Learning Intersecting Representations of Short Random Walks on Graphs

Nikolay Malkin

Nebojsa Jojic

Abstract

We build simple generative models of neighborhood features in graphs and apply them to the problems of node classification and link prediction. Motivated by the observation that nearest neighbor models, applied to a representation constructed from neighborhood-aggregated features, work surprisingly well on several such tasks, we develop latent variable models that reflect the overlapping structure seen in features aggregated in this way (Figure 1). Our models are trained in an unsupervised manner, rely on straightforward statistical inference procedures, and produce latent variable representations of data that are useful in multiple tasks. The generative model defines a joint distribution over features and labels, thus allowing us to use the posterior distribution in classification. The posterior can also be thought of as the output of a simple neural network of a particular architecture and optionally finetuned in a supervised way. We illustrate our method on well-studied benchmarks for node classification (in various label regimes) and link prediction (in both attributed and featureless graphs), as well as on a semi-supervised image classification problem. While simple and interpretable, our methods perform near or above the state of the art.

1. Introduction

In this paper we study several problems of learning from graph-structured data. The main object in these problems is an undirected graph in which every node is decorated by a vector of features. Graph-structured data arises in such diverse settings as the study of social networks Tang and Liu (2011), in computational biology Agrawal et al. (2018), and in natural language processing, where networks of scientific publications with bag-of-words features linked by the relation of citation are considered Yang et al. (2016). Two important machine learning tasks using such data are node classification (given class labels for some of the nodes, to predict the labels of the other nodes, while all features and edges are visible) and link prediction (given a graph with some of the edges deleted, to predict which edges were deleted).

Recent approaches to these problems (Kipf and Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018; Chen et al., 2018; Huang et al., 2018; Gao and Ji, 2019; Franceschi et al., 2019; Hu et al., 2019; Ma et al., 2019; Qu et al., 2019; Tian et al., 2019; Verma et al., 2019; Rong et al., 2020) have typically used graph neural networks (GNNs). Some of these methods are inspired by models used in computer vision: common layers are neighborhood feature aggregation (pooling) and nonlinearities applied to the features at each node. Certain models, especially for link prediction (Kipf and Welling, 2016; Tran, 2018a,b;

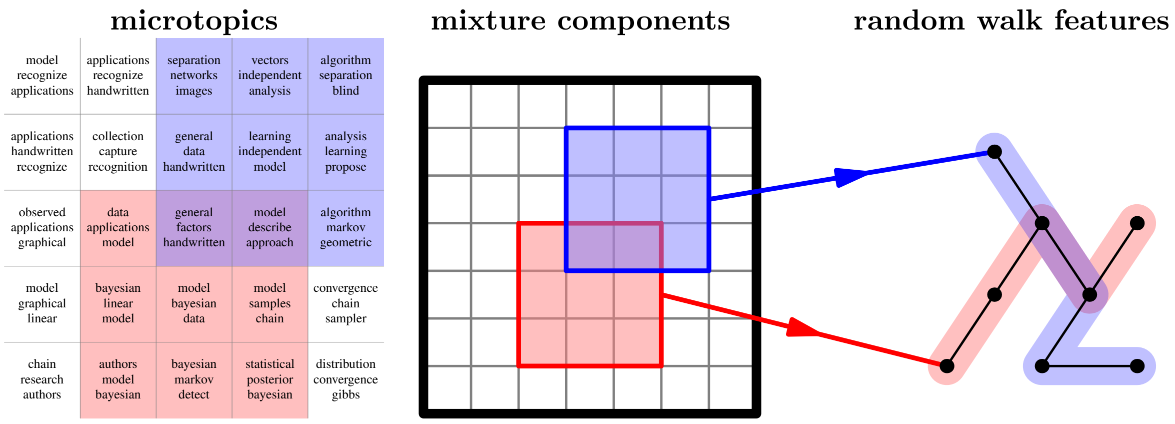

microtopics
mixture components
random walk features
model recognize applications
applications recognize handwritten
separation networks images
vectors independent analysis
algorithm separation blind
applications handwritten recognize
collection capture recognition
general data handwritten
learning independent model
analysis learning propose
observed applications graphical
data applications model
general factors handwritten
model describe approach
algorithm markov geometric
model graphical linear
bayesian linear model
model bayesian data
model samples chain
convergence chain sampler
chain research authors
authors model bayesian
bayesian markov detect
statistical posterior bayesian
distribution convergence gibbs

Figure 1: The 3×3 windows are mixture components in the counting grid (CG) model, which generates features of nodes aggregated along short random walks. Each window generates features from categorical distributions (microtopics) averaged over the grid cells. *Left:* Likely features (vocabulary words) generated from microtopics $p_{(x,y)}(f)$ of a CG trained on a network of machine learning papers. *Right:* 4-hop random walks from these two nodes have similar features, so the nodes are likely to be generated from overlapping windows. (Best viewed in color.)

Davidson et al., 2018; Pan et al., 2018; Zou and Lerman, 2019; Grover et al., 2019; Di et al., 2020), have followed the encoder-decoder paradigm, in which nodes are embedded in a high-dimensional space in such a way that the graph structure is reflected in the space's geometry.

In this work, we take a different approach: we construct node feature representations using a fixed aggregation rule and study simple unsupervised models of these features, namely, (parametrizations of) categorical mixture models. These models are primarily not trained end-to-end in a differentiable way, but rather make predictions using an elementary Bayesian inference over discrete latent variables.

Precisely, in a graph with N nodes and feature dimension F, let $\mathbf{X} \in \mathbb{R}^{N\times F}$ be the feature matrix normalized by rows to sum to a constant k, $\mathbf{A} \in \mathbb{R}^{N\times N}$ the adjacency matrix, and $\mathbf{D}$ the diagonal degree matrix. The expected features seen at the nodes along an n-hop random walk on the graph would be

$$\mathbf{X}^{(n)} := \mathbf{X} + (\mathbf{D}^{-1}\mathbf{A})\mathbf{X} + (\mathbf{D}^{-1}\mathbf{A})^2\mathbf{X} + \cdots + (\mathbf{D}^{-1}\mathbf{A})^n\mathbf{X} \in \mathbb{R}^{N\times F}. \tag{1}$$

The transformed feature matrix $\mathbf{X}^{(n)}$ is the output of a *fixed* aggregation rule.[1] Its i-th row, denoted $\mathbf{X}^{(n)}_i$, sums to $(n+1)k$ and, in the citation network setting, represents the features obtained by sampling k words from each document along a random walk starting at node i. As we show, on some node classification benchmarks, even the simplest models

1. The operator $\mathbf{D}^{-1}\mathbf{A}$ is common in graph learning and is sometimes used as the pooling operation in GNNs. Recently, Wu et al. (2019) found that simple feedforward models applied to features aggregated by a similar transform also perform surprisingly well.

— uniform mixtures and soft nearest neighbor classifiers — applied to $\mathbf{X}^{(2)}$ perform as well as some GNNs.

The short random walks starting at nearby nodes in a graph may intersect, so the nodes will share certain terms that enter into the expansion of (1). This motivates us to consider parametrizations of mixture models of the features $\mathbf{X}^{(n)}$ in which *the latent space also has an overlapping structure.* We adopt the counting grid (CG) model (Perina and Jojic, 2011). In CGs, the latent space is a discrete two-dimensional grid of cells. Each cell has an associated *microtopic*, a categorical distribution over features (e.g., words). The mixture *components* correspond to small squares (windows) in this grid and generate features from the microtopics averaged over their cells. CGs have never been used for graph learning, but fit the problem well (see Figure 1).

These generative models are trained in an unsupervised manner, but can then be used in semi-supervised node classification tasks. When labels for some nodes are given, we can use those nodes' posteriors over latent variables to infer a mapping from the latent space to the classes and estimate class probabilities for the unlabeled nodes. Similarly, for link prediction, we can train a CG and infer the probabilities of links between grid cells in the latent space. The computation of a node's posterior in the generative model can also be viewed as a neural attention layer with self-supervised initialization and (optionally) finetuned on a supervised task.

In this paper, we make the following contributions:

(1) We show the surprising performance of generative models of features aggregated from short random walks on three standard graph benchmarks, for both node classification and link prediction. The CG models surpass the (GNN) state of the art in many of these benchmarks, sometimes by a wide margin, with minimal hyperparameter tuning. Furthermore, our models have superior performance in very low label regimes without requiring additional regularization procedures. In addition, they are applicable to other domains, such as computer vision: when applied to the outputs of a deep neural net, they vastly outperform the state of the art in a scene classification problem.

(2) We explore the benefits of the generative paradigm: our models rely on elementary statistical inference and produce interpretable latent variable representations (Figure 2). They are aware of their uncertainty, and thus amenable to ensembling and active learning. The CG latent representation is robust enough that, unlike many recent GNN models, it works well simultaneously in different tasks and label regimes with few modifications.

(3) We demonstrate how our models can be optionally finetuned in a supervised way. These tuned models have high performance on node classification tasks with a large number of labels provided in training.

2. Latent variable models of short random walks

The models we will build are parametrizations of mixture models of the features $\mathbf{X}^{(n)}$. We motivate such models from both the generative model and neural network perspectives.

Consider a node classification problem, in which every node is to be assigned a class $c \in \{1, \ldots, C\}$. Suppose that labels are provided for a set of nodes T, where $T \subset \{1, \ldots, N\}$ is a training set. For $i \in T$, let c_i be the label and $y_i \in \mathbb{R}^C$ the one-hot label vector.

A **soft nearest neighbor** classifier would assign to an unlabeled node i a combination of the labels y_j for the nodes $j \in T$, weighted by the similarity of the features at i to the features at j. For features representing categorical observations, a typical choice for the similarity metric is the probability of observing the features of i under the distribution proportional to the features of j (i.e., exponent of the cross-entropy). We obtain the prediction vector for a node i:

$$\hat{y}_i \propto \sum_{j \in T} \exp\left(h \cdot \mathbf{X}_i^{(n)} (\log(\mathbf{X}_j^{(n)} + \varepsilon))^\top \right) y_j. \tag{2}$$

Here ε is a small constant added to the training set features to ensure they are positive and h is a parameter controlling the softness. In the $h \to \infty$ limit, this becomes a 1-NN classifier, which assigns to i the label of the most similar node in the training set T. On the CiteSeer benchmark (Giles et al., 1998) with features normalized to sum to $k = 3$, such a classifier applied to $\mathbf{X}^{(2)}$ with $h = \frac{1}{2}$, $\varepsilon = 0.1$ has a test accuracy of 72.5%. Thus a method requiring no training or features beyond a 2-hop neighborhood is competitive with early GNN models (Kipf and Welling, 2017; Veličković et al., 2018) and would be state of the art in 2018 (cf. Table 2).

