Deep Representation Learning on Labeled Graphs

Shuangfei Fan

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Bert Huang, Chair
Naren Ramakrishnan
Chandan K. Reddy
A. Lynn Abbott
Jennifer Neville

December 06, 2019
Blacksburg, Virginia

Keywords: Machine learning, graph representation learning, collective classification, graph generation

Deep Representation Learning on Labeled Graphs

Shuangfei Fan

(ABSTRACT)

We introduce recurrent collective classification (RCC), a variant of ICA analogous to recurrent neural network prediction. RCC accommodates any differentiable local classifier and relational feature functions. We provide gradient-based strategies for optimizing over model parameters to more directly minimize the loss function. In our experiments, this direct loss minimization translates to improved accuracy and robustness on real network data. We demonstrate the robustness of RCC in settings where local classification is very noisy, settings that are particularly challenging for ICA. As a new way to train generative models, *generative adversarial networks* (GANs) have achieved considerable success in image generation, and this framework has also recently been applied to data with graph structures. We identify the drawbacks of existing deep frameworks for generating graphs, and we propose labeled-graph generative adversarial networks (LGGAN) to train deep generative models for graph-structured data with node labels. We test the approach on various types of graph datasets, such as collections of citation networks and protein graphs. Experiment results show that our model can generate diverse labeled graphs that match the structural characteristics of the training data and outperforms all baselines in terms of quality, generality, and scalability. To further evaluate the quality of the generated graphs, we apply it to a downstream task for graph classification, and the results show that LGGAN can better capture the important aspects of the graph structure.

Deep Representation Learning on Labeled Graphs

Shuangfei Fan

(GENERAL AUDIENCE ABSTRACT)

Graphs are one of the most important and powerful data structures for conveying the complex and correlated information among data points. In this research, we aim to provide more robust and accurate models for some graph specific tasks, such as collective classification and graph generation, by designing deep learning models to learn better task-specific representations for graphs. First, we studied the collective classification problem in graphs and proposed recurrent collective classification, a variant of the iterative classification algorithm that is more robust to situations where predictions are noisy or inaccurate. Then we studied the problem of graph generation using deep generative models. We first proposed a deep generative model using the GAN framework that generates labeled graphs. Then in order to support more applications and also get more control over the generated graphs, we extended the problem of graph generation to conditional graph generation which can then be applied to various applications for modeling graph evolution and transformation.

Dedication

To the most important people in my life: my twin sister

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor, Professor Bert Huang, for his invaluable guidance during the work and also for his unwavering support and encouragement. I feel so fortunate and appreciative for being one of the many students that could work with you. You are truly a great inspiration for me in research and thanks for being such a great mentor that guides me towards the right path for the past five years. Also I am hornored to have Professor Naren Ramakrishnan, Professor Chandan K. Reddy, Professor A. Lynn Abbott, and Professor Jennifer Neville as my committee members. Great thanks to them for their valuable time and guidance along the way. Moreover, I would like to thank my colleagues in our lab. Thank you so much for you support and company, I cherish those spacious time we spent together, it has been wonderful working with you. Most importantly, I would like to thank my twin sister and my parents, they helped me get through those tough times with infinite amounts of love, support and encouragement, they made me who I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will discuss the importance of studying graph data and their applications in various domain; then we will discuss several important tasks for understanding graph data, such as collective classification, representation learning, and graph generation. We will also review existing methods for these problems. Finally we will briefly describe our proposed frameworks for graph representation learning based on deep learning algorithms for different applications.

1.1 Graphs

Graphs form an universal language for modeling complex data. They are powerful for representing various kinds of relationships among data and they allow a shared vocabulary between fields, such as computer science, math, economics and biology. Examples of graphs being used to represent various data in different domains are in Figure 1.1.

In graph theory, a graph is a structure consisting of a set of nodes in which some pairs are related. Edges are the tool we use to represent the relationships between these nodes. There are many types of graphs for representing various kinds of data. Examples are shown in Figure 1.2. For example, the nodes in the graph may be labeled. If the nodes represent the users in an online social network, the label of each node could be the nationality, the gender, or the location of that user. Similarly, the edges may also be labeled. For example,

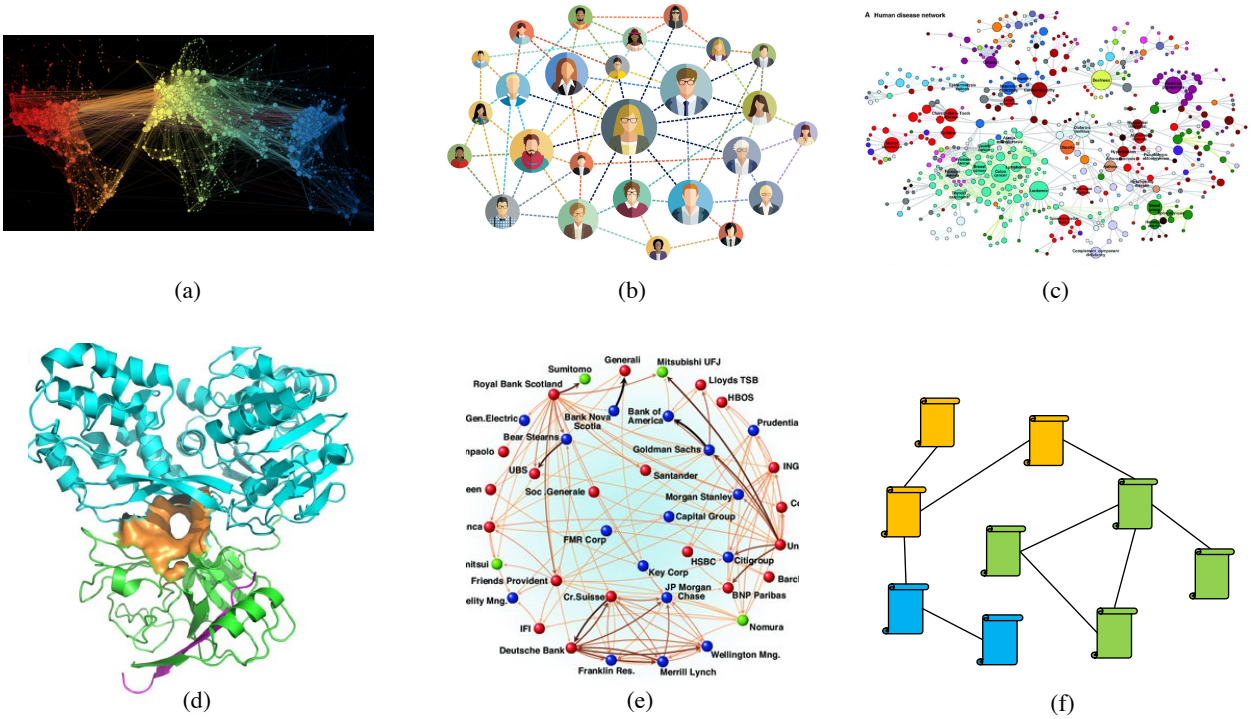
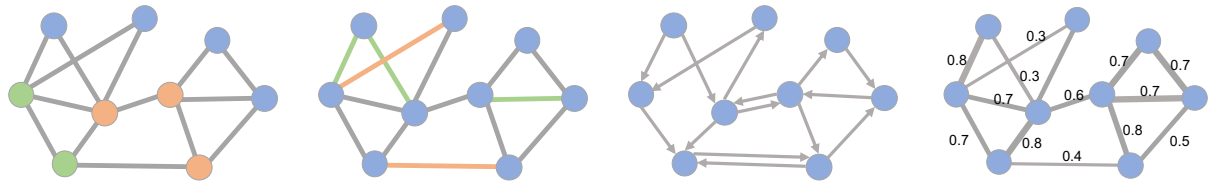


Figure 1.1: Examples of various data represented as networks: (a) Airport traffic network [11]; (b) Social networks [12]; (c) Medical human disease networks [52]; (d) Protein-protein interaction networks [69]; (e) Economic networks [75];(f) Citation networks.

in a small social network of college students where the edges represent the relationship between users, relationships might be labeled as of friend-friend, but also following-follower or professor-students and so on.

Moreover, the edges may also be directed or undirected. For example, if the nodes represent a user on an online social network that allows the following action, if user A follows user B, there will be an edge from user A to user B. However this action or edge is directed and thus this type of graph is called directed. In contrast, if the nodes represent people at a conference and there will be an edge if two people shake hands, in this case, the edge is undirected because this is an action with equal communication from each side. If person A shakes hands with person B, it means the relationship also happens in the other direction.



(a) (b) (c) (d)

Figure 1.2: Examples of various types of graphs. (a) Node-labeled graphs; (b) Edge-labeled graphs; (c) Edge-directed graphs; (d) Edge-weighted graphs.

Thus this type of graph is called undirected.

Also sometimes the strength of the relation or interaction between nodes varies instead of equally spread, therefore, the edges could also be weighted to represent these differences. For example, in the airport traffic network, the nodes represent the airport and the edges between them represent the volume of the traffic between two airports. In this case, we should use a weighted graph to convey the information in the data more accurately.

1.2 Classical Machine Learning Tasks with Graphs

Machine learning on graphs is an important goal with various applications in many areas, such as drug design for discovering new medicine and recommendation in social networks. The main challenge is to find a good method to represent the graph structure so that it can be easily adopted by machine learning models on different downstream tasks. There are many classical machine learning tasks with graphs, such as node classification, link prediction, collective classification, and graph classification.

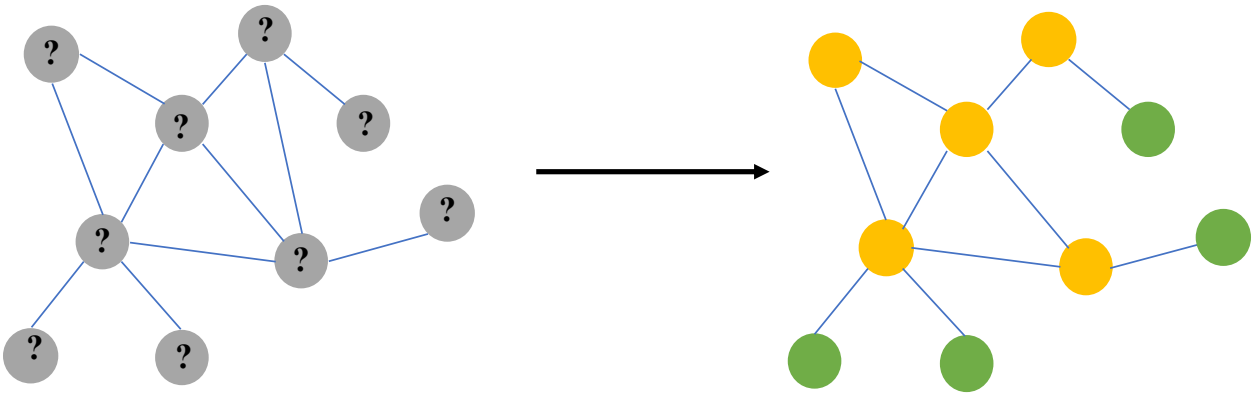


Figure 1.3: Node classification on graphs

1.2.1 Node Classification

Node classification is one of the most important problems for graph data. The definition of the problem is that, given some data represented as graphs, how to provide a strategy for labeling the nodes based on some predefined labelling rules. An example is shown in Figure 1.3. This problem has various applications, such as classifying papers in a citation network into categories of different research areas, or categorizing users on social networks based on their age or gender.

Similarly to the node classification problem, collective classification is also trying to solve one of the traditional machine learning tasks: classification. However, different from node classification, collective classification is the task that classify the nodes in the graph simultaneously. The assumption is that the relationships among nodes contains important information that will help improve the performance of node classification.

