Support ✓

Image Credit: `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`

Showcase of Optimization on ELBO



Classification ✓

Complete Support ✓

Image Credit: `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`

Computation of ELBO

$$ELBO = E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log\frac{p_\theta(\boldsymbol{A},\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right] =$$

$$E_{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\left[log\frac{p_\theta(\boldsymbol{A}|\boldsymbol{Z},\boldsymbol{X})p_\theta(\boldsymbol{Z}|\boldsymbol{X})}{q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})}\right]$$

Therefore, we can simply sample from the prior $p_\theta(\boldsymbol{Z}|\boldsymbol{X})$, our variational marginal $q_\phi(\boldsymbol{Z}|\boldsymbol{A},\boldsymbol{X})$, and our reconstruction posterior $p_\theta(\boldsymbol{A}|\boldsymbol{Z},\boldsymbol{X})$ to get our objective function.

Importance of Reparameterization Trick

Goal: Take Monte Carlo integrals to calculate the gradient of our objective.

This goal might not be directly achievable because of problems such as...

- **No Analytic Solution**
- **Higher Variance**

Summary: The reparameterization trick can help in both departments.

Reparameterization Trick: Case of No Analytic Solution

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(z)}[f_\theta(z)] &= \nabla_\theta \Big[\int_z p_\theta(z) f_\theta(z) dz \Big] \\
&= \int_z \nabla_\theta \Big[p_\theta(z) f_\theta(z) \Big] dz \\
&= \int_z f_\theta(z) \nabla_\theta p_\theta(z) dz + \int_z p_\theta(z) \nabla_\theta f_\theta(z) dz \\
&= \underbrace{\int_z f_\theta(z) \nabla_\theta p_\theta(z) dz}_{\text{What about this?}} + \mathbb{E}_{p_\theta(z)} \Big[\nabla_\theta f_\theta(z) \Big]
\end{aligned}
$$

Figure: Generic Gradient Calculation of Expected Value

Image Credit: `http://gregorygundersen.com/blog/2018/04/29/reparameterization/`

Reparameterization Trick: Lowering Variance

In some cases ($p_\theta(z) \sim N(\mu, \sigma^2)$ being one of them), our gradient can have an analytic solution, meaning that we an expectation and we can estimate it using samples.

So... what's the problem?

Let's see the case I presented as an example!

$q \sim N(\theta, 1)$ Example

Let's say that we were interested in the case where $q_\theta \sim N(\theta, 1)$ and we are interested in the following optimization problem:

$$\theta = argmin_\theta E_q[x^2]$$

Obviously we know the answer is $\theta = 0$ because

$$min_\theta E_q[x^2] = min_\theta \Big((E_q[x])^2 + Var(x) \Big) = min_\theta (\theta^2 + 1) \rightarrow \theta = 0$$

$\nabla_\theta E_q[x^2]$ is as follows

$$\nabla_\theta E_q[x^2] = \nabla_\theta \int q_\theta(x) x^2 dx = \int x^2 \nabla_\theta q_\theta(x) \frac{q_\theta(x)}{q_\theta(x)} dx = \int q_\theta(x) \nabla_\theta \log q_\theta(x) x^2 dx = E_q[x^2 \nabla_\theta \log q_\theta(x)]$$

e $q_\theta(x) = N(\theta, 1)$, this method gives

$$\nabla_\theta E_q[x^2] = E_q[x^2(x - \theta)]$$

Figure: Gradient Calculation of Normal Case

Image Credit:

The Reparameterization

Motivation: The result on the previous slide shows we can sample from q; so theoretically, everything is fine. Why would the reparamerization trick even be needed here?

Let's see what happens when we take the expectation with respect to a distribution that is independent of the parameter we took the gradient with respect to.

$$x = \theta + \epsilon, \quad \epsilon \sim N(0, 1)$$

$$E_q[x^2] = E_p[(\theta + \epsilon)^2]$$

can write the derivative of $E_q[x^2]$ as follows

$$\nabla_\theta E_q[x^2] = \nabla_\theta E_p[(\theta + \epsilon)^2] = E_p[2(\theta + \epsilon)]$$