The form of equation (2) is ubiquitous in machine learning. It is also a **neural network**

$$\hat{Y} = \operatorname{softmax}\left(\mathbf{X}^{(n)} \mathbf{W}_1^\top \right) \mathbf{W}_2, \tag{3}$$

with a softmax layer with weights $\mathbf{W}_1 = h \log(\mathbf{X}_T^{(n)} + \varepsilon)$ and a linear layer with weights $\mathbf{W}_2 = Y_T$ (the matrix of one-hot labels). Equation (2) is also equivalent to an attention layer (Bahdanau et al., 2015; Vaswani et al., 2017): $\hat{Y} = \operatorname{softmax}(QK^\top)V$, which directly uses data as a query matrix $Q = \mathbf{X}^{(n)}$, the log of training data as a key matrix $K = h \cdot \log(\mathbf{X}_T^{(n)} + \varepsilon)$, and one-hot label embedding as a value matrix $V = Y_T$. As is evident from (2), inference for all nodes simultaneously can be expressed by two matrix multiplications and a softmax operation.

These equations also have the same form as the inference equation in a **uniform categorical mixture model**. Viewing the entry $\mathbf{X}_{if}^{(n)}$ as a count of observations of feature f at node i, we could find a set of categorical distributions over the features, $p(f|s)$, where $f \in \{1, \dots, F\}$ and $s \in \{1, \dots, S\}$ is a latent variable (the index of the mixture component), so as to maximize the likelihood

$$\prod_{i=1}^{N} \frac{1}{S} \sum_{s=1}^{S} p(\mathbf{X}_i^{(n)}|s), \quad p(\mathbf{X}_i^{(n)}|s) = \prod_{f=1}^{F} p(f|s)^{\mathbf{X}_{if}^{(n)}}. \tag{4}$$

This yields a posterior over the mixture indices (clusters) for every node i, $p(s|\mathbf{X}_i^{(n)}) \propto p(\mathbf{X}_i^{(n)}|s)$. The labels of the training nodes can now be propagated to the clusters (by estimating the distribution over classes in each cluster, $p(c|s)$), and then to the unlabeled data, by computing

$$p(c|s) \propto \sum_{j \in T : c_j = c} p(s|\mathbf{X}_j^{(n)}), \quad \hat{y}_i(c) \propto \sum_s p(c|s) p(s|\mathbf{X}_i^{(n)}). \tag{5}$$

In practice, if the number of labeled nodes is small (perhaps much smaller than the number of mixture components), some mixture indices will have very low posteriors for all nodes in

T, and the predictions will be poor for evaluation nodes whose posteriors are concentrated at these unused indices. Therefore, in a low-label setting, we may choose to soften the posteriors in (5), either through the use of appropriate hyper-priors, or as we have done in our experiments, simply by introducing hyperparameters $\alpha, \beta \geq 1$:

$$p(c|s) \propto \sum_{j \in T : c_j = c} p(s|\mathbf{X}_j^{(n)}/\alpha), \quad \hat{y}_i(c) \propto \sum_s p(c|s) p(s|\mathbf{X}_i^{(n)}/\beta). \tag{6}$$

When we apply this inference procedure to a 121-component uniform mixture model of the features $\mathbf{X}^{(2)}$ in the Cora node classification benchmark (McCallum et al., 2000), we reach an accuracy of 82.3% (with optimal α, β chosen based on a validation set), again ranking among the early GNNs (cf. Table 2). (A product ensemble of 32 such models reaches 83.5%.)

This way of predicting labels $\hat{y}$ is again equivalent to the form in (2) or (3), by setting

$$\mathbf{W}_1^\top = [\log p(f|s)/\beta]_{f \in [1..F], s \in [1..S]}, \quad \mathbf{W}_2^\top = [p(c|s)]_{c \in [1..C], s \in [1..S]} \tag{7}$$

Thus, the soft nearest neighbor model is a special case of a mixture model: the mixture components are simply the training nodes with their feature distributions. However, as in the earliest semi-supervised learning approaches (McLachlan, 1992), training a mixture model on all features allows discovering the natural clusters in the data, which are then used to transfer the sparse labeling to the rest of the data, as long as the labeled nodes cover most of the natural clusters.

As we show in the experiments, these different views of the same model (3) have practical uses. First, we can choose how we wish to set model parameters depending on the size of the labeled subset of the nodes. When the labeled set is large, then we can directly optimize the parameters $\mathbf{W}_1, \mathbf{W}_2$ in a supervised manner. But for data with few labeled examples, the first layer parameters $\mathbf{W}_1$ can be constructed from a mixture model trained only on input features, which we can think of as unsupervised pre-training, and only the second layer is set based on the small number of labeled examples. A hybrid approach would over-train the first layer parameters, too, for a few iterations.

Additionally, a latent variable model (4) can be used in a link prediction task. If the matrix $\mathbf{A}$ has been damaged, with some edges deleted, the features $\mathbf{X}^{(n)}$ can still be computed, and the model trained on these features. We can then infer the likelihood of a link on the level of the latent variables:

$$p(s_1 \sim s_2) = \frac{\sum_{i \sim j} p(s_1|\mathbf{X}_i^{(n)}) p(s_2|X_j^{(n)})}{\sum_i p(s_1|\mathbf{X}_i^{(n)}) \sum_j p(s_2|\mathbf{X}_j^{(n)})}, \tag{8}$$

where the sum in the numerator is taken over the visible edges $i \sim j$. We then reconstruct the probability of a link between nodes i and j as

$$p(i \sim j) = \sum_{s_1} \sum_{s_2} p(s_1 \sim s_2) p(s_1|\mathbf{X}_i^{(n)}) p(s_2|\mathbf{X}_j^{(n)}). \tag{9}$$

Pairs of nodes that were not linked in the damaged graph may have high likelihood of a link under this model, because they are similar (in terms of their posteriors over the latents) to pairs of nodes that *were* linked. (In this high-label setting, it is sometimes beneficial to *harden* (raise to a power) the posteriors, always using $p(s|c\mathbf{X}_i^{(n)})$ in place of $p(s|\mathbf{X}_i^{(n)})$ in (8) and (9), where $c \geq 1$ is a fixed parameter.)

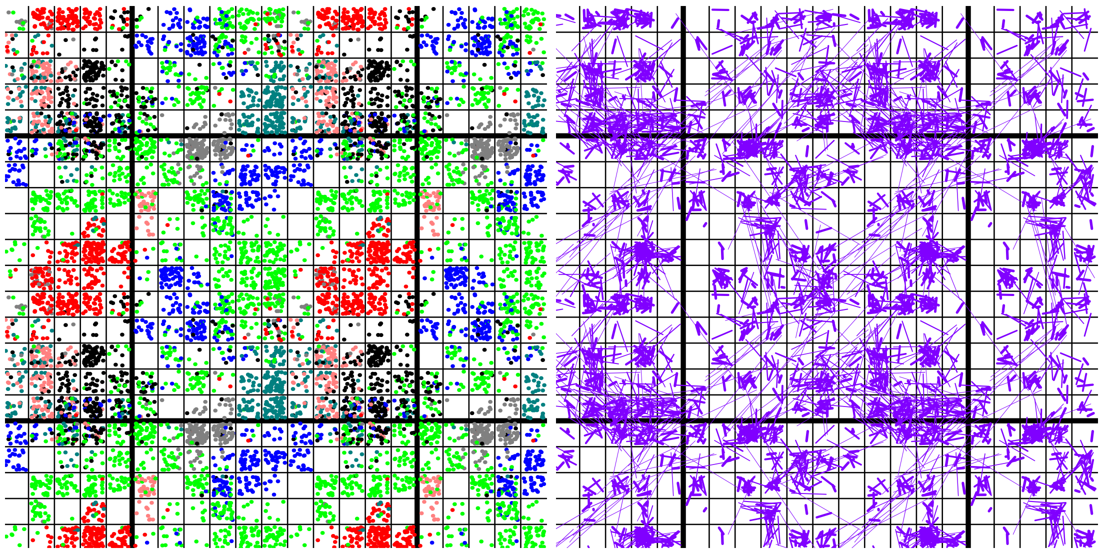


Figure 2: *Left:* A 11 / 3 CG is trained on the Cora data set McCallum et al. (2000). Each of the 2708 nodes, colored by category, is drawn in the grid cell where its posterior is highest. The 11 × 11 fundamental region is repeated on all sides to illustrate the toroidal wrap-around. *Right:* A random sample of the links in the graph are drawn as segments between their two nodes' most likely cells in the latent space. In the code repository (Appendix A), we show a video of posteriors over the grid for nodes in a random walk. (Best viewed in color.)

2.1 Counting grid parametrization of mixture models

Simple categorical mixture models — fixed (i.e., soft nearest neighbors) and learned — can perform as well as the early GNNs on some node classification tasks. Hence we ask: can an appropriately parametrized model, whose latent variable structure reflects the overlapping structure of short random walks, do even better? Intersecting neighborhoods should be modeled with intersecting representations, of which there are surprisingly few Perina and Jojic (2011); Hyvärinen et al. (2001); Iwata et al. (2008). The counting grid representation (Perina and Jojic, 2011; Jojic and Perina, 2011) is a natural fit, as it models bags of discrete features and can efficiently use cumulative sums or average pooling operations on modern hardware for training and inference. We review it here.

Recall that we are building a mixture model of features $p(f|s)$. A CG depends on two parameters: the grid size K and the window size W — we call this model a "K / W CG". It is parametrized as a family of K^2 categorical distributions $p_{(x,y)}(f)$, one for each integer coordinate pair (x, y) on a discrete $K \times K$ grid, which are called *microtopics*. The mixture components s also correspond to the positions on the grid. The distribution $p(f|s = (X, Y))$ is the mixture (average) of the microtopics $p_{(x,y)}(f)$ in a $W \times W$ square centered at (X, Y).

For example, if $W = 5$, we have

$$p(f|s = (X,Y)) = \frac{1}{W^2} \sum_{\substack{X-2\leq x\leq X+2 \\ Y-2\leq y\leq Y+2}} p_{(x,y)}(f). \tag{10}$$

All indices are to be interpreted modulo K, that is, the grid has a toroidal wrap-around.