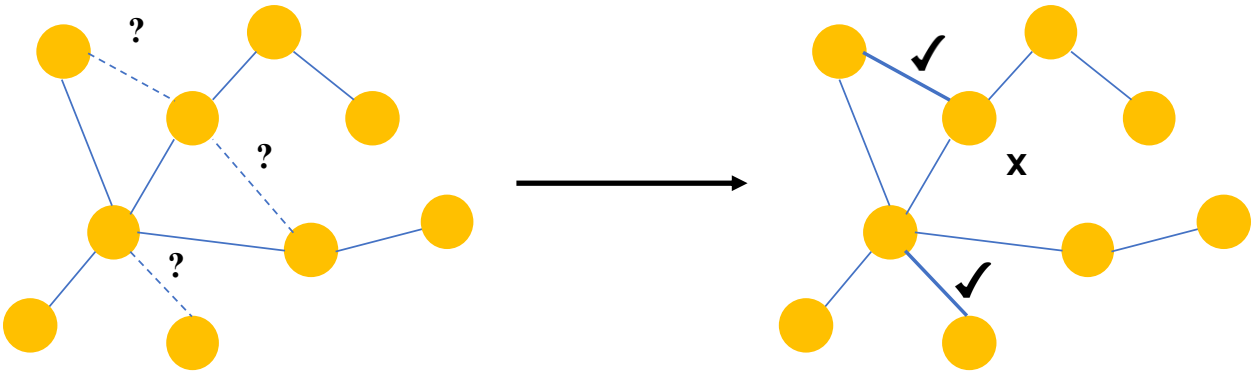


Figure 1.4: Link prediction on graphs

1.2.2 Link Prediction

Link prediction is another traditional yet important task for graph analysis. The goal is to identify whether a link exists between two nodes. An example is shown in Figure 1.4. Link prediction can be treated as a binary classification problem over all possible links. It is difficult due to the sparsity that comes from the imbalance between the positive and negative examples.

1.2.3 Graph Classification

Graph classification is the task to predict the label of the whole graph. Figure 1.5 shows an example for classifying graphs based on the structures. In supervised classification, the models are constructed to learn from the training data based on either on similarity or features, such as graph kernels and graph boosting [71]. Graph classification has various applications, especially for chemical informatics, where millions of graphs are available as training data such as molecular graphs for chemical compounds.

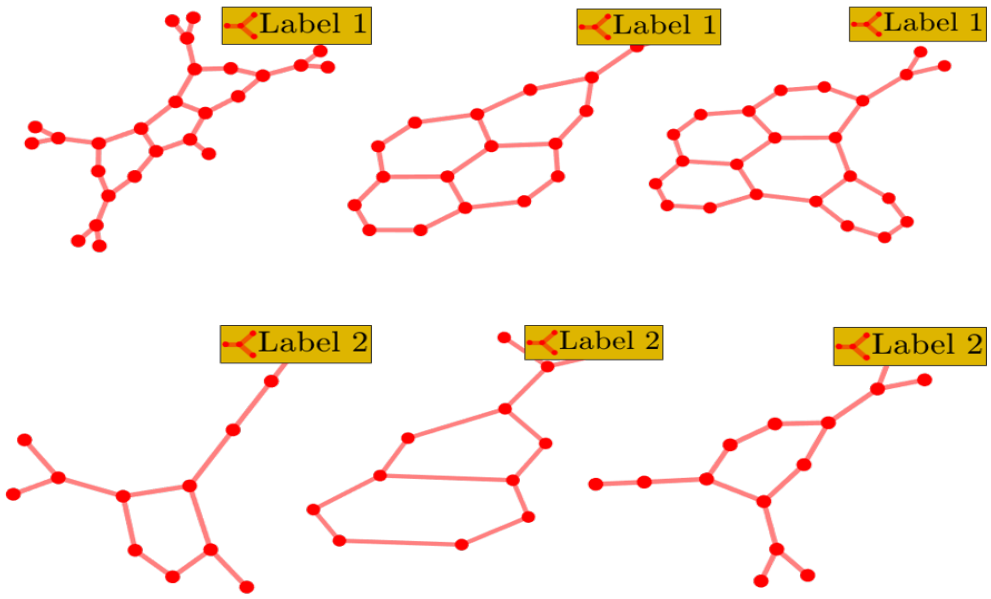


Figure 1.5: Graph classification on different structure [85]

1.3 New Tasks with Graphs

Recently, with the development of deep learning, many new tasks emerged in computer vision and natural language processing area, such as generating realistic data based on the distribution learned from the original data. This task has vast applications for both images and text, and it has rapidly become one of the most popular topics researchers are exploring. New achievements in those two areas also brought new ideas for graph researchers, and people are exploring how to build generative models for this complex data structure.

1.3.1 Graph Generation

Graph generation is one of the core topics in graph analysis. Many methods have been proposed to solve this problem, which can be traced back to at least 1959 when Erdös and Rényi [18] first introduced the Erdös-Rényi (E-R) model for generating random graphs. The model is based on the assumption that each pair of nodes are connected with a fixed predefined probability. However, this assumption is not realistic in most real world networks. To mimic the structure of real graphs, Albert and Barabási [2] proposed the preferential attachment model by further customizing the probability of each possible edge to be conditioned on current degrees of nodes. Separately, Airoldi et al. [1] proposed the mixed-membership stochastic block model (MMSB) to generate graphs that have a fixed number of communities based on a probability matrix to determine the possibility of a node pair from two communities been connected. This model allows to learn distributions from observed data, which make it more useful random graphs based on basic assumptions.

Other classical graph generative models include exponential random graph models (ERGMs) [70, 79], the stochastic block model (SBM) [29], the Watts-Strogatz model [87] and the Kronecker graph model [47], and many more. These older approaches have limited ability to learn about graph distributions from collections of graphs.

Recent advances in deep learning produced more powerful models for graph data, such as graph neural networks (GNNs) [10, 43, 48], which can work directly on graphs and leverage the structural information in relational systems and therefore make more accurate predictions. In other domains, Sutskever et al. [81] proposed a sequence-to-sequence (seq2seq) model based on the *attention mechanism* that has gained enormous traction due to its superior performance on modeling sequential data while overcoming the limitation of distance in input and output when modeling the dependencies. Based on the success of the attention

mechanism, Vaswani et al. [83] proposed the transformer architecture, which further improved the seq2seq model by enabling parallel computation with a self-attention mechanism to learn global text representation. To address the problem of long-range relation modeling in graph neural networks, Veličković et al. [84] proposed graph attention networks (GAT), which incorporate the self-attention mechanism so that the model can learn the importance of the connected nodes instead of treating all equally.

Recently Goodfellow et al. [31] described generative adversarial networks (GANs), which have been widely explored in computer vision and natural language processing [93, 94] for generating realistic images and text, as well as performing tasks such as style transfer. The success of this general GAN framework has proven it to be a powerful tool for learning the distributions of complex data. Motivated by the power of GANs, researchers started to use them for generating graphs too. Bojchevski et al. [7] proposed NetGAN, which uses the GAN framework to generate random walks on graphs. De Cao and Kipf [14] proposed MolGAN, which generates molecular graphs using the combination of a GAN framework and a reinforcement learning objective.

There are multiple applications for graph generation. It can be useful for simulation studies, especially when access to labeled graph data is limited by access or privacy concerns. We can also use these models to generate datasets, or augment existing datasets, to do graph-based analyses such as communication segmentation, node classification, anomaly detection, and link prediction. Figure 1.6 shows an example for designing new drugs using generative models for graphs.

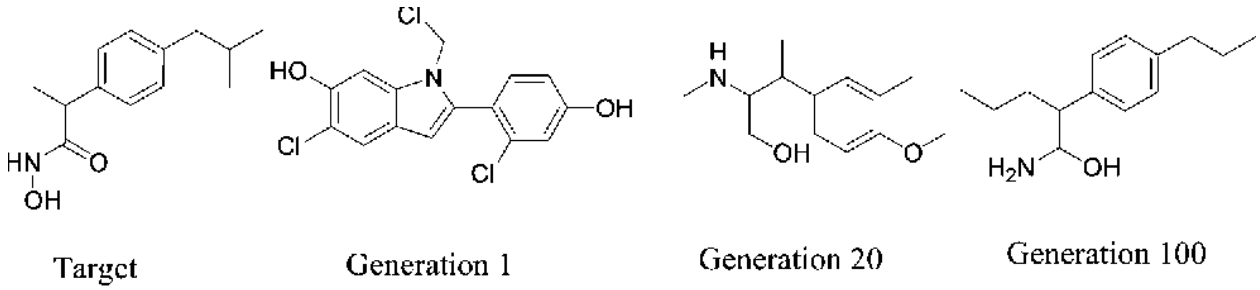


Figure 1.6: Examples of the application for graph generation for drug design [65]

1.3.2 Graph Transformation and Evolution

Graph generation has become a very popular topic in graph analysis as the success of the generative models achieved in computer vision and natural language processing tasks. However, compared to them, the applications of generated graphs are relatively constrained and most of them are used for drug design.

Moreover, models for unconditional graph generation limit their control over the generating procedure and restricts the applicability to many problems where graphs transform from one state to another and evolve in dynamic network settings. Some graphs are more complex than static ones, such as dynamic graphs and growing networks. They have another dimension of information along the time sequence, and the graph will evolve or grow as time passes, such as in social networks and web networks. These graphs can better represent the information in this rapidly changing world, but it also bring challenges for researchers to learn and model that. Therefore, we expanded the graph generation problem to a graph growing problem that tries to generate or expand graphs based on current seen sub-structures and predict new nodes or edges features and connectivity. Figure 1.7 shows an example for the evolution of the subgraphs in Yahoo networks.

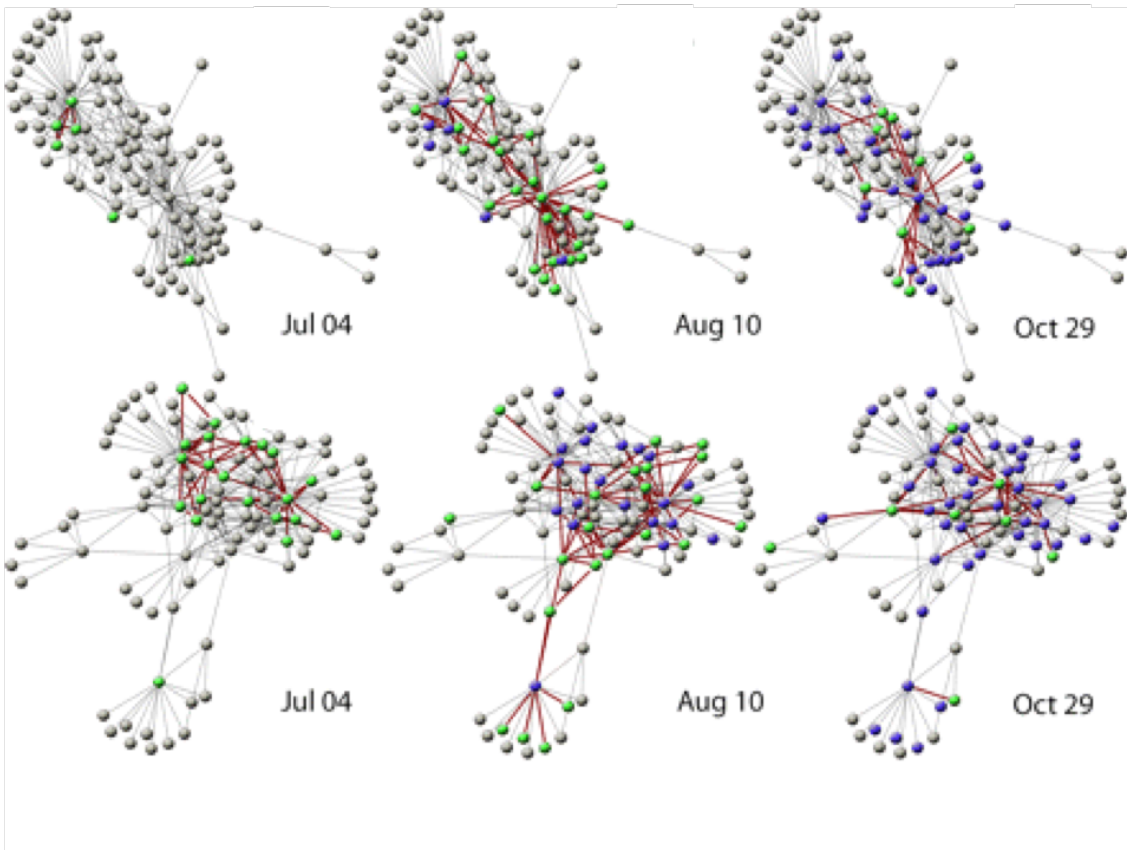


Figure 1.7: Examples for the application of conditional graph generation for modeling graph evolution on subgraphs of Yahoo networks [4]

1.4 Main Contribution

Our work mainly focused on two important topics in graph analysis. One is about collective classification, which tries to infer the node label based on the combined information from the node itself and also related nodes. The other one is about graph generation, which tries to generate realistic graphs that can be used in many applications such as drug design and traffic forecasting.