Figure: Reparameterization of Normal Case

Reparameterization Simulation

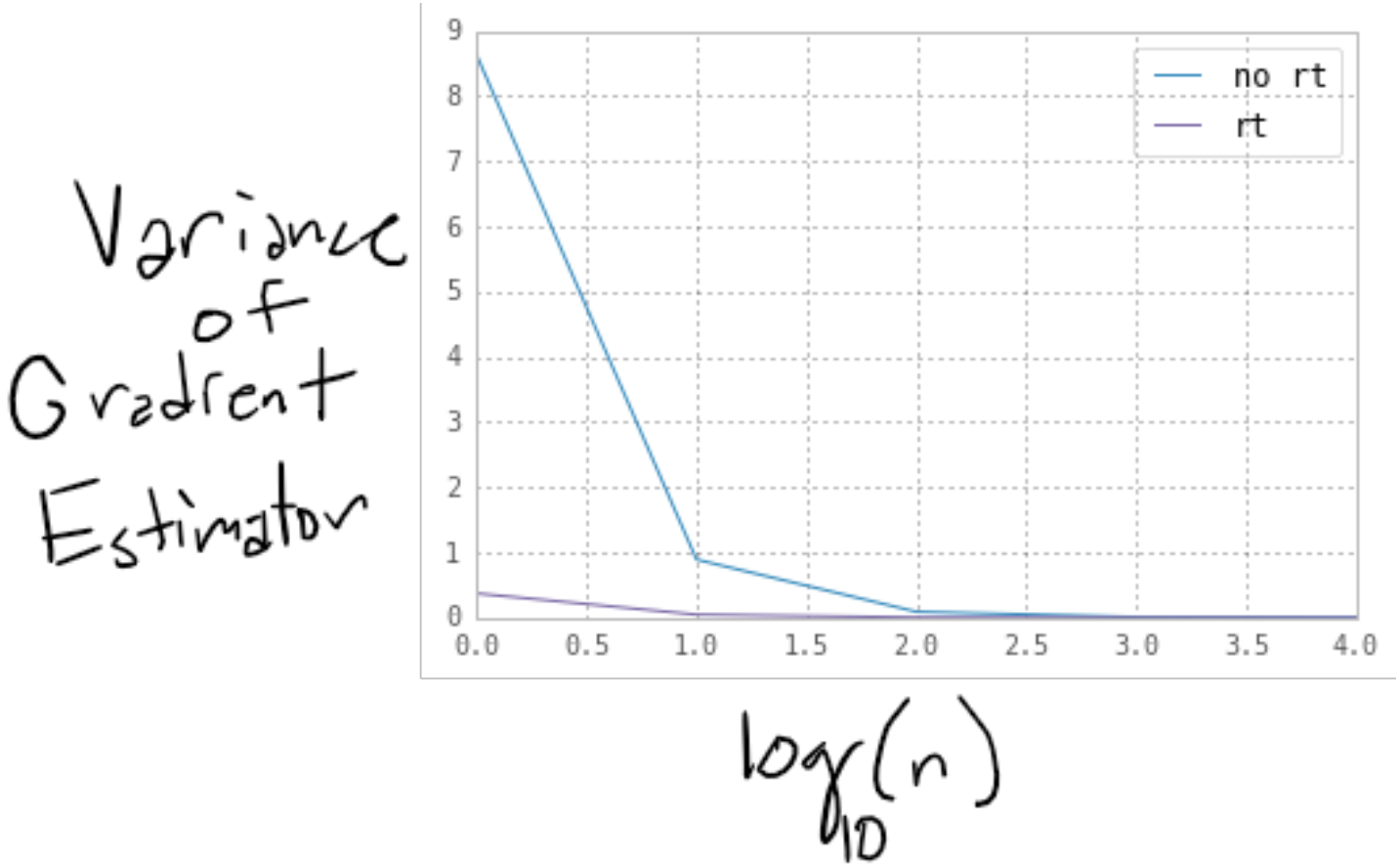


Figure: Variance of Gradient Estimator as a Function of Logarithmic Sample Size

Formal Model

$$
\begin{aligned}
p(\boldsymbol{z}) &= \mathcal{N}(0, \boldsymbol{I}) \\
(\mu, \sigma) &= \text{EncoderNN}_{\phi}(\boldsymbol{x}) \\
q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) &= \mathcal{N}(\boldsymbol{z}; \mu, \text{diag}(\sigma)) \\
(\mu_l, \sigma_l) &= \text{DecoderNN}_{\theta}(\boldsymbol{z}) \\
p_{\theta}(\boldsymbol{x}|\boldsymbol{z}) &= \mathcal{N}(\boldsymbol{x}; \mu_l, \text{diag}(\sigma_l))
\end{aligned}
$$

Image Credit: Ismael Mendoza

$$P(\boldsymbol{Z}|\boldsymbol{X}) \sim \mathcal{N}(\mu, \sigma^2 \boldsymbol{I})$$

$$\mu, \sigma = GNN_{\phi}(\boldsymbol{A}, \boldsymbol{X})$$

$$q_{\phi}(\boldsymbol{Z}|\boldsymbol{A}, \boldsymbol{X}) := \prod_{i=1}^{n} q_{\phi_i}(\boldsymbol{Z}_i|\boldsymbol{A}, \boldsymbol{X})$$

$$\hat{\boldsymbol{A}} = \frac{\boldsymbol{Z}\boldsymbol{Z}^{\boldsymbol{T}}}{||\boldsymbol{Z}||^2} + \boldsymbol{1}\boldsymbol{1}^{\boldsymbol{T}}$$

$$\boldsymbol{Z}^* = GNN_{\theta}(\hat{\boldsymbol{A}}, [\boldsymbol{Z}|\boldsymbol{X}])$$

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

Personal Tangent

Jeff and I are actually attempting to do something similar, but would use something like Graphite to learn a latent variable to generate $\boldsymbol{X}$, the feature matrix, instead of $\boldsymbol{A}$, the adjacency matrix. This tells us more about the nodal attributes of the graph while preserving graph relationships rather than trying to recreate the graph itself.