If the mixture components $\log p(f|s)$ are encoded in a matrix $\mathbf{W}_1$, as in (7), we can express (10) through standard neural net operators: if the microtopics are encoded in a $F \times (K \times K)$ matrix of parameters $\mathbf{W}_{\text{CG}} = [\log p_{(x,y)}(f)]$, then we have

$$\mathbf{W}_1^\top = \log\left(\text{AvgPooling}_{W\times W}(\text{softmax}(\mathbf{W}_{\text{CG}}))\right), \tag{11}$$

where the average pooling uses circular padding. This form can be directly used for prediction by (3).[2]

The windows centered at nearby positions will overlap, and the microtopics on their intersection will enter into the averages defining both mixture components (see Figure 1). This essential property regularizes the model, forcing the mixture components to vary smoothly over the grid.

Figure 2 shows a CG trained on the features $\mathbf{X}^{(2)}$ of a citation network. Most links in the latent space are short: nodes that are linked in the graph tend to have high posterior in nearby grid positions. The windows around nearby cells overlap, just as the aggregated features for linked nodes share terms. The CG has "discovered" a natural clustering of the data. It is now possible to infer the likely node categories (colors) for each grid position using few labeled examples, or to infer which links between positions are likely.

2.2 Training

We explain our two training modes: unsupervised (fitting the model to maximize the data likelihood) and supervised (viewing the entire model as a simple neural network).

Unsupervised: In past work, CGs were trained by an EM algorithm Perina and Jojic (2011). However, they can also be trained by writing down the data log likelihood, by substituting the expression (10) for $p(f|s)$ into (4), and maximizing it by gradient descent on the parameters $\mathbf{W}_{\text{CG}}$. For the results below we take the latter approach, although we experimented with both methods. (We remind the reader that the precise optimization loss is the negative logarithm of (4):

$$\mathcal{L} = -\log \sum_{i=1}^{N} \frac{1}{S} \sum_{s=1}^{S} \prod_{f=1}^{F} p(f|s)^{\mathbf{X}_{if}^{(n)}}, \quad p(f|s) = [\mathbf{W}_1^\top]_{f,s},$$

with $\mathbf{W}_1^\top$ computed by (11). Thus $\mathcal{L}$ is easily expressed in neural net libraries.) Prediction is then done by computing $\mathbf{W}_2$ by (7) and applying the inference equation (3).

We emphasize that the tuning of hyperparameters (α, β) based on a validation set has very low computational cost and requires no retraining of the model. Indeed, once a CG

2. Unlike what is common in convolutional networks, softmax and AvgPooling here process the model *parameters*, not the data.

Graph	Nodes	Edges	Feature dim	Classes	Training nodes	N_1	N_2	N_3
Cora	2708	5278	1433	7	140 (5.2%)	4.9	36.8	128.1
CiteSeer	3327	4614	3703	6	120 (3.6%)	3.7	15.1	43.5
PubMed	19717	44324	500	3	60 (0.3%)	5.5	60.1	394.6

Table 1: Data set statistics. Training node counts are for the standard splits. N_i is the average size of a i-hop neighborhood.

	Cora	**CiteSeer**	**PubMed**
Method	acc%	acc	acc
DeepWalk[1]	67.2	43.2	65.3
Planetoid[2]	75.7	64.7	77.2
GCN[3]	81.5	70.3	79.0
GAT[4]	83.0	72.5	79.0
GMNN[5]	83.7	73.1	**81.8**
LDS-GNN[6]	84.1	**75.0**	–
H-GCN[7]	84.5	72.8	79.8
CG	82.24±1.05	72.49±0.59	79.46±0.71
CG ensemble	**84.68±0.45**	72.68±0.36	80.18±0.26

[1]Perozzi et al. (2014) [2]Yang et al. (2016) [3]Kipf and Welling (2017) [4]Veličković et al. (2018) [5]Qu et al. (2019) [6]Franceschi et al. (2019) [7]Hu et al. (2019)

Table 2: Node classification accuracies, standard splits (256 runs (single) / 16 runs (ensemble)). Here and in other tables, we report baselines from the respective papers, comparing to selected methods that at one time were the state of the art.

has been trained, the inference of the mapping $p(c|s)$ and of the target labels by (6) reduces to two matrix multiplications.

Supervised: As we saw above, the label inference using a mixture model can also be seen as a simple neural network. Thus, in a setting with many labeled training nodes, we can perform gradient descent over the entire model — both the CG parameters $\mathbf{W}_{\text{CG}}$ and the mapping $p(c|s)$ encoded in $\mathbf{W}_2$ — to optimize the predictions $\hat{Y}$ computed by (3) over the training labels. In such cases, we initialize the CG parameters with the unsupervised model and treat the gradient descent steps as supervised finetuning.

Ensembling: Classification using generative models of input features benefits from ensembling of multiple models. If several CGs are trained with different random initializations, different local minima may be reached. A given input data point may be well-modeled, or close to a training example, in some models, but not in others. As we show in the experiments and discuss further in the appendix, multiplying the predictions of several models can greatly improve classification accuracies, more than is the case for GNNs.

We provide runnable training code, implementation details, and execution time comparisons in the appendix and supplementary material.

	Cora		**CiteSeer**		**PubMed**	
Method	$K = 5$	$K = 10$	$K = 5$	$K = 10$	$K = 5$	$K = 10$
GCN[1]	66.4±4.3	72.9±3.1	55.6±5.8	64.2±3.9	66.1±3.9	75.6±1.6
GAT[2]	68.2±5.5	73.9±4.3	55.5±1.8	61.6±0.4	64.2±4.8	73.6±1.9
Graph U-Net[3]	64.4±5.4	71.5±3.0	49.4±5.8	61.2±3.5	65.1±4.7	68.7±3.7
GraphMix GCN[4]	72.0±6.5	79.3±1.4	58.6±2.3	**70.8±1.4**	67.7±3.9	**77.1±3.6**
CG ensemble	**72.6±6.1**	**79.9±1.5**	**65.4±4.4**	69.6±1.7	**69.0±5.0**	71.1±3.1

[1]Kipf and Welling (2017) [2]Veličković et al. (2018) [3]Gao and Ji (2019) [4]Verma et al. (2019)

Table 3: Node classification accuracies (%) with $K = 5$ or 10 training nodes per class (16 runs). First four rows from Verma et al. (2019) (10 runs).

3. Experiments: Node classification

In our main experiments, we use the three most commonly studied citation network data sets: Cora (McCallum et al., 2000), CiteSeer (Giles et al., 1998), and PubMed (Unknown). The first two are networks of machine learning papers classified into several subject areas; PubMed is a larger network of medical articles on diabetes, also divided into three subdomains. The data set statistics are shown in Table 1.

We always use 31×31 CGs with window size 5 and model the 2-hop features $\mathbf{X}^{(2)}$. The (lack of) sensitivity to such choices is discussed below.

Standard setting (unsupervised). In the standard benchmark, 20 nodes of each class are provided in training; 500 nodes are used for validation and 1000 for evaluation. We use the standard data splits of Kipf and Welling (2017).

For each graph, we train 256 31 / 5 CGs of the features $\mathbf{X}^{(2)}$. These models are unsupervised and do not depend on the data splits. The feature normalization constant was chosen to be 3 for Cora and CiteSeer and 6 for PubMed; the settings $k = 3, 6$ were the only ones considered for each data set. We use the training nodes to infer the mapping $p(c|s)$ via (5), then evaluate the posterior class probabilities. We consider two variants: a single grid and an ensemble of grids. For ensembling, we take the predictions of 16 grids and multiply the class probabilities of those that have at least the median validation accuracy to obtain the prediction. The smoothing constants α, β are chosen so as to maximize the validation accuracy and kept the same for all runs; we find $(\alpha, \beta) = (4.5, 1)$, $(9, 6)$, and $(3.5, 1)$ for the three graphs. We include the soft nearest neighbor model in the ensemble if it increases the validation accuracy; however, this occurs only for the CiteSeer data set.

The results (Table 2) show that the CG models perform higher than the baselines GCN (Kipf and Welling, 2017) and GAT (Veličković et al., 2018) on all three graphs and improve over the state of the art (Hu et al., 2019) on the Cora data set.

Low-label setting (unsupervised). We evaluate our method when even fewer labels are provided for training. We perform the experiments of Verma et al. (2019): $K = 5$ or 10 labeled nodes of each class are seen in training and the same number in validation, with the rest of nodes used for testing. The experimental setup is otherwise identical to the one for standard splits. The constants (α, β) are found for $K = 10$ and kept the same for $K = 5$; we find $(\alpha, \beta) = (4.5, 1.5)$, $(10, 11)$, $(1, 2.5)$ for the three graphs.

Method	**Cora** acc%	**CiteSeer** acc	**PubMed** acc
GCN[1]	86.0	77.2	86.5
FastGCN[2]	85.0	75.6	88.0
CG pre-tuning	79.80±0.95	72.83±0.64	81.82±0.41
CG supervised	87.42±0.58	77.17±0.59	87.88±0.48
CG sup. ensemble	88.30±0.24	78.02±0.28	88.09±0.21

[1]Kipf and Welling (2017) [2]Chen et al. (2018)

Table 4: Node classification accuracies, large splits (256 runs (single) / 16 runs (ensemble)).

The results are reported in Table 3. CGs tend to greatly improve on the GNN baselines. Below, we present experiments on an even sparser labeling regime: an active learning problem where *no initial labels* are given.

High-label setting (supervised). We study whether CGs can benefit from supervised training in regimes with higher supervision. We use the large data splits of Chen et al. (2018): the test set of 1000 nodes is kept the same as in the standard split; the remaining nodes are split into 500 for validation and the rest for training.

We begin with the same setup as for the standard splits. Once an (unsupervised) CG has been trained, the CG parameters ($\mathbf{W}_{\text{CG}}$) and mappings $p(c|s)$ ($\mathbf{W}_2$) are made trainable as described in "Training" above. We perform full gradient descent through the entire computation graph to minimize the cross-entropy loss of the class predictions over the training set. We make 100 steps with learning rate 10^4 and momentum 0.9. The validation set is used for early stopping. The parameters (α, β) are chosen to maximize the final validation set accuracy; we find $(3.5, 2.5)$, $(3, 2.5)$, and $(2, 1.5)$ for the three graphs. We evaluate the tuned models on the testing set, both single models and products of 16.

The results, in Table 4, show that it is possible to tune the CG mixture model and the linear layer jointly by gradient descent to achieve large increases in performance in high-supervision settings. The finetuned CGs perform better than the baseline GNN models and benefit from ensembling.