1.4.1 Collective Classification

First, we studied collective classification in networks, where the canonical method for solving this problem is the iterative classification algorithm (ICA) with relational information incorporated [62]. We showed that ICA introduces a bias that is inconsistent with the actual prediction algorithm during training procedure. So we proposed a variant of ICA, recurrent collective classification (RCC), as a procedure analogous to recurrent neural network prediction, which enables gradient-based strategies for optimizing over model parameters and is showed to be compatible with many classifiers just as ICA [19, 21, 22].

The problem RCC aims to solve is the discrepancy between training ICA with relational features based on the true labels and the fact that at prediction time, ICA uses estimated labels. To illustrate this discrepancy, we run experiments that consider situations where local classification becomes more and more difficult. We experimented with four data sets, two bibliographic data sets, one social network data set and one image dataset. For each experiment, we evaluate on four different approaches for node classification: (1) local prediction using only the local features; (2) ICA trained using the true labels; (3) Gibbs Sampling (GS) [55] trained using the true labels; and (4) RCC trained using back-propagation. They are all trained using a multi-class logistic regression loss function. ICA and GS are trained with the concatenated relational features computed from the true labels in addition to local features as input. RCC is trained using the training labels only in computing the loss, but never as input to the classifier in any form.

For the citation and network data, we generate versions of the data set where different fractions of the features are removed, in the range $[0.0, 0.9]$. For image data, we add varying amounts of salt-and-pepper noise. In effect, the experiments are run on versions of the data sets where prediction is harder, and more relevantly, where the assumption that the

predicted labels are exactly the true labels becomes more and more incorrect. The results for all data sets suggest that RCC is more robust to weak local signal than ICA, GS, and local classifier. Also we demonstrate that, by training RCC with back-propagation, we more directly optimize the training loss of collective classification, which translates to improved accuracy and robustness on real network data. This robustness enables effective collective classification in settings where local classification is very noisy, settings that previously were particularly challenging for ICA and variants.

1.4.2 Graph Generation

After working on the previous project and gaining a better understanding about representation learning for graphs, we explored the direction that treats representation learning as one way to get a better understanding about the graphs, and then we can use this information to generate fake realistic graphs that can capture the topology of real networks. There are multiple applications for graph generation. It can be useful for simulation studies, especially when access to labeled graph data is limited by access or privacy concerns. We can also use these models to generate datasets, or augment existing datasets, to do graph-based analyses such as communication segmentation, node classification, anomaly detection, and link prediction. To address this problem, building on the framework of Generative Adversarial Network [31], we proposed labeled-graph generative adversarial networks (LGGAN), which is a deep generative model trained using a GAN framework to generate graph-structured data with node labels [23, 24, 26]. LGGAN can be used to generate various kinds of graph-structured data, such as citation graphs, knowledge graphs, and protein graphs. Specifically, the generator in an LGGAN generates an adjacency matrix as well as labels for the nodes, and its discriminator uses a graph convolution network [43] with residual connections to identify real graphs using adaptive, structure-aware higher-level graph features.

In experiments, we evaluated our model on various datasets with different graph types—such as ego networks and proteins—and with different sizes. Our experiments demonstrate that LGGAN effectively learns distributions of different graph structures and that it can scale up to generate large graphs without losing much quality. Moreover, we also tested the approach with different discriminative models as well as different GAN frameworks on various types of graph datasets, such as collections of citation networks and protein graphs. Experiment results show that our model can generate diverse labeled graphs that match the structural characteristics of the training data and outperforms all baselines in terms of quality, generality, and scalability.

1.4.3 Conditioned Graph Generation

We explored the problem of graph generation and proposed a deep generative model for generating labeled graphs which can be easily extended to heterogeneous graphs, such as protein interaction graphs and scene graphs in computer vision problems. However, most approaches including our model (LGGAN) are unconditional generative models, which limits their control over the generating procedure and restricts the applicability to real world where graphs transform from one state to another and evolve in dynamic network settings. Some graphs are more complex than static ones, such as dynamic graphs and growing networks. They have another dimension of information along the time sequence, and the graph will evolve or grow as time passes, such as in social networks and web networks. These graphs can better represent the information in this rapidly changing world, but it also bring challenges for researchers to learn and model that.

Therefore, we expanded the graph generation problem to a graph growing problem that tries to generate or expand graphs based on current seen sub-structures and predict new

nodes or edges features and connectivity. To do this, we introduced an attention-based graph evolution model (AGE). AGE is a model for conditional graph generation based on the attention mechanism [81] and the transformer framework [83] that allows consideration of global information with parallel computation across all graph nodes. AGE adopts the encoder-decoder structure, where the encoder tries to learn the representation of conditioned graphs using a self-attention mechanism, and the decoder tries to generate the representation of the target graphs using the correlation with the conditioned graphs and also with itself. The decoder can thus capture both global and local information. Specifically, the encoder of AGE takes in a source graph G_s represented by its initial representations, which is the concatenation of the adjacency matrix and the feature matrix (we can leave out the feature matrix if it's not given) and maps it to a high-level embedding. The decoder of AGE is composed of two attention structures, the source-to-target attention which encodes the information from the source graph and the self-attention which incorporates the information of the previously generated nodes. The decoder is an autoregressive model that generates an output sequence of nodes one at a time, where each step is also conditioned on the previously generated nodes. We perform experiments on datasets in various applications. We first evaluated AGE on the task of graph evolution in space. We tested this problem setting on citation networks where the problem is to predict the expansion of ego networks with farther-hop neighbors. The results indicated that AGE is a strong conditional graph generator in both its ability to mimic graph distributions and match the target graphs. Moreover, since many networks are not static and they change and evolve over time, with the addition and deletion of new nodes and edges, we also tested AGE on the problem of graph evolution in time. The problem is to model the evolution of graphs from time t to time t'. The evaluation results show that for this problem, AGE still maintains a high-level of performance compared to all the other generative models in terms of both the realism of generated graphs and the similarity of the generated graphs to the target ones. By considering both the addition and

deletion of nodes and edges, we proved that AGE is able to learn not only graph evolution through growth, but also the more complex setting of volatile evolution.

To conclude, the results show that AGE can not only generate extremely realistic graphs, but also has the strong ability to model the evolution of graphs as a powerful conditioned graph generative model. This graph-conditioned generation framework greatly enriches the potential applications for graph generation on modeling graph evolution in various domains, such as for predicting information propagation in social networks, disease control for healthcare, and traffic prediction in road networks.

1.5 Overview and Outline

The remaining part of this document is organized as follows:

- Chapter 2 describes the new method we propose for training iterative collective classifiers for labeling nodes in network data. We introduce recurrent collective classification (RCC), a variant of ICA analogous to recurrent neural network prediction. RCC accommodates any differentiable local classifier and relational feature functions. We provide gradient-based strategies for optimizing over model parameters to more directly minimize the loss function. This is the main work of our paper published in Journal of Knowledge and Information Systems (KAIS).

- Chapter 3 describes the labeled-graph generative adversarial networks (LGGAN), which is a deep generative models we propose for graph-structured data with node labels based on the generative adversarial networks (GAN) framework.

- Chapter 4 describes the graph generative model we propose, which is a conditional graph generator based on the neural attention mechanism that can not only model

graph evolution in both space and time, but can also model the transformation between graphs from one state to another.

- Chapter 5 summarizes our research, explains their applications, and discusses future prospects.

Chapter 2

Literature Review

2.1 Collective Classification

Node classification is one of the fundamental tasks in analysis of network data [28, 50]. *Collective classification* addresses this task by making joint classifications of connected nodes [44, 64, 82]. Gibbs sampling (GS) is another approach for collective classification using the iterative classification framework [55, 76], that introduces randomization into the iterative classification. ICA and GS have been shown repeatedly to be effective frameworks for collective classification [39, 53, 55, 63, 76]. One of the more natural motivations for collective classification comes from the study of social networks, where the phenomenon of *homophily*—the tendency of individuals to interact with other similar individuals—has been an important concept [6, 56]. The types of dependencies that can exist in networks are not limited to assortative relationships, and methods such as ICA enable models to encode both assortative and non-assortative phenomena.

Collective classification has been studied in both inductive and transductive settings. In inductive settings, a collective classifier is typically trained on a fully labeled training network and evaluated at test time on a completely new network with no known labels. In transductive settings, the classifier is both trained and tested on partially labeled networks, or possibly in the same network with a different set of labels known during each phase [90]. In this thesis, we focus on the inductive setting.

Through the interpretation of non-terminal classifications as latent variables, ICA can be related to deep learning methods. Since the same classifier is used to predict each latent layer, ICA is most related to recurrent neural networks (RNNs), which feature a similar feedback loop in which an output of a neural network is used as its input in a subsequent iteration. RNNs were introduced decades ago [3, 13], but they have recently become prominent because of their effectiveness at modeling sequences, such as those occurring in natural language processing, e.g., [32, 33, 57, 80]. A now standard method for gradient optimization of RNN parameters is known as back-propagation through time [37, 38, 89], which unrolls recurrent networks and computes gradients for the parameters separately before combining them into a single update.

After the first version of our manuscript was published [20], two other groups have independently and concurrently pursued very similar directions, applying deep learning to collective classification [60, 68]. Moore and Neville [60] proposed to use RNNs for node-based relational classification tasks. They transform each node and its set of neighbors into an unordered sequence and use an LSTM-based RNN to predict the class label as the output of that sequence. Pham et al. [68] proposed another deep learning model for collective classification. Their approach is fully end-to-end, with a neural network that computes hidden units connected in the same structure as the input network, but the hidden units do not output class probabilities. Instead, they are abstract representations of learned local and relational features. These concurrent studies represent different ideas for bringing the power of neural networks to collective classification.

2.2 Graph Generation

Generative graph models were pioneered by Erdös and Rényi [18], who introduced random graphs where each possible edge appears with a fixed independent probability. More realistic models followed, such as the preferential attachment model of [2], which grows graphs by adding nodes and connecting them to existing nodes with probability proportional to their current degrees. Goldenberg et al. [29] proposed the stochastic block model (SBM), and Airoldi et al. [1] proposed the mixed-membership stochastic block model (MMSB). The SBM is a more complex version of the Erdös-Rényi (E-R) model that can generate graphs with multiple communities. In SBMs, instead of assuming that each pair of nodes has identical probability to connect, they predefine the number of communities in the generated graph and have a probability matrix of connections among different types of nodes. Compared to the E-R model, SBMs are more useful since they can learn more nuanced distributions of graphs from data. However, SBMs are still limited in that they can only generate graphs with this kind of community structure.

With the recent development of deep learning, some works have proposed deep models to represent the distribution of graphs. Li et al. [49] proposed DeepGMG, which introduced a framework based on graph neural networks. They generate graphs by expansion, adding new structures at each step. Li et al. [49] proposed GraphRNN, which decomposes the graph generation into generating node and edge sequences from a hierarchical recurrent neural network. Simultaneously, researchers have also been developing other implicit yet powerful methods for generating graphs, especially based on the success of generative adversarial networks [31]. For example, Bojchevski et al. [7] proposed NetGAN, which uses the GAN framework to generate random walks on graphs from which structure can be inferred, and De Cao and Kipf [14] proposed MolGAN to generate molecular graphs using the combination of the GAN framework and reinforcement learning.

However, these recently proposed deep models are either limited to generating small graphs with 40 or fewer nodes [49], or to generating specific types of graphs such as molecular graphs [14, 91] (with no straightforward generalization to other domains due to specialized tools to calculate molecule-specific loss). Most broadly, most of these recently proposed methods cannot generate labeled graphs.

Conditioned Graph Generation Recently, researchers also have proposed to use deep models to learn distributions for graph generation. These methods can be divided into two categories. Some of these generate graphs in a sequential manner by adding new nodes and corresponding edges step-by-step to represent the generative process of the graph. Examples of these are the DeepGMG model [49] and the GraphRNN model You et al. [92]; Some methods are based on the variational auto-encoder framework, which learn to approximate the density functions [73, 78]; While the others model the graph generation procedure in a non-sequential way. Among them, many models are based on generative adversarial networks (GANs) [31], which learn data distributions without explicitly defining a density function [7, 14, 25]. However, these deep models are either limited to generating small graphs with less than thirty nodes [49, 78], or to generating specific types of graphs such as molecular graphs [14, 91]. More importantly, the overarching drawback of all these deep generative models is that they are unconditioned, which severely limits their applicability to real-world tasks.

To further strengthen the power of graph generative models, Fan and Huang [25] proposed a conditioned model, which can generate graphs conditioned on discrete labels based on the conditional GAN frameworks [58, 67]. However, it can not be applied to circumstances where we want to generate graphs conditioned on another graph, which would be an interesting topic for graph evolution and graph transformation. Also Jin et al. [40] proposed a model with

the junction tree encoder-decoder framework for graph to graph transformation, however they only target on molecular optimization problem.