Building Up Model Assumptions

Property: Under $r(\boldsymbol{Z_1}, \boldsymbol{Z_2}, ..., \boldsymbol{Z_n} | \boldsymbol{A}, \boldsymbol{X})$, any $\boldsymbol{Z}_i$ is independent of all other $\boldsymbol{Z}_i$ when conditioned on $\boldsymbol{A}, \boldsymbol{X}$ and all neighboring latent variables of node i.

Takeaway: Graphite usually expects latent distributions that follows this property so that the following optimization problem is solved and computationally inexpensive.

Mean Field Approximation

A mean field approximation is an approximation that is used as follows:

$$r(\boldsymbol{Z_1}, \boldsymbol{Z_2}, ..., \boldsymbol{Z_n}|\boldsymbol{A}, \boldsymbol{X}) \approx \prod_{i=1}^{n} q_{\phi i}(\boldsymbol{Z}_i|\boldsymbol{A}, \boldsymbol{X})$$

Variational Marginal Solution

Objective:

$$min_{\phi_1,\ldots,\phi_n} KL\left[r(\boldsymbol{Z_1}, \boldsymbol{Z_2}, \ldots, \boldsymbol{Z_n}|\boldsymbol{A}, \boldsymbol{X})||\prod_{i=1}^{n} q_{\phi i}(\boldsymbol{Z}_i|\boldsymbol{A}, \boldsymbol{X})\right]$$

Solution: $q_{\phi i}(\boldsymbol{Z}_i|\boldsymbol{A}, \boldsymbol{X}) = \mathscr{O}_{\mathscr{G}}^{MF}(\boldsymbol{Z}_i, \{q_{\phi_j}\}_{j\in\mathscr{N}_{(i)}})$ where $\mathscr{N}_{(i)}$is the neighbors of node i

This solution is provided by Wainwright et al. 2008 and is given without proof.

Connection Between WL and Kernel Embeddings with GNN (Eq 17 & Theorem 2)

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

$$\mu_i^{(l)} = \mathscr{O}_{\mathscr{G}}^{MF}(\{\mu_i^{(l)}\}_{j\in\mathscr{N}_{(i)}})$$

Theorem 2. *Let $\mathcal{G}$ be any undirected latent variable model such that the conditional distribution $r(\mathbf{Z}_1, \cdots, \mathbf{Z}_n|\mathbf{A}, \mathbf{X})$ expressed by the model satisfies Property 1.*

Then there exists a choice of η_l, $\mathcal{F}_l$, $\mathbf{W}_l$, and $\mathbf{B}_l$ such that for all $\{\boldsymbol{\mu}_i^{(l-1)}\}_{i=1}^n$, the GNN propagation rule in Eq. (2) is computationally equivalent to updating $\{\boldsymbol{\mu}_i^{(l-1)}\}_{i=1}^n$ via a first order approximation of Eq. (17).

Proof. See Appendix A. □

Figure

Image Credit: *Graphite: Iterative Generative Modeling of Graphs*

How GNNs Accomplish This (Last Sentence of Sec. 5)

This is why GNNs in a VAE setting are so powerful. They can learn the parameters needed to make the computation incredibly simple while still doing the impressive learning that a VAE does.

Note: I am just relaying the result Theorem 2 back to you. I'm not entirely sure how this result is practical in the paper and have left it as a discussion question.

Experimental Results: Reconstruction

Graphite competes against (Variational) Graph Autoecnoder. The task is to reconstruct the original graph (autoencoder case) or create a good latent variable model (variational autoencoder case).

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

Figure: Results of (V)GAE v. Graphite in Reconstruction and Density Estimation

Image Credit: *Graphite: Iterative Generative Modeling of Graphs*

Why the overwhelmingly better performance? GAE/VGAE doesn't use learnable parameters for its graph reconstruction where Graphite does. Graphite learns θ for $\boldsymbol{Z}^*$iteratively until the final graph reconstruction.

Experimental Results: Link Prediction

Goal: Predict whether two nodes share an edge.

Setup:

1. Given a complete graph, take a connected subgraph that has 85% of the edges of the original graph. This removes some true edges from the data.
2. Take the 85% subgraph for training, 5% of the removed edges for validation, and 10% of the removed edges for testing.
3. Balance the validation and testing sets by injecting some fake edges. This allows for our model to have the option to make some false positive classifications.
4. Run Graphite!

Experimental Results: Link Prediction (cont.)

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

Figure: Results of Link Prediction

Image Credit: *Graphite: Iterative Generative Modeling of Graphs*

Experiment Results: Semi-Supervised Node Learning

Goal: Given labels of some nodes in a graph, try to classify some of the unlabeled nodes.

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

Figure: Results of Node Classification

Image Credit: *Graphite: Iterative Generative Modeling of Graphs*

Important Takeaway: Graphite is incredibly strong for this task because it has a default behavior that is **transductive**. This means that Graphite can use testing nodes without their labels. This is because even if Graphite doesn't have access to a node label, it has access to its locality (node neighbors). Therefore, $f(\boldsymbol{A})$ still contains a lot of information for unlabeled nodes.