Variants and ablation studies. Our models are not very sensitive to the training hyperparameters (which were tuned extensively): Table 5 shows the performance of our models on random splits of the Cora data set, of the same size as the standard splits. In each case, we use the same training settings as for the standard splits and the same search for hyperparameters (α, β) or (h, ε), and report the best result. (It has been argued (Shchur et al., 2018) that this setup provides a fairer evaluation of algorithms, as GNNs are highly sensitive to splits and parameter settings.) The 2-hop features $\mathbf{X}^{(2)}$ appear to be optimal for the mixture models: 1-hop features are too sparse for the model to find an informative clustering, while $\mathbf{X}^{(n)}$ with n too large will "blur" the features and give too much weight to distant nodes. In the appendices, we present additional comparisons to existing models, variants, and studies of different ensembling and finetuning schemes.

		Cora
Method	walk steps	acc%
31 / 5 CG	1	80.14±1.70
31 / 5 CG	2	82.01±1.28
31 / 5 CG	3	81.56±1.25
21 / 3 CG	2	81.88±1.48
11 / 3 CG	2	81.41±1.48
121-cmpt mixture	2	81.37±1.92
soft NN	1	66.85±2.89
soft NN	2	72.50±2.21
soft NN	3	74.53±2.06
soft NN	4	75.39±2.30

Table 5: Accuracies on *random* splits of the Cora data set (32 runs), under various settings of CG sizes, windows, and random walk lengths, as well as simple mixture / soft nearest neighbor models. Note that a 121-component mixture is equivalent to a 11 / 1 CG. Verma et al. (2019) found accuracies of 77.8, 77.7, and 77.6% for baselines GCN (Kipf and Welling, 2017), GAT (Veličković et al., 2018), and Graph U-Net (Gao and Ji, 2019) on such splits.

3.1 Active learning in node classification

We illustrated that we can use CGs either in a standard semi-supervised approach that models the joint distribution of features and labels, or in a discriminative model optimized based on provided (input, label) pairs. We also showed that the two approaches can be combined: having trained a generative CG model of features without any labels, we can refine the model in a supervised manner to minimize the discriminative loss on a labeled set. This flexibility is useful: When we have large amounts of labels, then supervised training can yield better results, while with a small amount of labels, an unsupervised approach is usually better.

We now briefly discuss advantages of models that capture not only the posterior distribution $p(\text{label}|\text{features})$ but the data distribution $p(\text{features})$ in active learning scenarios. In particular, we consider the very first bootstrapping step, when *no labels* are available, but we can see the features of all data points. Given a 31 / 5 CG modeling the features $\mathbf{X}^{(2)}$ of the Cora graph, we seek a mechanism that will ask for some 25 documents to be labeled, so that after their labels are embedded in the grid using (5), the prediction accuracy on the remaining 2683 nodes would be high. We look at two strategies:

(R) randomly sampling the 25 documents;

(G) picking the 25 samples from a uniformly spaced 5×5 grid placed over the CG.

In the latter strategy, for each of the 25 locations in the grid, we sample one of the 2708 documents using their posterior probability of mapping to the location as a weight. Thus the first strategy samples from the data distribution, whereas the second samples more diversely to cover most of the data clusters regardless of their frequency in the data. If the trained model captured the diversity in the data well, then we would expect the latter strategy to perform better.

We trained 32 CG models and tested each of the two strategies by repeating the sampling 50 times. The strategy (G) depends on the grid used for sampling, but we found that the dependence is minimal: while the grids differ in prediction quality, as a sampling guide, they do not differ much. On average, the pretrained CGs using strategy (R) had an accuracy of 62.55% on the remaining 2683 nodes, while with strategy (G) the average was 65.11% (std = 1.2% for both strategies). *Each* grid had a higher accuracy under strategy (G) than under strategy (R) by at least 1% and by as much as 4%. The product ensemble of the 32 grids also did significantly better with grid sampling than with random sampling, with an accuracy of 71.12% (strategy (R)) compared with 68.48% (strategy (G)).

3.2 Application to image classification

In this section, we demonstrate how our methods can be applied to problems beyond graph learning, classification of wearable camera images, extending and improving upon the work of Perina and Jojic (2012); Perina et al. (2017).

Let us briefly summarize this earlier work. We focus on the All-I-Have-Seen (AIHS) data set (Perina and Jojic, 2012), a set of 43522 images captured by a camera worn for several weeks by a human subject. Among these images, 5860 are classified as belonging to one of 45 scenes (office, playground, etc.). The task considered by Perina et al. (2017) is one of semi-supervised classification: given all images and class labels for a small number (1, 2, or 3) of examples of images of each class, to predict the class labels of the remaining images. Each image x is represented by a 4096-dimensional feature vector $f(x)$ by evaluating a deep layer of a pretrained convolutional neural network and renormalizing appropriately.[3] Unsupervised CGs are trained on the representations $f(x)$, and the grid positions with highest posterior are computed for each of the training images. Classes are assigned to each unlabeled image by a simple nearest-neighbor model: by computing its most likely grid position and assigning the label of the training node which is nearest to it (in the Euclidean distance on the torus). This algorithm allowed the authors to obtain impressive results at the ultra-low labeling rate, as measured in classification accuracy on the unlabeled nodes (row CG-NN in Table 6).

In a first experiment, following Perina et al. (2017), we train 64/7 CGs on the same features and evaluate the classification accuracy, with two differences. First, rather than the EM training used in the past CG literature, we adopt the same SGD training algorithm and training settings that we used in the experiments on graphs. Second, instead of assigning labels to evaluation nodes using nearest neighbors, we use the probabilistic inference described in §2. We do not use a validation set, but nor do we perform any hyperparameter tuning; the hyperparameters α and β are fixed at 1. As shown in Table 6 (row $\mathrm{CG}^{(0)}$), our CG algorithms achieve considerably higher classification accuracies than those obtained by Perina et al. (2017).

These algorithms do not take advantage of an essential part of the data set: the ordering of the images through time. Recognizing this, another major contribution of Perina et al. (2017) is an algorithm that uses counting grid positions as states of a hidden Markov

3. The features are the outputs of the fc6 layer of AlexNet (Krizhevsky et al., 2012), divided by 200. This layer is ReLU-activated, so the features are nonnegative.

	train ex / class		
Method	1	2	3
fc6-NN	18.2	19.9	21.6
100-D autoencoder	26.8	26.9	29.5
CG-NN	29.1	31.1	34.6
CG-HMM	31.4	32.6	36.3
$\text{CG}^{(0)}$	37.3	38.6	39.0
$\text{CG}^{(1)}$	47.0	46.7	47.2
$\text{CG}^{(2)}$	50.4	49.5	49.4
$\text{CG}^{(3)}$	50.9	50.6	50.6

Table 6: Image classification accuracies (%) on the AIHS data set with varying number of training images per class. $\text{CG}^{(n)}$ refers to the CG evaluated using the features $\mathbf{X}^{(n)}$. Average over 30 random choices of training nodes. First four rows are from Perina et al. (2017); fc6-NN is a nearest-neighbor classifier on the fc6 features.

model during training, where the transition probabilities enforce that consecutive images are generated from nearby grid positions (row CG-HMM in Table 6).

We take a different approach: we view the data set as a path graph, where images that are adjacent in time are connected by an edge. We apply the transform (1) to the image features $f(x)$ to obtain aggregated feature matrices $\mathbf{X}^{(n)}$, and perform classification using these aggregated features.[4] In the spirit of simplicity and adaptability, we reuse the CGs trained on the unpooled features, but perform label embedding and inference using the pooled features, again with *no validation or parameter tuning.*

The results appear in the lower rows of Table 6. The CGs evaluated with pooled features perform better than all baselines by a wide margins. Remarkably, with more feature pooling, the performance with 1 training example per class begins to exceed that with 3 examples per class, which illustrates at once the strength of these models of pooled features and the importance of the softness of the posterior mapping.

4. Experiments: Link prediction

In the link prediction task, a random sample of 15% of the edges is deleted from a graph; the objective is to predict which of the unlinked node pairs in the damaged graph are present in the original graph. One-third of the deleted edges are used for validation and the rest for testing. An equal number of non-edges are used for validation and testing.

Attributed graphs. We train CGs on features $\mathbf{X}^{(2)}$ derived from the incomplete graph. The reconstructed probabilities of links $p(i \sim j)$ are computed using (8) and used for scoring, with the hardening constant c chosen to maximize the result on the validation set. We also experiment with using the score $p(i \sim j)(\deg(i)+1)(\deg(j)+1)$ for the link between i and j, where $\deg(i)$ is the degree of i in the damaged graph, which optimizes for the metric given certain assumptions about the distribution of node degrees. Finally, we consider ensembles, where 8 CGs are trained on the same damaged graph and the link probabilities

4. The pooled features are divided by $2n+1$, as would be done for the categorical features in our main experiments if the graph were a path.

	Cora		**CiteSeer**		**PubMed**	
Method	AUC%	AP%	AUC	AP	AUC	AP
(V)GAE[1]	91.4	92.6	90.8	92.0	96.4	96.5
MTGAE[2]	94.3	95.2	95.6	96.4	96.0	96.3
$\mathcal{S}/\mathcal{N}$-VGAE[3]	94.1	94.1	94.7	95.2	97.1	97.1
SCAT[4*]	93.9	93.9	94.7	95.2	94.6	94.4
sGraphite-VAE[5]	93.7	93.5	94.1	95.4	94.8	96.3
CG	94.4±0.7	94.6±0.7	**95.8±0.7**	**96.0±0.8**	95.8±0.2	95.6±0.2
CG ensemble	**95.6±0.5**	**95.7±0.6**	**96.9±0.5**	**96.9±0.5**	96.4±0.2	96.2±0.3
CG + deg	94.6±0.7	**95.0±0.7**	**95.6±0.6**	**95.9±0.5**	**97.2±0.2**	97.1±0.2
CG + deg ensemble	**95.7±0.2**	**96.0±0.5**	**96.4±0.6**	**96.5±0.6**	**97.5±0.2**	**97.5±0.2**

[1]Kipf and Welling (2016) [2]Tran (2018a,b) [3]Davidson et al. (2018) [4]Zou and Lerman (2019) [5]Di et al. (2020)
[*]Our result for Zou and Lerman (2019) fixes a bug in the published code's evaluation of test metrics (see Appendix F for details).

Table 7: Results on the link prediction task in attibuted graphs (16 runs). We report area under the ROC curve (AUC) and precision-recall curve (AP). + deg: degree-weighted score.