Chapter 3

Recurrent Collective Classification

3.1 Introduction

Data science tasks often require reasoning about networks of connected entities, such as social and information networks. In classification tasks, the connections among network nodes can have important effects on node-labeling patterns, so models that perform classification in networks should consider network structure to fully represent the underlying phenomena. For example, when classifying individuals by their personality traits in a social network, a common pattern is that individuals will communicate with like-mined individuals, suggesting that predicted labels should also tend to be uniform among connected nodes. Collective classification methods aim to make predictions based on this insight. In this paper, we introduce a collective classification framework that will enable an algorithm to more directly optimize the performance of trained collective classifiers.

Iterative classification is a framework that enables a variety of supervised learning methods to incorporate information from networks. The base machine learning method can be any standard classifier that labels examples based on input features. The iterative classification algorithm (ICA) operates by using previous predictions about neighboring nodes as inputs to the current predictor. This pipeline creates a feedback loop that allows models to pass information through the network and capture the effect of structure on classification. In spite of the feedback loop being the most important aspect of ICA, existing approaches train

models in a manner that ignores the feedback-loop structure. In this paper, we introduce *recurrent collective classification* (RCC), which corrects this discrepancy between the learning and prediction algorithms, incorporating principles used in deep learning and recurrent neural networks into the training process.

Existing learning algorithms for iterative classification resort to an approximation based on the unrealistic assumption that the predicted labels of neighbors are their true classes [51, 63]. This assumption is overly optimistic. If it were true, iteration would be unnecessary. Because the assumption is overly optimistic, it causes the learned models to cascade and amplify errors when the assumption is broken in early stages of prediction. In contrast, ICA uses predicted neighbor labels as feedback for each subsequent prediction, which means that if the model was trained expecting these predicted labels to be perfect, it will not be robust to situations where predictions are noisy or inaccurate. In this paper, we correct this faulty assumption and develop an approach that trains models for iterative classification by treating the intermediate predictions as latent variables. We compute gradients to the classification loss function using back-propagation through iterative classification.

To compute gradients for ICA, we break down the ICA process into differentiable (or sub-differentiable) operations. In many cases, the base classifier is differentiable with respect to its parameters. For example, if it is a logistic regression, it has a well-studied gradient. ICA also computes dynamic relational features using the predictions of network neighbors. These relational features are also functions through which gradients can be propagated. Finally, because the same base-classifier parameters should be used at all iterations of ICA, we can use methods for recurrent neural networks such as back-propagation through time (BPTT) [89] to compute the combined gradient. In contrast with existing strategies for training ICA, the resulting training optimization more closely mimics the actual procedure that ICA uses for prediction.

Algorithm 1 The Iterative Classification Algorithm

Input: Adj. matrix $\mathbf{A}$, node features $\mathbf{X}$, num. of iterations T, classifier f, and relational feature function g.
Initialize labels Y {e.g., uniform probability}
for t from 1 to T **do**
  $\mathbf{R} \leftarrow g(Y; \mathbf{A})$ {Compute relational features}.
  $Y \leftarrow f(\mathbf{X}, \mathbf{R}; \mathbf{\Theta})$ {Compute new predictions}.
end for
return labels Y

The RCC framework accommodates a variety of base classifiers and relational feature functions. The only restriction is that they must be differentiable. Therefore, RCC is nearly as general as ICA, and its prediction procedure is practically identical to ICA. The key difference is that the view of the algorithm as nested, differentiable functions enables a training procedure that is better aligned with RCC and ICA prediction.

We evaluate RCC on data where collective classification has previously been shown to be helpful. We demonstrate that RCC trains classifiers that are robust to situations where local predictions are inaccurate.

3.2 Iterative Classification

In this section, we review the *iterative classification algorithm* (ICA) and the standard method for training its parameters. ICA provides a framework for node classification in networks. ICA is given a decorated graph $G = \{V, E, \mathbf{X}\}$, where $V = \{v_1, \ldots, v_n\}$, E contains pairs of linked nodes $(v_i, v_j) \in E$, and each node is associated with a respective feature vector $\mathbf{x}_i \in \mathbb{R}^d \equiv \mathcal{X}$, where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Using these inputs, ICA outputs a set of predictions Y classifying each of the nodes in V into a discrete label space $\mathcal{Y}$. Throughout this paper, we will consider the multi-class setting, in which the labels can take one of k class-label values. ICA makes the label predictions by iteratively classifying nodes by their

local features $\mathbf{x}_i$ and their dynamic *relational features* $\mathbf{r}_i$, which is in a common space $\mathbf{r}_i \in \mathcal{R}$. In other words, the classifier is a function f that maps $\mathcal{X} \times \mathcal{R}$ to $\mathcal{Y}$, parameterized by a parameter variable $\mathbf{\Theta}$.

ICA first initializes labels as $Y^{(0)}$, and it then iterates the steps

$$\begin{aligned} \mathbf{R}^{(t-1)} &\leftarrow g(Y^{(t-1)}; \mathbf{A}) \\ Y^{(t)} &\leftarrow f(\mathbf{X}, \mathbf{R}^{(t-1)}; \mathbf{\Theta}) \end{aligned} \tag{3.1}$$

from iterations $t = 1$ to $t = T$.

The dynamic relational features $\mathbf{R}$ where $\mathbf{R} = \{\mathbf{r}_1, \ldots, \mathbf{r}_n\}$ are computed based on the current estimated labels of each node. They enable the classifier to reason about patterns of labels among connected nodes. E.g., a common relational feature is the average prediction of neighboring nodes.

Using any such aggregation statistic creates a relational feature vector of dimensionality k, where each entry is the occurrence rate of its corresponding label in the node's neighbors. I.e.,, the relational features are computed by a feature function g that maps $\mathcal{Y}^n$ to $\mathcal{R}^n$.

Since the dynamic relational features are computed based on the output of the classifier, the entire process is iterated: (1) all nodes are labeled by the classifier f using the current dynamic relational features, then (2) the dynamic relational features are computed with g based on the new labels. These two phases are repeated either until the predictions converge and do not change between iterations or until a cutoff point.

The ICA framework is general in that any classifier f can be used and any form of a dynamic relational feature function g can be used. In practice, researchers have used naive Bayes, logistic regression, support vector machines as classifiers, and they have used averages, sums, and presence as relational features [39, 53, 63, 76]. The generality of the framework is a key

benefit of ICA—one that we aim to preserve in our proposed framework. In contrast to specially designed methods for collective classification, the modularity of ICA makes it a flexible meta-algorithm.

In many settings, we consider real-valued local and relational features, so each node is described by a feature vector created by concatenating its local features with its relational features $[\mathbf{x}_i,\ \mathbf{r}_i]$. In this case, it is convenient to notate the classification function in matrix form. Let $\mathbf{X}$ denote the feature matrix for the graph, such that the ith row of $\mathbf{X}$, i.e., $\mathbf{x}_i$ is the (transposed) feature vector for node v_i. Similarly, let $\mathbf{R}$ denote the relational feature matrix, such that the ith row of $\mathbf{R}$ is the dynamic relational feature vector of node v_i. For convenience, we consider the case where one type of dynamic relational feature is used, meaning the dimensionality of $\mathbf{R}$ is n by k (though it is easy to extend both ICA and RCC to multiple relational features).

The training procedure for ICA trains the classifiers by computing relational features using the training labels. Given a training set consisting of a set of nodes V, edges E, node features X, and ground-truth labels $\hat{Y}$, one generates relational features R using the true training labels, creating fully instantiated, fully labeled inputs for the classifier. The model parameters $\mathbf{\Theta}$ are learned by setting $\hat{\mathbf{R}} = g(\hat{Y})$, then using $\mathbf{X}$ and $\hat{\mathbf{R}}$ as the input to a supervised training scheme appropriate for fitting the parameters $\mathbf{\Theta}$ of f.

This training methodology is only correct in the situation where we expect a perfect-classification fixed-point. In such a fixed point, the relational features are computed using the true labels, and the classifiers perform perfectly, exactly predicting the true labels of the nodes; the relational features are computed, using the perfectly predicted labels, to be exactly the same features that are computed using the true training labels; since the relational features are computed using the exactly predicted true labels, the output is the same as in the first step, and the algorithm converges perfectly to the true labels. Using the matrix form

of the relational feature computation above, this absurd fixed point would be characterized as $Y = f\left(\mathbf{X},\ g(\hat{Y}); \mathbf{\Theta}\right)$.

Unfortunately, such a fixed point is unrealistic. In practice, the classifications can be inaccurate or made with low confidence due to a lack of reliable local information. Thus, training the model to expect that the neighbor labels be perfect creates an overconfidence that can lead to cascading errors.

3.3 Recurrent Collective Classification

We propose the recurrent collective classification (RCC) framework to address the previously mentioned deficiencies in ICA training. The RCC framework computes derivatives for the stages of collective classification and provides a general scheme for how to compute the gradient of the loss function with respect to the classifier parameters. Algorithm 2 summarizes RCC gradient computation. This gradient computation enables the training of collective classifiers in a manner that more directly mimics how they will be applied. At test time, a collective classifier is often given only local features of nodes connected in a network. Thus, any relational features are derived from *predicted* neighbor labels. A classifier considering these neighbor labels should therefore consider common patterns of misclassification of neighbor labels. In contrast to the ICA training procedure, which invites the classifier to become overly reliant on the verity of the neighbor labels.

Prediction in RCC is analogous to ICA. We initialize $\mathbf{P}^{(0)}$ (e.g., setting it to all zeros, or to random values), then iterate the steps $\mathbf{R}^{(t-1)} \leftarrow g(\mathbf{P}^{(t-1)}; \mathbf{A})$, and $\mathbf{P}^{(t)} \leftarrow f(\mathbf{X}, \mathbf{R}^{(t-1)}; \mathbf{\Theta})$ from iterations $t = 1$ to $t = T$. This recursive procedure can be interpreted as a recurrent neural network, as illustrated in Figure 3.1.

Algorithm 2 RCC Gradient Computation

Input: Graph $G = \{V, E\}$, number of iterations T, classifier f, and relational feature function g.
Initialize parameter $\mathbf{\Theta}$ set $\mathbf{P}^{(0)} = 0$
for t from 1 to T **do**
  $\mathbf{R}^{(t)} \leftarrow g(\mathbf{P}^{(t-1)}; \mathbf{A})$ {Relational features}.
  $\mathbf{P}^{(t)} \leftarrow f(\mathbf{X}, \mathbf{R}^{(\mathbf{t})}; \mathbf{\Theta})$ {New predictions}
end for
$\mathbf{\Delta}^{(\mathbf{T})} \leftarrow \mathbf{L}'(\mathbf{P}^{(\mathbf{T})})$ {Loss gradient for output}
for t from T to 2 **do**
  for j from 1 to N **do**
    $\delta_j^{(t-1)} \leftarrow \sum_{i:(i,j)\in E} \delta_i^{(t)} \cdot f'\left(\mathbf{r}_i^{(t-1)}\right) \cdot g_i'\left(\mathbf{p}_j^{(t)}\right)$
  end for
end for
$\nabla(\mathbf{\Theta}) \leftarrow \sum_{t=1}^{T} \mathbf{\Delta}^{(t)} f'(\mathbf{\Theta})$

RCC training requires that the local classifier function f is equipped with efficiently computable gradients with respect to the parameters $\mathbf{\Theta}$ and the relational features $\mathbf{R}$, and that the relational feature function g is equipped with an efficiently computable gradient with respect to the current predictions $\mathbf{P}$. Since these functions map matrices to matrices, their gradients are computed via matrix calculus. However, as we discuss in Section 3.3.1, they often have sparse structure that makes their computation efficient. As a shorthand, with some abuse of notation, we refer to these gradients using the following definitions:

$$\begin{aligned} f'_{(t)}(\mathbf{\Theta}) &:= \partial f(\mathbf{X}, \mathbf{R}^{(t-1)}; \mathbf{\Theta}) / \partial \mathbf{\Theta} \\ f'(\mathbf{R}^{(t-1)}) &:= \partial f(\mathbf{X}, \mathbf{R}^{(t-1)}; \mathbf{\Theta}) / \partial \mathbf{R}^{(t-1)} \\ g'(\mathbf{P}^{(t)}) &:= \partial g(\mathbf{P}^{(t)}; \mathbf{A}) / \partial \mathbf{P}^{(t)}. \end{aligned} \tag{3.2}$$

At iteration T, we evaluate a loss function L on the final prediction $\mathbf{P}^{(T)}$. RCC admits any differentiable loss function. For example, for multinomial distribution, we can use Softmax classifier, which uses the cross-entropy loss. The cross-entropy loss is to minimize the cross-entropy between the estimated distribution $\mathbf{q}$ and the "true" distribution $\mathbf{p}$ as the loss

function. It is defined as:

$$L = H(p, q) = -\sum_{x} p(x) \log q(x) \tag{3.3}$$

Denote the gradient of the loss with respect to the predictions $\mathbf{P}^{(t)}$, i.e., $\mathbf{\Delta}^{(t)} := \frac{\partial L}{\partial \mathbf{P}^{(t)}}$. Given the gradients for each iteration, the gradient of the loss with respect to the parameters $\mathbf{\Theta}$ is $\frac{\partial L}{\partial \mathbf{\Theta}} = \sum_{t=1}^{T} \mathbf{\Delta}^{(t)} f'(\mathbf{\Theta})$.

Based on these formulas, RCC is able to use any gradient-based optimization over the parameter space, so long as it is able to compute the loss gradients $\mathbf{\Delta}$. These gradients can be computed using chain rule and dynamic programming, as in back-propagation. The general formula is:

$$\begin{aligned} \mathbf{\Delta}^{(t-1)} = \frac{\partial L}{\partial \mathbf{P}^{(t-1)}} &= \left(\frac{\partial L}{\partial \mathbf{P}^{(t)}}\right)\left(\frac{\partial \mathbf{P}^{(t)}}{\partial \mathbf{P}^{(t-1)}}\right) \\ &= \mathbf{\Delta}^{(t)} \left(\frac{\partial \mathbf{P}^{(t)}}{\partial \mathbf{R}^{(t-1)}}\right)\left(\frac{\partial \mathbf{R}^{(t-1)}}{\partial \mathbf{P}^{(t-1)}}\right). \end{aligned} \tag{3.4}$$

Though these matrix gradients can be large for arbitrary matrix functions, collective classification has a particular structure that enables efficient computation.

3.3.1 Derivative Structure

Key aspects in understanding the efficient computability of these gradients are the independence relationships among the input and output variables. These independence relationships are universal to any local classifier and to any relational feature.

Because the local classifier operates on each node independently, in each iteration, there is no dependence between the classification of any node and the relational features of any

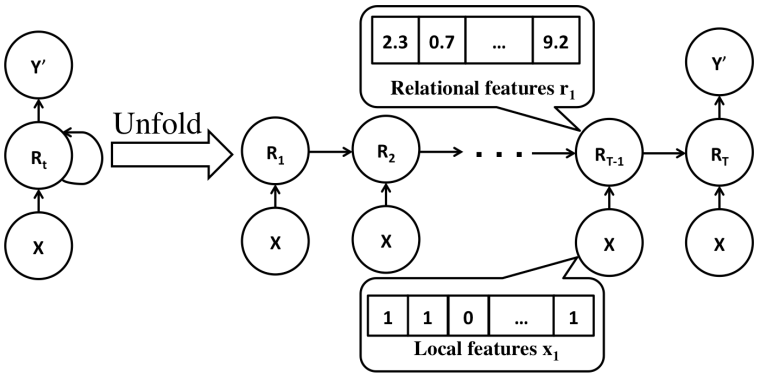


Figure 3.1: Structure of RCC/ICA prediction. The recurrent form (left) unrolls into a form (right) that explicitly considers each iteration as a separate operation.

other node. The matrix derivative of the classifier function f is block diagonal, only having nonzero derivative between each node's relational-feature row $\mathbf{r}_i$ and its output-prediction row $\mathbf{p}_i$.

Because relational features are completely determined by the neighbors of each particular node, there is only dependence between the predictions of any node j and the relational feature of node i if they are neighbors in the graph. The matrix derivative of the relational feature function g therefore has a block structure in the shape of the sparse adjacency matrix.

Considering both of these sparse block structures, we can define a new shorthand for the nonzero elements of the matrix derivatives:

$$\begin{aligned} f'\left(\mathbf{r}_i^{(t-1)}\right) &:= \partial f(\mathbf{x}, \mathbf{r}_i^{(t-1)}; \mathbf{\Theta}) / \partial \mathbf{r}_i^{(t-1)} \equiv \partial \mathbf{p}_i^{(t)} / \partial \mathbf{r}_i^{(t-1)} \\ g_i'\left(\mathbf{p}_j^{(t)}\right) &:= \partial \left[g(\mathbf{p}_j^{(t)}; \mathbf{A}) \right]_i / \partial \mathbf{p}_j^{(t)} \equiv \partial \mathbf{r}_i^{(t)} / \partial \mathbf{p}_j^{(t)}. \end{aligned} \tag{3.5}$$

Figure 3.2 illustrates the sparse structure of the full matrix gradients using these definitions. The matrix chain rule then simplifies to

$$\frac{\partial \mathbf{P}^{(t)}}{\partial \mathbf{R}^{(t-1)}} = \begin{bmatrix} \left[f'(\mathbf{r}_1^{(t-1)})\right] & [\mathbf{0}] & \cdots & [\mathbf{0}] \\ [\mathbf{0}] & \left[f'(\mathbf{r}_2^{(t-1)})\right] & \cdots & [\mathbf{0}] \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{0}] & [\mathbf{0}] & \cdots & \left[f'(\mathbf{r}_N^{(t-1)})\right] \end{bmatrix} \quad \frac{\partial \mathbf{R}^{(t-1)}}{\partial \mathbf{P}^{(t-1)}} = \begin{bmatrix} [\mathbf{0}] & \left[g_1'(\mathbf{p}_2^{(t-1)})\right] & [\mathbf{0}] & \cdots \\ \left[g_2'(\mathbf{p}_1^{(t-1)})\right] & [\mathbf{0}] & \left[g_2'(\mathbf{p}_3^{(t-1)})\right] & \cdots \\ [\mathbf{0}] & \left[g_3'(\mathbf{p}_2^{(t-1)})\right] & [\mathbf{0}] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Figure 3.2: Matrix gradients for RCC. The left gradient is block-diagonal, since predictions depend only on each node's relational features. The right gradient has block sparsity matching the sparsity of adjacency matrix $\mathbf{A}$.

$$\begin{aligned} \boldsymbol{\delta}_j^{(t-1)} &= \sum_{i:(i,j)\in E} \left(\frac{\partial L}{\partial \mathbf{p}_i^t}\right)\left(\frac{\partial \mathbf{p}_i^t}{\partial \mathbf{r}_i^{(t-1)}}\right)\left(\frac{\partial \mathbf{r}_i^{(t-1)}}{\partial \mathbf{p}_j^{(t-1)}}\right) \\ \boldsymbol{\delta}_j^{(t-1)} &= \sum_{i:(i,j)\in E} \boldsymbol{\delta}_i^{(t)} \cdot f'\left(\mathbf{r}_i^{(t-1)}\right) \cdot g_i'\left(\mathbf{p}_j^{(t)}\right). \end{aligned} \tag{3.6}$$

where the loss derivatives $\boldsymbol{\delta}_i^{(t)}$ are gradient vectors and the classifier and feature function derivatives, f' and g' are small Jacobian matrices. This structure is significantly sparser and more efficient to compute than the full matrix calculus in Equation (3.4).

3.3.2 Example Local Classifiers

One benefit of ICA is that it can use any local classifier and relational feature. Similarly, given Section 3.3.1, we can also easily use many local classifiers f and relational features g into RCC. In this section, we give some examples of these configurations.

First, we consider linear classifiers with activation functions where the linear product prediction scores are squashed by different activation functions. A common activation function is the **logistic sigmoid function**:

$$f(\mathbf{x_i}, \mathbf{r_i}^{(\mathbf{t-1})}; \boldsymbol{\Theta}) = \frac{\mathbf{1}}{\mathbf{1}+\exp\left(-\left[\mathbf{x_i}, \quad \mathbf{r_i^{(t-1)}}\right]\cdot\boldsymbol{\Theta}\right)} = \mathbf{p_i}. \tag{3.7}$$

where we write $[\mathbf{x}_i, \mathbf{r}_i]$ to indicate the horizontal concatenation of the row vectors $\mathbf{x}_i$ and $\mathbf{r}_i$. Let $\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\Theta}_{\mathbf{x}} \\ \boldsymbol{\Theta}_{\mathbf{r}} \end{bmatrix}$, separating the parameters for the relational features to submatrix $\boldsymbol{\Theta}_{\mathbf{r}}$. The derivative with respect to $\mathbf{r}_i^{(t-1)}$ is the Jacobian matrix $f'(\mathbf{r_i}^{(t-1)}) = \text{diag}(\mathbf{p}_i^{(t-1)}(1-\mathbf{p}_i^{(t)}))\boldsymbol{\Theta}_{\mathbf{r}}^{\top}$. The gradient with respect to the parameters is

$$f'(\boldsymbol{\Theta}) = -\,\text{diag}(\mathbf{p}_i^{(t-1)}(1-\mathbf{p}_i^{(t)})) \cdot \left[\mathbf{X}, \mathbf{R}^{(t-1)}\right] \ . \tag{3.8}$$

Another activation function is the **tempered softmax function**, i.e., a generalization of both the normalized multi-class logistic and the max function:

$$f(\mathbf{x_i}, \mathbf{r_i}^{(\mathbf{t-1})}; \boldsymbol{\Theta}) = \frac{\exp\left(\left(\left[\mathbf{x_i}, \quad \mathbf{r_i}^{(\mathbf{t-1})}\right] \cdot \boldsymbol{\Theta}\right)/\tau\right)}{\mathbf{1}^{\top} \exp\left(\left(\left[\mathbf{x_i}, \quad \mathbf{r_i}^{(\mathbf{t-1})}\right] \cdot \boldsymbol{\Theta}\right)/\tau\right)} = \mathbf{p_i}. \tag{3.9}$$

where τ is the temperature parameter. As τ approaches 0, the limit of the softmax is an argmax indicator vector indicating the maximal entry, and at $\tau = 1$, this is exactly the multi-class logistic class probability.

The Jacobian of the softmax with respect to the relational features is $f'(\mathbf{r_i}^{(t-1)}) = \frac{1}{\tau}(\text{diag}(\mathbf{p_i}) - \mathbf{p_i}\mathbf{p_i}^{\top})\boldsymbol{\Theta}_{\mathbf{r}}^{\top}$. And the gradient with respect to the parameters is

$$f'(\boldsymbol{\Theta}) = -\frac{1}{\tau}(\text{diag}(\mathbf{p_i}) - \mathbf{p_i}\mathbf{p_i}^{\top}) \cdot \left[\mathbf{X}, \mathbf{R}^{(t-1)}\right] \ . \tag{3.10}$$

The local classifier used in RCC can also seamlessly incorporate non-linear classifiers such as multi-layer perceptrons and neural networks that have well-understood derivatives.

3.3.3 Example Relational Features

There are many aggregation operators can be used to define the relational features. Commonly used features include the **sum** of probabilities for each class, the **proportion** of each class in the neighborhood, **mode**, which is the class label with the highest probability among neighbors and **exists**, which is an indicator for each class label [76]. The choice of which one to use depends on the application. Here we discuss three of them and provide their derivatives.

The sum aggregation function g can be written as $\mathbf{r_i} = g(\mathbf{P}^{(t)}; \mathbf{a}_i) = \mathbf{a}_i^\top \mathbf{P}^{(t)}$, where $\mathbf{a_i}$ is the adjacency vector of node i. The Jacobian of the sum feature is $g_i'(\mathbf{p}_j^{(t)}) = a_{ij}\mathbf{I}_k$, where $\mathbf{I}_k$ is the identity matrix.

The proportion operator is similar to sum, except that we scale the adjacency matrix $\mathbf{A}$ by normalizing each row vector. The operation can be summarized as $\hat{\mathbf{A}} = \mathbf{A} \oslash (\mathbf{A} \cdot \mathbf{1}_N)$, where $\mathbf{1}_N$ is the all-ones matrix of size N and N is the total number of nodes. The operator $\oslash$ performs element-wise division. The proportion operator uses $\hat{\mathbf{A}}$ for the relational feature function $\mathbf{r_i} = g(\mathbf{P}^{(t)}; \hat{\mathbf{a}}_i) = \hat{\mathbf{a}}_i^\top \mathbf{P}^{(t)}$, where $\hat{\mathbf{a}}_\mathbf{i}$ is the normalized adjacency vector of node i. The Jacobian for the proportion feature function is $g_i'(\mathbf{p}_j^{(t)}) = \hat{a}_{ij}\mathbf{I}_k$.