	Cora		**CiteSeer**		**PubMed**	
Method	AUC%	AP%	AUC	AP	AUC	AP
SocioDim[1]	84.6	88.5	80.5	85.0	84.2	87.8
DeepWalk[2]	83.1	85.0	80.5	83.6	84.4	84.1
(V)GAE[3]	84.3	88.1	78.9	84.1	82.7	87.5
MTGAE[4]	89.6	**91.5**	86.0	**89.2**	92.6	93.0
CG	89.1±1.2	90.3±1.1	**87.5±1.1**	**89.2±1.1**	90.8±0.2	90.9±0.2
CG ensemble	**90.2±0.8**	91.3±0.9	**86.2±1.1**	88.2±1.0	91.6±0.2	91.7±0.2
CG + deg	89.2±1.2	90.7±1.0	**87.2±1.2**	**89.2±1.1**	**94.6±0.2**	**94.7±0.1**
CG + deg ensemble	89.5±1.0	91.4±0.8	84.4±1.3	87.1±1.0	**94.9±0.1**	**95.0±0.1**

[1]Tang and Liu (2011) [2]Perozzi et al. (2014) [3]Kipf and Welling (2016) [4]Tran (2018b)

Table 8: Results on the link prediction task in featureless graphs (16 runs). First three rows from Kipf and Welling (2016).

are averaged over the models to produce the prediction. The CG hyperparameters were the same as in the node classification tasks — *no hyperparameter tuning was done.*

As shown in Table 7, the CGs perform higher than the previous state of the art on all three graphs. We do not exclude the possibility that the scores could be even higher with other CG sizes, feature normalizations, etc. Furthermore, as illustrated in Figure 2, the link probabilities tend to be high for pairs of nearby positions. The CGs perform strongly even when the link probability (8) is replaced by a *fixed* function of the distance between the two grid positions (see the appendix).

Featureless graphs. We now consider link prediction in graphs where the nodes do not have associated features — the only data provided is the adjacency structure.

We use the same setup as above, with two changes. First, we use as node features the one-hot encoding of the node indices. (For the large PubMed graph, we encode only the vertices that have degree at least 5 in the damaged graph, to reduce the feature dimension.) Second, we choose to model the 4-hop random walk features $\mathbf{X}^{(4)}$.

In this setting, CGs continue to perform strongly (Table 8), sometimes even exceeding the performance of baseline models on the graphs *with* features (Table 7).

5. Conclusion

We found that a simple normalization and aggregation of node features using short random walks in (1) renders the recently intensely studied graph problems easy enough that the traditional semi-supervised approaches from the last millennium (Seeger, 2002; McLachlan, 1992) often work as well as popular graph neural networks. Such a transformation of the input data led us to develop models that mirror in their parameter space the intersecting nature of short random walks, leading to superior performance to the current state of the art in several node classification and link prediction tasks.

The generative approach produces models that are less sensitive to hyperparameters, are readily ensembled, and can be tuned as neural network layers with self-supervised pre-training. Applications to graph clustering and visualization, inductive node classification problems, tasks with larger graphs and graphs with non-categorical features, and the use of self-supervised CG layers in deeper models are interesting subjects for future work.

The intersecting representation using CGs is a natural choice for modeling graph features, but this approach can transfer to other settings. The parametrization of a weight matrix using overlapping windows to activate a grid of related neurons (11) is applicable to any neural net layer. Other kinds of data, with no explicit graph structure, may benefit from such an approach. For example, our results, together with the high performance of CGs in low-label text classification (Jojic and Perina, 2011; Perina et al., 2013), suggest approaches to modeling of overlapping text spans for more complex language tasks.

We direct the reader to the appendix for further experiments, comparisons, visualizations, and discussion.

Appendix A. Experiment details

A.1 Code

The code repository `https://github.com/malkin1729/graph-cg` contains example code that illustrates our algorithms. The code is runnable "out of the box" on a typical machine equipped with a GPU (see the README file for commands and library prerequisites).

The example code allows to:

(1) Train a CG on the Cora data set.
(2) Evaluate the model, using the trained CG or a pretrained model (included), on the node classification task.
(3) Visualize the label embeddings of a (pre)trained CG.
(4) Finetune the CG on a large training set.
(5) Train a CG and evaluate it on the link prediction task.

We have tried to make the example code as simple as possible so that the reviewer can read and run it, while still showing the main algorithms. Full code will be released upon publication.

A.2 Implementation and hyperparameters

We implemented our CG algorithms in Matlab (for EM on CPU) and the Torch package Paszke et al. (2017) (for SGD on GPU).

As noted in the main text, the CG microtopic probabilities $\mathbf{W}_{\mathrm{CG}}$ were parametrized in the log domain. As is commonly done, the probabilities were clipped from below: we always clipped $\mathbf{W}_{\mathrm{CG}}$ to the range $[-3, 3]$, so the ratio of probabilities of two features under any microtopic could not exceed $e^6 \approx 400$. This clipping parameter was not tuned. The CGs were trained with 8192 (link prediction) or 65536 (node classification) batches of size 256; we used the Adam optimizer Kingma and Ba (2014) with learning rate 0.01 (LP) or 0.001 (NC).

The parameters of CG grid size K, window size W, and feature normalization constant k were chosen on the basis of small experiments with random splits and visual inspection of the inferred label embeddings: we considered $K \in \{21, 31\}$, $W \in \{3, 5, 7\}$, $k \in \{3, 6\}$.

The only parameters that were systematically tuned were:

- For node classification with CGs, the smoothing constants (α, β). We considered $\alpha, \beta \in \{1, 1.5, 2, 2.5, \ldots, 12\}$.

- For soft nearest neighbor models, the constants h and ε. We considered $h \in \{0.5, 1, 1.5, 2, \ldots, 3\}$ and $\varepsilon \in \{0.01, 0.03, 0.1, 0.3, 1\}$.

- For link prediction, the hardening constant c. We considered $c \in \{1, 2, 3, \ldots, 10\}$.

None of these constants affect the unsupervised CG training, which is the computation bottleneck. Thus the hyperparameter search and inference gave almost no computation overhead.

	Cora	**CiteSeer**	**PubMed**
Epoch time (batch size 256)	140ms	427ms	441ms
Epoch time (batch size full)	29ms	65ms	45ms
EM iteration (CPU)	640ms	1670ms	2070ms

Table 9: Execution time comparison

A.3 Complexity and execution time

The theoretical number of arithmetic operations to compute the posterior of an input sample (in $\mathbb{R}^F$) under a K / W CG is $O(K^2F)$: the softmax and average pooling to compute $\mathbf{W}_1$ take $O(K^2)$ operations; the tensor contraction of the resulting $K^2 \times F$ matrix with the input takes $O(K^2F)$ operations. Thus both a single training iteration and model evaluation on N node examples (for node classification) have a theoretical complexity of $O(NK^2F)$.

For link prediction, computation of the link probability for every pair of nodes by (8) and (9) involves computing two products of three matrices: $(K^2 \times N), (N \times N), (N \times K^2)$ and $(N \times K^2), (K^2 \times K^2), (K^2, N)$. Naïvely, the complexity is $O(N^2K^2 + NK^4)$, but the $N \times N$ adjacency matrix is sparse, so optimizations are possible. This step is performed only once and is practically instant.

In practice, when we use libraries optimized for GPU computations, the computation bottleneck turns out to be in the softmax and average pooling, not in the matrix multiplications. The average training time for a single epoch of training by gradient descent (on our Tesla K80 GPU) or a single iteration of the EM algorithm from Jojic and Perina (2011) (on 4 cores of our 3.70 GHz Intel Xeon E5 CPU) is shown in Table 9. The most expensive part of the computation is the computation of the matrix $\mathbf{W}_1$ (equation (11)). For this reason, training is more efficient with larger batch sizes; the memory requirement is low enough that training with full gradient descent is possible on all three data sets with 31 / 5 CGs.

In the evaluation phase, this matrix can be precomputed, as it depends only on the parameters $\mathbf{W}_{\text{CG}}$, not input data. The computation of posteriors becomes a single matrix multiplication and softmax.

Appendix B. Visualization

Included in the code repository are two visualizations of CGs:

(1) `cg-vocab-cora.html` and `cg-vocab-pubmed.html` are visualizations of counting grids trained on the Cora and PubMed data sets. In each grid cell, we show the three words with highest probability under the corresponding microtopic, normalized by the words' probabilities averaged over the entire grid.
We were not able to obtain a vocabulary index for Cora. However, the data set we use is a subset (in both nodes and vocabulary) of a larger citation network for which vocabulary is available. The vocabulary shown is based on a likely alignment of documents and words between the small and large data sets found using classical statistical machine translation techniques.
The vocabulary for PubMed is available in certain distributions of the data set. The vocabulary words are stemmed.

(2) `cg-walk.mp4` is a video of a random walk on the Cora citation graph. At the n-th video frame, we show the log posterior over grid positions for the n-th node in the random walk. Observe that most steps are short in the latent space and are between documents with the same class label, but there are occasional jumps to distant parts of the grid.

Applications of CGs to graph visualization are a potential subject for future work. As we saw in the main text, nodes that are close to the graph, and nodes belonging to the same category, tend to have posteriors concentrated at nearby positions in a trained CG. Thus CGs may provide a useful summary of the graph topology. Similarly, an unsupervised graph clustering method may begin with a trained CG and aggregate nearby squares to form clusters, "tearing" the grid at areas with low posterior.

Appendix C. Additional studies and comparisons

C.1 Ensembling methods

In this section we compare methods for ensembling the predictions of several CGs. We considered seven methods, two of which had access to the validation set accuracies of the individual models. The comparisons are performed with the same set of smoothing constants (α, β), found on the standard split.

The following methods were compared:

- Sum of output class probabilities from 16 CGs.
- Product of output class probabilities from 16 CGs.
- Product of output class probabilities: multiply the predictions of 8 most certain (minimum entropy) for each sample.
- Product of output class probabilities: multiply the predictions of 8 least certain (maximum entropy) for each sample.
- Mixture: Use a generative model that is a uniform mixture of the 16 individually trained CGs. This is a mixture model with $16 \cdot 31^2$ components. Perform inference in the same way.
- Product and mixture (best 8 of 16): product or mixture of the CGs that have at least the median validation accuracy (among the 16).

The accuracies, on random splits, are shown in Table 10. In addition, we include the CGs finetuned on the 140 training samples, in the same setup as used for finetuning in the high-label case. We see that supervised finetuning has some benefit even in low-label settings, and that the accuracy increases as a result of ensembling no matter which ensembling strategy is used. (We chose the CG product strategy, which is statistically indistinguishable from CG mixture.)