For the mode aggregation operator, we can use the tempered softmax function, providing a differentiable form $\mathbf{r_i} = g(\mathbf{P}^{(t)}; \mathbf{a}_i) = \frac{\exp(\mathbf{a}_i^\top \mathbf{P}^{(t)}/\tau)}{\mathbf{1}^\top \exp(\mathbf{a}_i^\top \mathbf{P}^{(t)}/\tau)}$ where small τ values closely mimic the true mode. The Jacobian of the mode aggregation relational feature is $g_i'(\mathbf{p}_j^{(t)}) = (1/\tau)a_{ij} \cdot (\text{diag}(\mathbf{r}_i) - \mathbf{r}_i\mathbf{r}_i^\top)$.

3.3.4 Computational Complexity

Prediction in RCC (and ICA) requires computing relational features and predicting T times. For most relational features, each node must perform $O(k)$ work per neighbor, amounting to $O(|E|k)$ total computation. Assuming a constant number of relational features, the linear local classifier performs $O(d+k)$ work. Thus each prediction iteration requires $O(|E|k+d)$ time, and the full prediction requires $O(T(|E|k+d))$ time.

Gradient-based optimization of the RCC objective function is a non-convex program, so the total number of iterations for learning is nontrivial to analyze in general. However, we can analyze the computational complexity of each gradient computation. The computation of the gradient requires three phases: the computation of the classifier and relational feature derivatives, the computation of the loss derivatives (back-propagation), and the computation of the parameter derivatives.

The cost of computing the classifier and relational feature derivatives depends on the chosen classifier and relational features. In the examples provided earlier, the cost is $O(k^2)$ for both.

The back-propagation formula in Section 3.3.1 computes a single row of the $\boldsymbol{\Delta}^{(t)}$ derivative, which has N total rows. The inner summation iterates over the neighbors of the current node, performing a vector-matrix product and a matrix-matrix product. The derivatives are Jacobians of size $O(k)$ by $O(k)$, so the total cost is $O(k^2)$. Thus, the combined cost is $O(T|E|k^2)$ for back-propagating the derivative through T iterations.

Finally, computation of the gradient for $\boldsymbol{\Theta}$ also depends on the local classifier. For the linear classifiers described earlier, the cost is $O(dk)$ per iteration, so $O(Tdk)$. Therefore, the entire gradient computation costs $O(T(dk+|E|k^2))$, which is comparable to the cost of prediction itself, since k is often a small quantity.

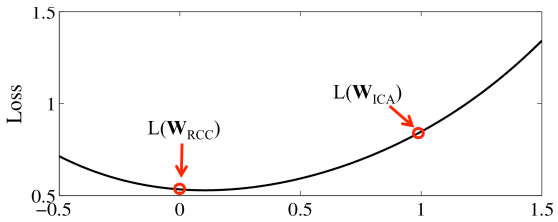


Figure 3.3: Cross-section of the training objective. Circles are solutions from RCC and ICA training. The RCC solution is at a local minimum while ICA's is not.

3.4 Experiments

In this section, we describe experiments that test whether the proposed training method for RCC is able to improve upon existing methods of training iterative classifiers. We explore scenarios where the local classifier produces inaccurate predictions, challenging the faulty assumption implied by training ICA and Gibbs sampling (GS) with relational features computed using the true labels. The results illustrate that our hypothesis is correct, identifying a variety of settings where RCC better optimizes the training objective and produces more accurate predictions on held-out data.

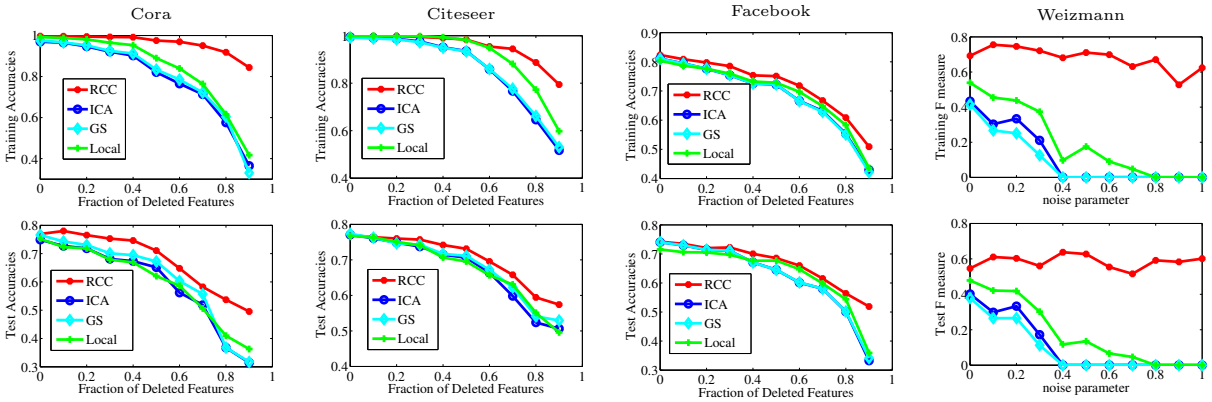


Figure 3.4: Performance of collective classifiers on the four tasks. Each curve plots the average training or testing accuracy or F-measure over the amount of noise (feature removal or salt-and-pepper). RCC dominates all methods on training accuracy, and it performs significantly better in testing than others when there is weak local signal.

3.4.1 Dataset

We experimented with four data sets, two bibliographic data sets, one social network data set and one image dataset. The Cora data set is a collection of 2,708 machine learning publications categorized into seven classes [76]. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3,703 unique words. The CiteSeer data set is a collection of 3,312 research publications crawled from the CiteSeer repository [76]. It consists of 3,312 scientific publications categorized into six classes. Each publication in the data set is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1,433 unique words. The social network data we use is the Facebook ego network data [54], which includes users' personal information, friend lists, and ego networks. We combine the ego networks and node features to form a single connected network, with 4,039 users' anonymized profile data and links between them. We used the feature "education type," as a label, aiming to predict how many degrees each user has, i.e., the four classes are whether their highest level of education is secondary, undergraduate, or graduate, or unknown. The image dataset we use is the Weizmann horse-image segmentation set [8]. We subsample 20 images of horses on various backgrounds. We use features described by Domke [16] for each pixel: We expand the RGB values of each pixel and the normalized vertical and horizontal positions into 64 features using sinusoidal expansion. The goal is to identify which pixels are part of a horse and which are background. We pose this task as a collective classification by constructing a grid graph over neighboring pixels. Specifically, we expand the feature vectors by multiplying our basic feature vector with all binary vectors $\mathbf{c}$ of the appropriate length and take the $sin(\mathbf{c} \times s)$ and $cos(\mathbf{c} \times s)$ as our final feature vectors, which have 64 features in total.

3.4.2 Training

For each experiment, we evaluate on four different approaches for node classification: (1) local prediction using only the local features; (2) ICA trained using the true labels; (3) GS trained using the true labels; and (4) RCC trained using back-propagation.They are all trained using a multi-class logistic regression loss function. ICA and GS are trained with the concatenated relational features computed from the true labels in addition to local features as input. RCC is trained using the training labels only in computing the loss, but never as input to the classifier in any form.

For each of the learning objectives, we optimize using the adagrad approach [17], in which gradients are rescaled based on the magnitude of previously seen gradients. For the gradient $\mathbf{g}_\tau$ at optimization iteration τ, one updates the variable $\mathbf{\Theta}$ with $\mathbf{\Theta}_\tau \leftarrow \mathbf{\Theta}_{\tau-1} - \eta \frac{\mathbf{g}_\tau}{\sqrt{\sum_{i=1}^{\tau} \mathbf{g}_i \odot \mathbf{g}_i}}$, where the gradient division by the historical magnitude is elementwise. Adagrad is one of many approaches that has been shown in practice to accelerate convergence of gradient-based optimization. Training is done with 2,000 iterations of adagrad and an initial learning rate of $\eta = 0.1$. We evaluate performance of each method using a range of regularization parameter settings from 1×10^{-3} to 1.

For the document and social network data, we perform snowball sampling to extract a random 1/5 of the nodes to hold out as a isolated test network. We train on the induced graph of the 4/5 remaining nodes, and measure predictive accuracy on both the training graph and testing graph. For the image data, we train on two random splits of 10 training and 10 testing images. The training accuracy should more closely reflect whether each method's training strategy effectively optimizes the model to fit the observed data, and the testing accuracy should additionally reflect how well the learned model generalizes. We compute both training and testing accuracy by feeding the learned model only the local

features and link structure, meaning that though ICA and GS is typically *trained* with the training labels as input, we do not provide the labels to them when *evaluating* its training accuracy. Our hypothesis is that by directly computing the gradient of the actual prediction procedure for collective classification, RCC will produce better training performance, which should translate to better testing performance.

3.4.3 Empirical Evaluation of Loss

To illustrate that RCC training optimizes the training loss function better than training with the true labels, we can compare the training loss associated with the soft-max output loss. We first train model weights using (1) RCC and (2) the ICA approach using the true labels. We then define a tradeoff function $l(\alpha) = L(\mathbf{\Theta}_{\text{RCC}} + \alpha(\mathbf{\Theta}_{\text{ICA}} - \mathbf{\Theta}_{\text{RCC}}))$, where L is the loss function, $\mathbf{\Theta}_{\text{RCC}}$ is the parameter matrix from the model trained by RCC, $\mathbf{\Theta}_{\text{ICA}}$ is the weight matrix trained by the ICA strategy. We apply the loss function to different weight matrices on the line that connects $\mathbf{\Theta}_{\text{RCC}}$ and $\mathbf{\Theta}_{\text{ICA}}$. This curve is a cross-section of the high-dimensional function that is the actual training loss. When $\alpha = 0$, the value is the loss of the RCC weights $L(\mathbf{\Theta}_{\text{RCC}})$, and (2) when $\alpha = 1$, the value is that of the ICA weights $L(\mathbf{\Theta}_{\text{ICA}})$. The results for Cora are shown in Figure 3.3. For all four data sets, the loss obtained by RCC training $L(\mathbf{\Theta}_{\text{RCC}})$ is near a local minimum and the loss obtained by ICA training is not. Unsurprisingly, these results suggest that the training procedure of RCC is a better strategy to optimize the loss function than using the true-label relational features, corroborating our proposed approach.

Table 3.1: Performance of RCC with different settings.

f	g	Cora	Citeseer	Facebook	Weizmann
logistic	mode	0.787	0.770	0.745	0.802
logistic	**prop.**	0.811	**0.773**	**0.748**	0.806
logistic	sum	0.806	0.765	0.748	**0.807**
softmax	mode	**0.818**	0.766	0.747	0.805
softmax	prop.	0.815	0.764	0.745	0.807
softmax	sum	0.817	0.766	0.743	0.795

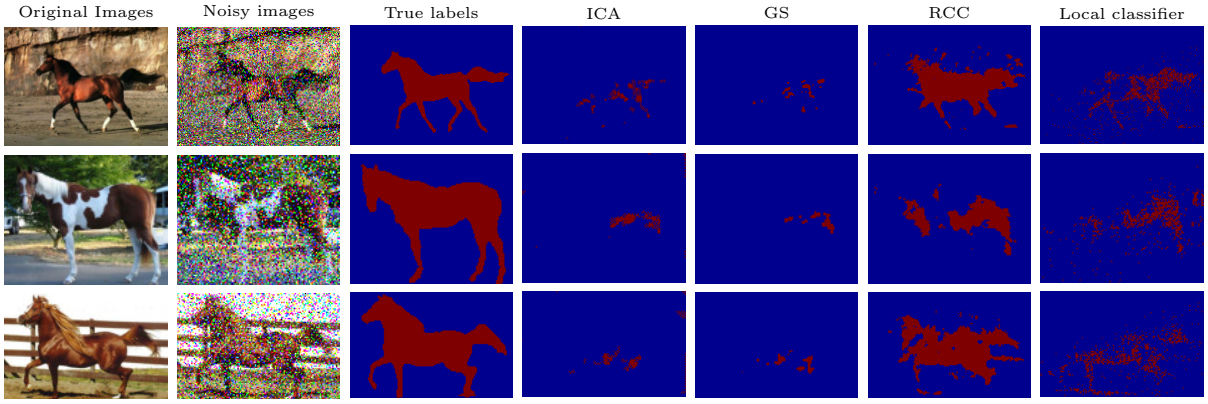


Figure 3.5: Example segmentations using RCC, ICA, GS and local classifiers on Weizmann data. Because of the noisy local classifiers, ICA and GS cascade errors into unreasonable segmentations, while RCC is more robust.

3.4.4 Comparing Relational Features and Classifiers

As we discussed before, RCC can be easily plugged in many local classifiers and it can also use many aggregation operators to include different relational features. To prove this, we run experiments with different classifier and aggregation operator for each of the four dataset: Cora, Citeseer, Facebook and Weizmann's dataset to compare the performance of different settings with RCC. For the local classifier, we choose from logistic sigmoid function and tempered softmax where we set τ to 0.5. For aggregation operators, we choose from mode, sum and proportion. For these experiments, we compare the average accuracy over 20 splits.

The results in Table 3.1 show that for all these four datasets, there are not much differences between these six experiments. So for the following experiments, we use logistic sigmoid function and proportion.

3.4.5 Comparison of Prediction Accuracy

The problem RCC aims to solve is the discrepancy between training ICA with relational features based on the true labels and the fact that at prediction time, ICA uses estimated labels. To illustrate this discrepancy, we run experiments that consider situations where local classification becomes more and more difficult. For the citation and network data, we generate versions of the data set where different fractions of the features are removed, in the range $[0.0, 0.9]$. For image data, we add varying amounts of salt-and-pepper noise. In effect, the experiments are run on versions of the data sets where prediction is harder, and more relevantly, where the assumption that the predicted labels are exactly the true labels becomes more and more incorrect. If RCC's training procedure is truly more robust to this scenario, we expect its improvement over ICA to become more pronounced as (local) prediction becomes more difficult. The results for these experiments are shown in Figure 3.4 where we plot the average accuracies or F-measures for the best-scoring regularization parameters over 20 splits of network data and 2 splits of image data. We compared RCC, ICA, GS, and the local classifier, which makes predictions only based on local features. The horizontal axis represents the amount of noise, and the vertical axis represents the training or testing accuracies achieved by these algorithms.

The results for all data sets suggest that RCC is more robust to weak local signal than ICA, GS, and local classifier. For the Cora and CiteSeer data, as the fraction of deleted features increases, the training accuracies of RCC stays stable until over 80% of local features have

been deleted. The training accuracies of ICA, GS, and the local classifier drop earlier and much faster. When 90% of the features are deleted, the training accuracies of ICA, GS, and the local classifier drop to 0.5, however the accuracies of RCC still remains around 0.9, showing that RCC is able to train models to fully utilize the relational structure. The RCC test accuracies also show better performance than the other three methods as the number of local features reduces. Especially when over 60% local features are deleted, the local predictors become less reliable which causes ICA's accuracy to significantly worsen, and RCC is able to withstand the lack of attribute information. The Facebook results follow similar trends, where the training and test accuracies are always better than the ICA and local classifier. The differences in the testing accuracy between RCC and ICA are statistically significant for all fractions of deleted features on the Cora tests, for 0.7 and 0.9 for the CiteSeer tests, and for all fractions 0.2 and higher on the Facebook tests.

One interesting effect not often reported in other research is the tendency for ICA trained using the true labels to produce predictors that perform *worse* than the local classifier, even on training data. This effect is exactly because of the discrepancy between the training regime and the actual prediction algorithm RCC aims to correct. For example, in all three of our data sets, there are settings, especially when the local classifier is noisy, that ICA has worse training accuracy than the local classifier. This effect is especially apparent in the image data, where the faulty assumption made by ICA causes it to consistently cascade local-classification errors, eventually leading ICA, GS, and the local predictor to predict that all pixels are background. RCC avoids this over-reliance on relational features, yet learns to incorporate relational features enough to improve upon the noisy local predictor. Figure 3.5 contains example images and the predicted segmentations using the various learning algorithms.

3.5 Conclusion

We presented recurrent collective classification, a variant of the iterative classification algorithm that uses differentiable operations, enabling back-propagation of error gradients to directly optimize the model parameters. The concept of collective classification has long been understood to be a principled approach to classifying nodes in networks, but in practice, it often suffers from making only small improvements, or not improving at all. One cause for this could be the faulty training procedure that we correct in our research. Our experiments demonstrate dramatic improvements in training accuracy, which translate to significant, but less dramatic improvements in testing performance. RCC is a key step toward fully realizing the power of collective classification. Thus, an important aspect to consider to further improve the effectiveness of collective classifiers is the generalization behavior of collective models. One future direction of research is exploring how a more direct training loss-minimization interacts with known generalization analyses, perhaps leading to further algorithm improvements. Another future direction we are exploring is how to apply similar approaches of direct loss minimization in transductive settings and how to expand the flexibility of the RCC framework to incorporate other variants of ICA.

Chapter 4

Labeled Graph Generative Adversarial Networks

4.1 Introduction

Labeled graphs are powerful complex data structures that can describe collections of related objects. Such collections could be atoms forming molecular graphs, users connecting on online social networks, and papers connected by citations. The connected objects, or nodes, may be of different types or classes, and the graphs themselves may belong to particular categories. Methods that reason about this flexible and rich representation can empower analyses of important, complex real-world phenomena. One key approach for reasoning about such graphs is to learn the probability distributions over graphs. In this paper, we introduce a method that learns generative models for labeled graphs in which the nodes and the graphs may have categorical labels.

A high-quality generative model should be able to synthesize labeled graphs that preserve global structural properties of realistic graphs. Such a tool could be valuable in various settings. One motivating example application is in situations where data owners wish to share graph data but must protect sensitive information. For example, online social network providers may want to enable the scientific community to study the structural aspects of their user networks, but revealing structure could allow reidentification or other privacy-invading

inferences [95]. A generative model that can create realistic graphs that do not represent real-world users could allow for this kind of study.

Recently Goodfellow et al. [31] described generative adversarial networks (GANs), which have been widely explored in computer vision and natural language processing [93, 94] for generating realistic images and text, as well as performing tasks such as style transfer. GANs are composed of two neural networks. The first is a generator network that learns to map from a latent space to the distribution of the target data, and the second is a discriminator network that tries to distinguish real data from candidates synthesized by the generator. Those two networks compete with each other during training and each improves based on feedback from the other. The success of this general GAN framework has proven it to be a powerful tool for learning the distributions of complex data.

Motivated by the power of GANs, researchers have used them for generating graphs too. Bojchevski et al. [7] proposed NetGAN, which uses the GAN framework to generate random walks on graphs. De Cao and Kipf [14] proposed MolGAN, which generates molecular graphs using the combination of a GAN framework and a reinforcement learning objective. However, there are many limitations of existing methods, such as the generality to graphs with different structures and scalability to different sized graphs. Furthermore, they are unable to generate graphs with node labels, a critical feature of some graph-structured data.

The rapid development of deep learning techniques has also led to advances in representation learning in graphs. Many works have been proposed to use deep learning structures to extract high-level features from nodes and their neighborhoods to include both node and structure information [22, 36, 43]. These methods have been shown to be useful for many applications, such as link prediction and collective classification.

Building on these advances, we propose *labeled graph generative adversarial network* (LGGAN),

a deep generative model trained using a GAN framework to generate graph-structured data with node labels. LGGAN can be used to generate various kinds of graph-structured data, such as citation graphs, knowledge graphs, and protein graphs. Specifically, the generator in an LGGAN generates an adjacency matrix as well as labels for the nodes, and its discriminator uses a graph convolution network [43] with residual connections to identify real graphs using adaptive, structure-aware higher-level graph features. Our approach is the first deep generative method that addresses the generation of labeled graph-structured data. In experiments, we demonstrate that our model can generate realistic graphs that preserve important properties from the training graphs. We evaluate our model on various datasets with different graph types—such as ego networks and proteins—and with different sizes. Our experiments demonstrate that LGGAN effectively learns distributions of different graph structures and that it can scale up to generate large graphs without losing much quality.

4.2 GAN framework

Since our proposed approach uses the generative adversarial network (GAN) framework, we briefly review some variations of GANs here. Our experiments test these variations for our task of labeled graph generation. The GAN framework was introduced by Goodfellow et al. [31], many variations have been proposed that proved to be powerful for generation tasks. Therefore, we adopt three popular variations and compare how well they perform for our task of labeled graph generation. We use the traditional, original GAN approach as the first approach. Beyond the traditional GAN framework, we use two other methods that include extra information of classification labels for the graphs themselves. The first follows the *conditional GAN* [59] framework, which feeds the graph label as an extra input

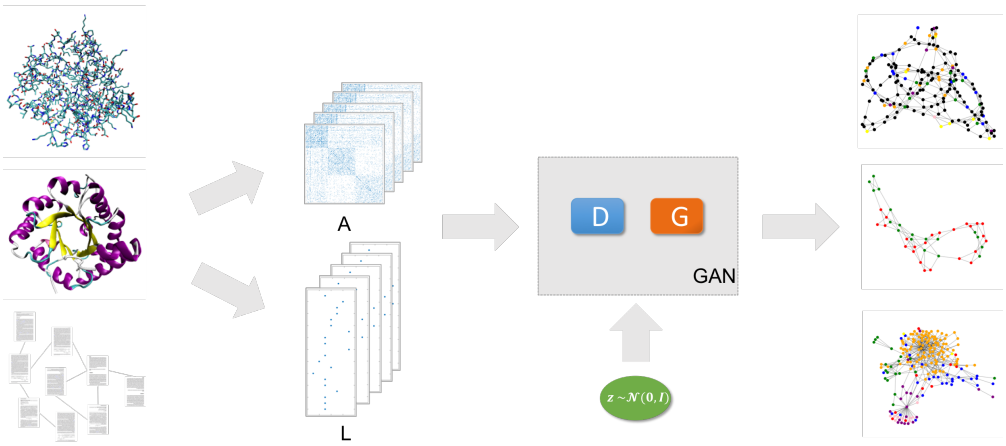


Figure 4.1: LGGAN: Adversarial training framework of our proposed model which shows how different graph-structured data such as protein and citation networks can be used to train for generating new graphs that can capture the topology of the real ones

to the generator in addition to the noise z. We can use this label to generate the graphs of different types. To improve on this, our last variation uses the *auxiliary conditional GAN* [67] framework, in which the discriminator not only distinguishes whether the graph is real or fake, but it also incorporates a classifier of the graph labels. The structure of all those three variations is illustrated in Figure 4.2.

4.2.1 Generative adversarial networks

Generative adversarial networks (GANs) [31] train implicit generative models by competitively training two neural networks. The first is the generative model G, which learns to map from a latent space to the distribution of the target data. The second network is the discriminative model D, which tries to separate the real data and the candidates predicted by the generator. These two networks compete with each other during training, via different objective functions. They adapt to improve itself based feedback from each other. The generator G and the discriminator D can be seen as two players in a minimax game where where the generator G tries to produce samples realistic enough to fool the discriminator, and

the discriminator D tries to differentiate the samples from the real data and the generator correctly. The famous objective for GAN training is

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D(G(\boldsymbol{z}))\right)\right] \tag{4.1}$$

4.2.2 Conditional GAN

Later after the orginal GAN was proposed, Mehdi et al.[59] introduce the conditional version of it, which can be constructed by simply modifying the GAN framework via feeding the class label $\boldsymbol{c}$ to both the generator and discriminator. By doing this, the model is then able to generate fake data that are conditioned on class labels instead of random data point from the distribution which means that this model gives us more control over the generated data. The objective function of Conditional GAN is similar to the original version, only need to add the condition of $\boldsymbol{c}$ to both generator and discriminator:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\left[\log D(\boldsymbol{x}|\boldsymbol{c})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D(G(\boldsymbol{z}|\boldsymbol{c}))\right)\right] \tag{4.2}$$

4.2.3 AC-GAN

Besides conditional GAN, Augustus et al.[66] proposed another variant of the GAN called AC-GAN that has the similar idea with conditional GAN. In the ACGAN, each generated sample has a pre-defined (randomly picked or based on some strategies) class label, $\boldsymbol{c} \sim p_{\boldsymbol{c}}$ in addition to the noise $\boldsymbol{z}$ as the input to the generator G. The generator G then uses both to generate samples $X_{fake} = G(\boldsymbol{c}, \boldsymbol{z})$. The discriminator will output probability distributions over both data and the class labels, $P(S|X), P(C|X) = D(X)$. The objective function of AC-GAN has two parts: the log-likelihood of the correct data, L_S, and the log-likelihood of