Ensembling is a common way of improving model performance. However, as discussed in the main text, we suggest that a generative model of the kind we study benefits more from such an approach than discriminatively trained GNNs. We performed the same experiments using the *sum* ensembling method on the Cora data set, using the baseline models GCN

Method	Cora acc%
GCN[1]	77.84±1.45
GAT[2]	77.74±1.86
Graph U-Net[3]	77.59±1.60
GraphMix + GCN[4]	82.07±1.17
CG	80.54±1.99
CG supervised	81.04±1.94
CG sum (16)	83.07±1.25
CG product (16)	83.11±1.27
CG product (min ent 8 of 16)	83.06±1.28
CG product (max ent 8 of 16)	82.76±1.11
CG mixture (16)	83.05±1.26
CG product (best 8 of 16)	83.39±1.31
CG mixture (best 8 of 16)	83.45±1.28

[1]Kipf and Welling (2017) [2]Veličković et al. (2018) [3]Gao and Ji (2019) [4]Verma et al. (2019)

Table 10: *Left:* Accuracies on *random* splits of the Cora data set (256 runs (single) / 16 runs (ensembles)) under different tuning / ensembling schemes. The italicized methods had had access to the validation set. The first four rows are from Verma et al. (2019).

Method	Cora acc%	CiteSeer acc	PubMed acc
DeepWalk[1]	67.2	43.2	65.3
Planetoid[2]	75.7	64.7	77.2
GCN[3]	81.5	70.3	79.0
GAT[4]	83.0	72.5	79.0
Graph U-Net[5]	84.4	73.2	79.6
GMNN[6]	83.7	73.1	**81.8**
LDS-GNN[7]	84.1	**75.0**	–
H-GCN[8]	84.5	72.8	79.8
SGCN[9]	81.0	71.9	78.9
G^{3}NN[10]	82.9	74.5	78.4
CG	82.24±1.05	72.49±0.59	79.46±0.71
CG ensemble	**84.68±0.45**	72.68±0.36	80.18±0.26

[1]Perozzi et al. (2014) [2]Yang et al. (2016) [3]Kipf and Welling (2017) [4]Veličković et al. (2018) [5]Gao and Ji (2019) [6]Qu et al. (2019) [7]Franceschi et al. (2019) [8]Hu et al. (2019) [9]Wu et al. (2019) [10]Ma et al. (2019)

Table 11: Accuracies on the node classification task.

and GAT, training each model 16 times using the reference implementations with default parameter settings. With ensembling, we obtained an improvement from $81.5 \pm 0.7\%$ to $82.2 \pm 0.2\%$ (GCN) and only from $83.2 \pm 0.7\%$ to $83.5 \pm 0.2\%$ (GAT), compared with an improvement of 2.4% with CGs.

C.2 Standard split node classification

In Table 11 we have included some additional comparisons on the node classification task (standard splits).

We would like to emphasize the contribution of Wu et al. (2019). In that paper, it was shown that simpler models approach the performance of GCNs on node classification tasks: they introduce "Simple GCNs", which are logistic regression models trained on the node features processed by a fixed linear aggregation rule:

$$\mathbf{X}_{\text{agg}} = \left((\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}\right)^n \mathbf{X},$$

where for the graphs we study $n = 2$.

Although this linear model is also motivated by a "historic omission of a simpler predecessor" (Wu et al. (2019)) to complex GNN models, our model has important differences. While the Simple GCN is a softmax model mapping the aggreated features to the labels, our softmax model maps features to prototypes or mixture indices, then performs an assignment of output labels to the prototypes or indices. The architecture of our method, described this way, is equivalent an unsupervised generative model, where the inference in classification tasks constitutes the index-label assignment — different from the direct approximation of the feature-training label mapping done by Wu et al. (2019).

Our method has certain advantages, at the cost of a larger computation time: it has higher performance on the node classification task, works simultaneously on many classification tasks with no retraining, can be used for active learning, and the same representation works for link prediction problems.

C.3 The importance of unsupervised initialization

We have demonstrated how a trained CG can be finetuned in a supervised way, with both the CG microtopic probabilities and the mapping $p(c|s)$ made trainable. This raises the question of our how method would perform without the initalization of parameters with an unsupervised CG, but rather trained end to end from random initialization. Using the same training settings as for finetuning with unsupervised initialization, and early stopping based on validation set accuracy, we obtain test accuracies on the standard splits of 78.9% (Cora), 68.3% (CiteSeer), and 78.2% (PubMed), considerably lower than using the unsupervised models (Table 11).

C.4 High-label node classification

We show comparisons of a larger set of methods on the high-label node classification task in Table 12. Note the impressively performing kernel method of Tian et al. (2019). This method also applies nearest neighbor-like models to a node representation derived by a simple *learned* aggregation scheme.

C.5 Link prediction with distance-based probabilities

As we noted above, the posterior embeddings of the documents in a trained CG are such that links are likely to be short. (See also the video of the log posterior distributions of

Method	**Cora** acc%	**CiteSeer** acc	**PubMed** acc
GCN[1]	86.0	77.2	86.5
GAT[2]	85.6	76.9	86.2
GraphSAGE[3]	82.2	71.4	87.1
FastGCN[4]	85.0	75.6	88.0
ASGCN[5]	87.4	79.7	90.6
DropEdge (best)[6]	88.2	**80.5**	**91.7**
$\mathcal{K}_3$ kernel[7]	**88.4**	80.3	89.4
CG pre-finetuning	79.80±0.95	72.83±0.64	81.82±0.41
CG supervised	87.42±0.58	77.17±0.59	87.88±0.48
CG supervised ensemble	88.30±0.24	78.02±0.28	88.09±0.21

[1]Kipf and Welling (2017) [2]Veličković et al. (2018) [3]Hamilton et al. (2017) [4]Chen et al. (2018) [5]Huang et al. (2018) [6]Rong et al. (2020) [7]Tian et al. (2019)

Table 12: Node classification task (large standard split). We report average and standard deviation over 256 runs (single) or 16 runs (ensemble). Note that "CG pre-finetuning" refers to the models evaluated before supervised training, but with the same smoothing constants. The optimal performance with untuned models is higher, as it would use different smoothing constants.

documents during a random walk over the Cora graph.) In the extreme, we can do link prediction by replacing (8) with

$$p(s_1 \sim s_2) = \begin{cases} 1 & s_1 = s_2 \\ 0 & s_1 \neq s_2 \end{cases}.$$

Equivalently, the predicted probability of a link between two nodes is the scalar product of their posterior probability vectors:

$$p(i \sim j) = \sum_s p(s|\mathbf{X}_i^{(2)})p(s|\mathbf{X}_j^{(2)}).$$

Such a simple scheme would not overfit to the edges present in the damaged graph as much as the computation of full $p(s_1 \sim s_2)$ by (8). In Table 13 we include this method (marked *); we see that it performs nearly as well as the method in the main text, and in some cases better. However, the gain from ensembling appears to be smaller with this method.

Appendix D. Additional applications

D.1 Hierarchical CGs

We discussed how local minima in packing data samples into a grid can damage the performance of a single CG. A remedy we proposed is to ensemble multiple CGs, assuming that each data sample will be embedded well with respect to its neighbors in at least some of the grids. An alternative to this approach would be to spend more time searching for a high-likelihood grid. In Jojic et al. (2016), it was shown that one way to get better local

	Cora		**CiteSeer**		**PubMed**	
Method	AUC%	AP%	AUC	AP	AUC	AP
(V)GAE[1]	91.4	92.6	90.8	92.0	96.4	96.5
MTGAE[2]	94.3	95.2	95.6	96.4	96.0	96.3
AR(V)GE[3]	92.4	93.2	92.4	93.0	96.8	97.1
$\mathcal{S}/\mathcal{N}$-VGAE[4]	94.1	94.1	94.7	95.2	97.1	97.1
Kernel method[5]	93.1	93.2	90.9	91.8	94.5	94.2
SCAT[6]	93.9	93.9	94.7	95.2	94.6	94.4
sGraphite-VAE[7]	93.7	93.5	94.1	95.4	94.8	96.3
CG	94.4±0.7	94.6±0.7	**95.8±0.7**	**96.0±0.8**	95.8±0.2	95.6±0.2
CG ensemble	**95.6±0.5**	**95.7±0.6**	**96.9±0.5**	**96.9±0.5**	96.4±0.2	96.2±0.3
CG + deg	94.6±0.7	**95.0±0.7**	**95.6±0.6**	**95.9±0.5**	**97.2±0.2**	97.1±0.2
CG + deg ensemble	**95.7±0.2**	**96.0±0.5**	**96.4±0.6**	**96.5±0.6**	**97.5±0.2**	**97.5±0.2**
CG*	**94.7±0.5**	94.6±0.6	**95.6±0.6**	95.5±0.7	95.4±0.2	95.1±0.3
CG ensemble*	**95.3±0.4**	**95.3±0.5**	**96.3±0.5**	**96.3±0.6**	95.7±0.2	95.5±0.3
CG + deg*	**95.4±0.5**	**95.4±0.6**	**95.9±0.5**	**95.9±0.6**	**97.2±0.2**	**97.2±0.2**
CG + deg ensemble*	**95.9±0.5**	**95.9±0.6**	**96.5±0.4**	**96.5±0.5**	**97.4±0.1**	**97.3±0.2**

[1]Kipf and Welling (2016) [2]Tran (2018a,b) [3]Pan et al. (2018) [4]Davidson et al. (2018) [5]Tian et al. (2019) [6]Zou and Lerman (2019) [7]Di et al. (2020)

Table 13: Results on the link prediction task (16 runs). We report area under the ROC curve (AUC) and precision-recall curve (AP); Tran (2018b) gives only the mean of the two. + deg: degree-weighted score. *: scalar product of posterior probability vectors (see text).

minima is to train a *hierarchical* CG model made of a series of grids. These grids, with the exception of the lowest grid in the hierarchy, model not the data features but *locations* in the grid below. The authors show that such a deep generative CG model can be collapsed to a single feature CG of the size of the top parent grid. However, training a hierarchy leads to models with higher perplexity than training directly the equivalently sized feature CG. The authors did not consider additional supervised over-training (nor did they study graph embedding) as we did here, so it is possible that stacking multiple softmax-linear layers pretrained using a hierarchical CG model can lead to results that would improve over the ones we report here (which already often beat the state of the art).

D.2 Logistic grids: an alternative overlapping representation

Here we give an example of another overlapping representation: *logistic grids* (LGs). While CGs reparametrize a uniform categorical mixture model by blurring microtopic parameters in windows, LGs blur them in the log domain. They are another example of how a linear neural net layer can be converted into one in which inputs activate a grid of related neurons.

Recall that a CG is defined by a matrix of microtopic parameters $\mathbf{W}_{\mathrm{CG}} \in \mathbb{R}^{F\times(K\times K)}$, from which the matrix of feature log-probabilities is computed as

$$\mathbf{W}_1^\top = \log\left(\mathrm{AvgPooling}_{W\times W}(\mathrm{softmax}(\mathbf{W}_{\mathrm{CG}}))\right). \tag{12}$$

A K / W LG is also described by a matrix $\mathbf{W}_{\mathrm{LG}}$. To compute $\mathbf{W}_1^\top$, we simply exchange the softmax and blurring:

$$\mathbf{W}_1^\top = \log\left(\mathrm{softmax}(\mathrm{AvgPooling}_{W\times W}(\mathbf{W}_{\mathrm{LG}}))\right). \tag{13}$$

Inference using the $\mathbf{W}_1^\top$ in the LG parametrization is equivalent to inference in a different generative model. In CGs, a feature generated from the mixture component corresponding to a window is generated by *one* of the grid cells in the window:

$$p(j|s=(X,Y)) = \frac{1}{W^2} \sum_{(x,y) \text{ in window at } (X,Y)} \exp\left(\log p_{(x,y)}(j)\right),$$

where the $\log p_{(x,y)}(j)$ are encoded in the matrix $\mathbf{W}_{\mathrm{CG}}$. In LGs, the feature is simultaneously generated by *all* grid cells in the window: if logits $u_{(x,y)}(j)$ are encoded in the matrix $\mathbf{W}_{\mathrm{LG}}$, then

$$p(j|s=(X,Y)) \propto \exp\left(\frac{1}{W^2}\sum_{(x,y)} u_{(x,y)}(j)\right).$$

We can see in the form of this equation that the model differs from CG in that the microsources $u_{(x,y)}$ modulate the distribution in the window multiplicatively, rather than additively.

It may seem that the LG parametrization (13) is equivalent to a simpler variant:

$$\mathbf{W}_1^\top = \log(\mathrm{softmax}(\mathbf{W}_{\mathrm{LG}'})),$$

which is just a uniform mixture model with K^2 components that does not depend on the grid structure and where the benefits of the overlapping representation, including capacity control described in Sec. E, are lost. It is true that, under certain conditions, given $\mathbf{W}_{\mathrm{LG}'}$, it is possible to find $\mathbf{W}_{\mathrm{LG}}$ such that $\mathrm{AvgPooling}_{W\times W}(\mathbf{W}_{\mathrm{LG}}) = \mathbf{W}_{\mathrm{LG}'}$.[5] However, this behavior is prevented by both (1) clipping of parameters $\mathbf{W}_{\mathrm{LG}}$, just as is done for CG parameters $\mathbf{W}_{\mathrm{CG}}$, and (2) the inductive biases of learning algorithms: even without clipping, gradient descent converges to LGs where the mixture components still vary smoothly over the grid, as shown in Figure 3.

Like CGs, LGs can be applied to various kinds of data. In Figure 3 we show a 100 / 5 LG trained on the MNIST data set of handwritten digits. We treat each of the 28^2 image pixels as a binary feature. In each position of the grid, we visualize the distributions over these pixel features for the 5×5 window starting at that position, proportional to $\exp(\sum \frac{1}{W^2} u_{(x,y)}(j))$. The color intensity is scaled by the similarity to the nearest training example. Thus, the bright digits faithfully copy the training samples, and the rest are interpolations among them, demonstrating various transformations from one digit to the next. The grid itself is larger than the data set, yet due to gradient descent learning of the model, the embedding appears ordered, and the intermediate digits plausible.

We briefly tested LGs on the node classification task, simply replacing (12) by (13), and found similar performance. It remains to be seen how these two different approaches to modeling intersections would fair in multi-layer models and different data sets.

5. This amounts to solving a certain linear system, which is nonsingular if K and W are coprime.

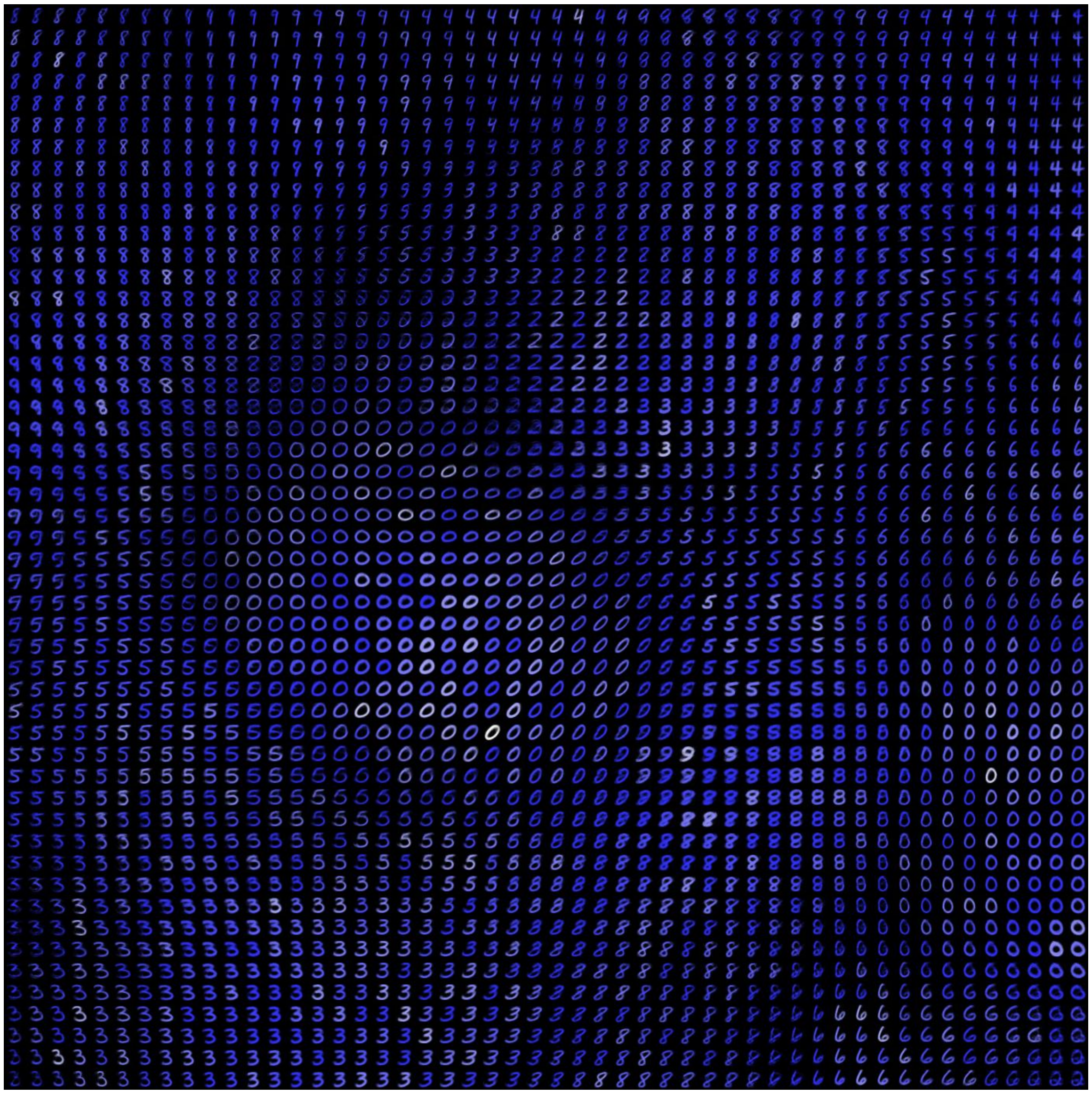


Figure 3: A piece of a 100 / 5 LG trained on the MNIST data set. See the file `LG.jpg` in the code repository for the full grid.

Appendix E. On the capacity of CGs

In this section we describe an heuristic that provides motivation for the comparisons in the main text.

A K / W CG is a parametrization of a K^2-component uniform mixture model. Its essential property is that the mixture components at nearby grid positions are not independent. We define the capacity to be the maximum size of a set of independent components — that is, the maximum number of grid positions that can be chosen so that the $W \times W$ windows around them do not overlap. This number is approximately $(K/W)^2$.

If we have chosen such a set of independent components, the other mixture components will interpolate between them: the expression of the mixture components as microtopics blurred in windows forces the components' parameters to vary smoothly over the grid. Thus the capacity can be used as a proxy for the CG's expressive power and ability to generalize.

A 7 / 1 CG, a 21 / 3 CG, and a 35 / 5 CG all have the same capacity. The 7 / 1 CG is a simple mixture model, in which all components are independent. In the 21 / 3 CG, overlaps are possible: the components corresponding to neighboring grid squares share $\frac{2}{3}$ of their microtopics. In the 35 / 5 CG, even larger overlaps are possible.

A 7 / 5 CG would allow little variation of mixture topics over the grid, while a 35 / 1 CG is a 1225-component mixture that would be prone to overfitting on a small set of examples. Experimental evidence shows that the CGs with approximately equal capacity tend to have comparable performance. For example, in the setting of Table 4 in the main text, a 31 / 3 CG (capacity ~ 107, far larger than the other grids in the table) has an average accuracy of just 79.6%.

E.1 Future work

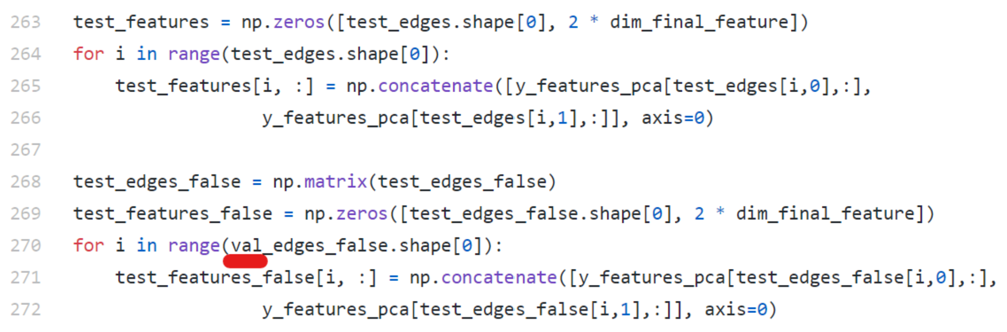
It would be interesting to study the performance of CGs in graphs with non-categorical features. While categorical distributions are a natural choice for modeling bags of words, in other settings, we may choose to model node features using Gaussians, log-normals, etc. The overlapping parametrization is possible here as well. Beyond CGs and logistic grids, one can imagine (and we have implemented, for image data) "Gaussian grids", where each grid position generates features from a Gaussian with mean and variance parameters averaged over a window (or, alternatively, a uniform mixture of the Gaussians in a window). However, interpreting real-valued features as positive or negative features, as we did in §3.2, can already work well in practice.

Appendix F. Evaluation errors in other work on link prediction

In this section we document and provide evidence for what we believe to be critical flaws in computation of evaluation metrics in two papers that have claimed to be the state of the art in link prediction on the Cora, CiteSeer, and PubMed data sets.

F.1 SCAT (Zou and Lerman, 2019)

We believe there is an error in the published code of Zou and Lerman (2019), which we considered to be the highest published result on the data sets we study. In our experiments, we have implemented what we think is a fix, which accounts for the discrepancy between



```
263  test_features = np.zeros([test_edges.shape[0], 2 * dim_final_feature])
264  for i in range(test_edges.shape[0]):
265      test_features[i, :] = np.concatenate([y_features_pca[test_edges[i,0],:],
266                  y_features_pca[test_edges[i,1],:]], axis=0)
267
268  test_edges_false = np.matrix(test_edges_false)
269  test_features_false = np.zeros([test_edges_false.shape[0], 2 * dim_final_feature])
270  for i in range(val_edges_false.shape[0]):
271      test_features_false[i, :] = np.concatenate([y_features_pca[test_edges_false[i,0],:],
272                  y_features_pca[test_edges_false[i,1],:]], axis=0)
```

Figure 4: Bug in Zou and Lerman (2019): we believe `val_edges_false` should be `test_edges_false`.

	original data split		flipped data split	
Code	val	test	val	test
SCAT (published)	93.06	96.26	*out-of-bounds ex.*	
SCAT (our fix)	93.06	92.95	92.95	93.06
CG	92.85	94.27	94.27	92.85

Table 14: Mean of AUC and AP scores with original (fixed) data split and flipped data split, with and without our suggested code change.

Table II of Zou and Lerman (2019) and Table 7. While SCAT presents an interesting approach and has quite a strong performance on the benchmark data sets, the reported test results appear to be overestimated by about 2%.

To our judgment, the error occurs when preprocessed node features are being written to arrays that will then be provided as inputs to a fully connected network.[6] The test set is twice as large as the validation set; however, only as many entries of the feature array for non-edge examples are populated as there are validation edges. That is, the second half of the negative test examples are always provided with input the 0 vector. A screenshot of this simple but hard-to-spot bug is in Figure 4.

As a probe, we perform a diagnostic test on the Cora data set. We fix all random seeds and fix a data split, then execute the example command `python trainCora.py -s S -d cora`, and examine the final validation and test accuracies. (We verified that the code produces identical outputs on several runs.) We then *interchange* the validation and test sets and attempt to run the same code. We follow these steps both with the published code and with our correction implemented.

As shown in Table 14, the published code fails to run with the flipped data, because there are more validation nodes than entries in the test feature array, while on the original data the discrepancy between test and validation scores is suspiciously high. However, with

6. Line 270 at `https://github.com/dmzou/SCAT/blob/40188296b034a2b81b480966961f20b9617fc192/trainCora.py`.

our correction, the validation and test accuracies change places, as expected, when the two sets are interchanged.

We have notified the authors of Zou and Lerman (2019) of this issue, but have not received a response as of this writing. In addition, we verified that our evaluation code does not have the same flaw (last row of Table 14).

F.2 GraphStar (Haonan et al., 2019)

The preprint Haonan et al. (2019) also claims to have achieved the state-of-the-art performance on the link prediction task, but we believe it also suffers from flaws in evaluation.

The published code loads and processes the graph data using the PyTorch Geometric package.[7] After the train/validation/test splits have been created, the positive examples of validation and test edges are symmetrized — a copy of each edge is inserted into the sets with its end nodes interchanged.[8] However, the negative examples are not symmetrized. As a result, the AUC and AP scores are computed over a set with a *duplicate copy of every positive validation/test example*, but a single copy of each negative example. The AUC score does not depend on such a change, but the AP score increases (by a simple derivation, when the positive examples are duplicated, the recall at a given score threshold is unaffected, but the precision p becomes $\frac{2p}{1+p}$).

We verified that twice as many positive examples as negative examples are indeed given as input to the scoring function. This behavior differs from that of other implementations, notably, the published code of baseline (V)GAE (Kipf and Welling, 2016)[9], as well as our code. This renders the results in Table 8 of Haonan et al. (2019) incomparable.

GraphStar may have other issues as well. For example, the maximum test accuracy is output as an evaluation metric. This is not a valid evaluation metric: according to Haonan et al. (2019) the model from the epoch with highest *validation* accuracy is used for testing — but this is not what is output by the published code.[10] Furthermore, the highest test score is computed as the mean of the maximum AUC score and the maximum AP score, which is possibly higher than the maximum mean of AUC and AP scores.

We have also notified the authors of the issue, but have not yet received a response. We verified that our code uses the same number of positive and negative examples for evaluation.

7. `https://pytorch-geometric.readthedocs.io/`
8. Lines 25-27 at `https://github.com/graph-star-team/graph_star/blob/0cb3f4841eff5f4e41b0a728a8164a6a44d91991/run_lp.py`.
9. `https://github.com/tkipf/gae`
10. Lines 307-321 at `https://github.com/graph-star-team/graph_star/blob/0cb3f4841eff5f4e41b0a728a8164a6a44d91991/trainer.py`.

References

Monica Agrawal, Marinka Zitnik, and Jure Leskovec. Large-scale analysis of disease pathways in the human interactome. *Pacific Symposium on Biocomputing*, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *International Conference on Learning Representations (ICLR)*, 2015.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. *International Conference on Learning Representations (ICLR)*, 2018.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical Variational Auto-Encoders. *Uncertainty in Artificial Intelligence (UAI)*, 2018.

Xinhan Di, Pengqian Yu, Rui Bu, and Mingchao Sun. Mutual Information Maximization in Graph Neural Networks. *International Joint Conference on Neural Networks (IJCNN)*, 2020.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning Discrete Structures for Graph Neural Networks. *International Conference on Machine Learning (ICML)*, 2019.

Hongyang Gao and Shuiwang Ji. Graph U-Nets. *International Conference on Machine Learning (ICML)*, 2019.

C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: an automatic citation indexing system. *International Conference on Digital Libraries*, 1998.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. *International Conference on Machine Learning (ICML)*, 2019.

William Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *Neural Information Processing Systems (NIPS)*, 2017.

Lu Haonan, Seth H. Huang, Tian Ye, and Guo Xiuyan. Graph Star Net for Generalized Multi-Task Learning. *arXiv 1906.12330*, 2019.

Fenyu Hu, Yanqiao Zhu, Shu Wu, Liang Wang, and Tieniu Tan. Hierarchical Graph Convolutional Networks for Semi-supervised Node Classification. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive Sampling Towards Fast Graph Representation Learning. *Neural Information Processing Systems (NIPS)*, 2018.

Aapo Hyvärinen, Patrik O. Hoyer, and Mika Inki. Topographic Independent Component Analysis. *Neural Computation*, 13:1527–1558, 7 2001.

Tomoharu Iwata, Takeshi Yamada, and Naonori Ueda. Probabilistic Latent Semantic Visualization: Topic Model for Visualizing Documents. *Knowledge Discovery and Data Mining (KDD)*, 2008.

Nebojsa Jojic and Alessandro Perina. Multidimensional counting grids: Inferring word order from disordered bags of words. *Uncertainty in Artificial Intelligence (UAI)*, 2011.

Nebojsa Jojic, Alessandro Perina, and Kim Dongwoo. Hierarchical learning of grids of microtopics. *Uncertainty in Artificial Intelligence (UAI)*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2014.

Thomas Kipf and Max Welling. Variational Graph Auto-Encoders. *Neural Information Processing Systems (NIPS) Workshop on Bayesian Deep Learning*, 2016.

Thomas Kipf and Max Welling. Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Neural Information Processing Systems (NIPS)*, 2012.

Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A Flexible Generative Framework for Graph-based Semi-supervised Learning. *Neural Information Processing Systems (NeurIPS)*, 2019.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3:127–163, 2000.

Geoffrey J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley, 1992.

Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially Regularized Graph Autoencoder for Graph Embedding. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. *Neural Information Processing Systems (NIPS) Autodiff Workshop*, 2017.

Alessandro Perina and Nebojsa Jojic. Image analysis by counting on a grid. *Computer Vision and Pattern Recognition (CVPR)*, 2011.

Alessandro Perina and Nebojsa Jojic. Spring lattice counting grids: Scene recognition using deformable positional constraints. *European Conference on Computer Vision (ECCV)*, 2012.

Alessandro Perina, Nebojsa Jojic, Manuele Bicego, and Andrzej Turski. Documents as multiple overlapping windows into a grid of counts. *Neural Information Processing Systems (NIPS)*, 2013.

Alessandro Perina, Sadegh Mohammadi, Nebojsa Jojic, and Vittorio Murino. Summarization and Classification of Wearable Camera Streams by Learning the Distributions Over Deep Features of Out-Of-Sample Image Sequences. *International Conference on Computer Vision (ICCV)*, 2017.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. *Knowledge Discovery and Data Mining (KDD)*, 2014.

Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov Neural Networks. *International Conference on Machine Learning (ICML)*, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. *International Conference on Learning Representations (ICLR)*, 2020.

Matthias Seeger. Learning with Labeled and Unlabeled Data. 2002. URL `https://infoscience.epfl.ch/record/161327/files/review.pdf`.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. *Neural Information Processing Systems (NeurIPS) Relational Representation Learning Workshop*, 2018.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23:447–478, 2011.

Yu Tian, Long Zhao, Xi Peng, and Dimitris Metaxas. Rethinking Kernel Methods for Node Representation Learning on Graphs. *Neural Information Processing Systems (NeurIPS)*, 2019.

Phi Vu Tran. Learning to Make Predictions on Graphs with Autoencoders. *Data Science and Advanced Analytics (DSAA)*, 2018a.

Phi Vu Tran. Multi-Task Graph Autoencoders. *Neural Information Processing Systems (NIPS) Workshop on Relational Representation Learning*, 2018b.

Unknown. PubMed. URL `https://linqs.soe.ucsc.edu/data`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. *Neural Information Processing Systems (NIPS)*, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2018.

Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. *arXiv 1909.11715*, 2019.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying Graph Convolutional Networks. *International Conference on Machine Learning (ICML)*, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. *International Conference on Machine Learning (ICML)*, 2016.

Dongmian Zou and Gilad Lerman. Encoding Robust Representation for Graph Generation. *International Joint Conference on Neural Networks (IJCNN)*, 2019.