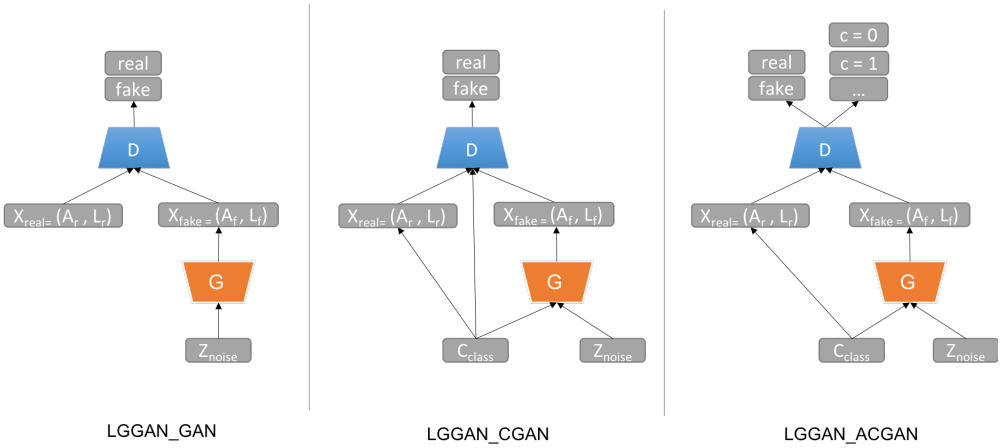


Figure 4.2: The adversarial training framework of LGGAN with different GAN structure.

the correct class, L_C.

$$L_S = \mathbb{E}\left[\log P(S = \text{real}|X_{\text{real}})\right] + \mathbb{E}[\log P(S = \text{fake}|X_{\text{fake}})] \tag{4.3}$$

$$L_C = \mathbb{E}[\log P(C = c|X_{real})] + \mathbb{E}[\log P(C = c|X_{fake})] \tag{4.4}$$

The objective function of the discriminator D is $L_S + L_C$ while the the objective function for the generator G is $L_C - L_S$. From the structure, we can see that the procedure of learning a mapping from z to the representation of AC-GANs is independent of class label. There are mainly 2 differences between AC-GAN and conditional GAN: Firstly, conditional GAN conditioned labels to both generator and discriminator, however in the structure of AC-GAN, it is only fed to the generator. Then the other one is that in AC-GAN, the discriminator not only tries to classify whether the sample comes from the real data or not, it also has a classifier that outputs the probability distribution over the class labels. It is proved that these modifications to the standard GAN formulation can produce excellent results and also help to stabilize the training procedure.

4.3 Model

In this section we introduce LGGAN — a deep generative model trained using the GAN framework to generate graph-structured data with node labels, such as citation graphs, knowledge graphs and protein graphs. Besides using the traditional GAN framework as we discussed in Section 4.2.1, in order to enable more control over the generated graphs, we propose two other models to include extra information of classification labels for the graphs themselves: one is to use the conditional-GAN [59] as we discussed in Section 4.2.2 which feed the graph label as an extra input in addition to the noise z to the generator. We can use this label to generate the graphs of different types. To improve on this, we also adopted the structure of AC-GAN [66] as we discussed in Section 4.2.3 which the discriminator not only distinguishes whether the graph is real or fake, but it also incorporates a classifier of the graph labels. By using this framework, we can then get extra information about how well our model is trained by computing the accuracy of the discriminator's classifier, serving as one of the evaluation metrics. The structure of all those three frameworks is shown in Figure 4.1.

4.3.1 Architecture

In this section, we provide details on the LGGAN architecture. As in the standard GAN framework, LGGAN consists of two main components: a generator G and a discriminator D. The generator G takes a sample from a prior distribution and generates a labeled graph g represented by an adjacency matrix A and a node label matrix L. The discriminator D then trains to distinguish samples from the dataset and samples from the generator. In LGGAN, both the generator and the discriminator are trained using CT-GAN [88], an improved version based on the improved Wasserstein GAN approach [35].

Generator LGGAN's generative model uses a multi-layer perceptron (MLP) to produce the graph. The generator G takes a random vector z sampled from a standard normal distribution and outputs two matrices: (1) $L \in R^{N \times C}$, which is a one-hot vector that defines the node labels; and (2) the adjacency matrix $A \in R^{N \times N}$, which defines the connections among nodes in graphs. The architecture uses a fixed *maximum* number of nodes N, but it is capable of generating structures of fewer nodes by dropping the nodes that are not connected to any of the other node in the generated graph. Since both the adjacency matrix A and the label matrix L are discrete and the categorical sampling procedure is non-differentiable, we directly use the continuous objects A and L during the training procedure. The original GAN structure uses the Jensen-Shannon (JS) divergence to measure the distance, which then cannot be used to generate discrete data. Therefore, we use variants of GANs that are based on Wasserstein distance, such as CT-GAN [88] and WGAN-GP [35], which has been shown to be applicable to discrete data such as text.

Discriminator The discriminator D takes a graph sample as input, represented by an adjacency matrix A and a node label matrix L, and it outputs a scalar value and a one-hot vector for the class label. For the discriminator, we use a graph convolutional network (GCN) [43], which is a powerful neural network architecture for representation learning with complex graph structures. GCNs propagate information along graph edges with graph convolution layers. GCNs are also permutation-invariant, which is important when analyzing graphs because they are usually considered unordered objects.

We add residual connections between hidden layers of the GCN to allow the model to fuse information from previous layers. We find that these residual connections circumvent the issues reported by Kipf and Welling [43] that limit their GCNs to only two or three layers. Allowing more depth is important because some graph types, such as proteins and molecules,

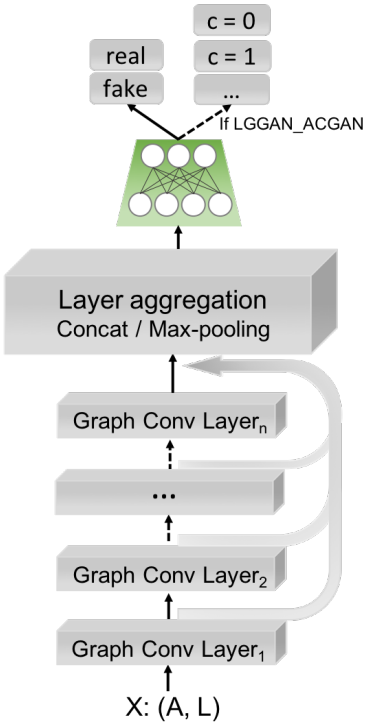


Figure 4.3: The structure of the LGGAN discriminator with residual connections.

have complex structure that require incorporation of information from nodes further in graph distance than are reachable with only three graph convolutions.

With residual connections, each layer of our GCN discriminator propagates with the following rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{4.5}$$

where $H(l) \in R^{N\times D}$ is the output matrix at the $l-1$th layer, I_N is the identity matrix, $\tilde{A} = A + I_N$ is the adjacency matrix of the graph g with self-connections added, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the diagonal degree matrix of graph g, $W^{(l)} \in R^{D\times F}$ is the trainable weight matrix at the lth layer, and $\sigma(\cdot)$ denotes an activation function (such as the sigmoid or ReLU [61]). Since we do not include node attributes, we set $H(0) = I_N$, where I_N is the identity matrix.

After n layers of propagation via graph convolutions, we aggregate the outputs from each

layer with an aggregation function agg, such as concatenation and max-pooling. We then concatenate the aggregated matrix with the node label matrix L and output Z_g as the final representation we learned for graph g:

$$Z_g = f(X, A, L) = \left[\text{agg}\left(H^{(1)}, \ldots, H^{(n)}\right); L\right] \tag{4.6}$$

The representation Z_g of the graph will further be processed by a linear layer to produce the outputs of the discriminator: the graph-level scalar probability of the input being real data and a classifier to predict the category that the graph belongs to with a one-hot vector c. We illustrate the structure of this discriminative model in Figure 4.3.

In Section 4.4.4, we evaluate the influence of the depth of GCN with and without residual connections and also with different aggregate functions to guide how to choose from different settings of LGGAN for the experiments.

4.3.2 Training

GANs [31] train via a min-max game with two players competing to improve themselves. In theory, the method converges when it reaches a Nash equilibrium, where the samples produced by the generator match the data distribution. However, this process is highly unstable and often results in problems such as mode collapse [30]. To deal with the most common problems in training GAN, such as mode collapse and unstable training, we use the CT-GAN [88] framework, which is one of the state-of-the-art approaches. CT-GAN adds a consistency term to the Wasserstein GANs (WGAN-GP) [72] that preserves Lipschitz continuity in the training procedure of WGAN-GPs. We also adopt several techniques such as feature matching and minibatch discrimination that were shown to encourage convergence and help avoid mode collapse [72]. Details are shown in the supplementary materials.

Wasserstein GAN Wasserstein GAN (WGAN) framework [5], as it prevents mode collapse and leads to more stable training. In this work, they introduced WGANs which minimize an approximation of the Wasserstein distance between the real distributions and the distribution of the generated samples. They proposed to use gradient clipping as an constraint on the 1-Lipschitz continuity to help WGAN to converge. In a later followup work, Gulrajani et al. [35] proposed a better method that uses a gradient penalty as an alternative soft constraint compared to the gradient clipping. Therefore, the loss function for the discriminator is modified to

$$L(\boldsymbol{x}^{(i)}, G_\theta(\boldsymbol{z}^{(i)}; \phi) = D_\phi(G_\theta(\boldsymbol{z}^{(i)})) - D_\phi(\boldsymbol{x}^{(i)}) + \alpha \left(\|\nabla_{\hat{\boldsymbol{x}}^{(i)}} D_\phi(\hat{\boldsymbol{x}}^{(i)})\| - 1\right)^2. \tag{4.7}$$

CT-GAN CT-GAN [88] is a recently proposed model based on WGAN-GP. It improves the WGAN-GP approach by adding a soft consistency term to enforce the Lipschitz constraint. We train our model based on it since CT-GAN has been shown to further stabilize the training procedure. The objective function for CT-GAN is

$$\begin{aligned} L(\boldsymbol{x}^{(i)}, G_\theta(\boldsymbol{z}^{(i)}; \phi) =& D_\phi(G_\theta(\boldsymbol{z}^{(i)})) - D_\phi(\boldsymbol{x}^{(i)}) + \lambda_1 \left(\|\nabla_{\hat{\boldsymbol{x}}^{(i)}} D_\phi(\hat{\boldsymbol{x}}^{(i)})\| - 1\right)^2 + \\ & \lambda_2 \mathbb{E}_{\boldsymbol{x}_1, \boldsymbol{x}_2} \left[\max\left(0, \frac{d(D(\boldsymbol{x}_1), d(D(\boldsymbol{x}_1))}{d(\boldsymbol{x}_1, \boldsymbol{x}_2)} - M\right)\right]. \end{aligned} \tag{4.8}$$

Feature matching To stabilize the training procedure of GAN, Salimans et al. [72] proposed another technique: feature matching to prevent the generator from overtraining on the current discriminator. It specifies a new objective function for the generator:

$$\|\mathbb{E}_{\boldsymbol{x} \sim p_{data}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})} f(G(\boldsymbol{z}))\|_2^2, \tag{4.9}$$

where $f(x)$ denote the activation of an intermediate layer of the discriminator. Instead of directly maximizing the output of the discriminator, the generator is trained to match the expected value of the features on an intermediate layer of the discriminator. The discriminator is thus trained to find the features that are most discriminative between the real samples from samples produced by the generative model.

4.3.3 Node Ordering

A common representation for graph structure uses adjacency matrices. However, using matrices to train a generative model introduces the issue of how to define the node ordering in the adjacency matrix. There are $n!$ permutations of n nodes, and it is time consuming to train over all of them.

For LGGAN, we use the framework of GCN with residual connections and a node aggregation operator [14] as the discriminator. This discriminator is invariant to node ordering, avoiding the issue. However, for the generator, we use an MLP, which does depend on node ordering. Therefore, we adapt the approach by You et al. [92] where we arrange the nodes in a breadth-first-search (BFS) ordering for each training graph.

In particular, we preprocess the adjacency matrix A and node label matrix L by feeding them into a BFS function. This function takes a random permutation π_g of the nodes in graph g as input, picks a node v_i as the starting node, and then outputs another permutation π'_g that is a BFS ordering of the node in graph g starting from node v_i. By specifying a structure-determined node ordering for the graph, we only need to consider all possible BFS orderings, rather than all possible node permutations. This reduction makes a significant difference for computational complexity when graphs are large.

4.4 Experiments

In this section, we first explore the 6 different settings of LGGAN (three different GAN frameworks for both LGGAN and LGGAN_s) we proposed and discuss about their advantages and disadvantages. Moreover, we also compare our model with other graph generation algorithms to demonstrate its ability to generate high quality labeled graphs in diverse settings.

4.4.1 Baselines

We compare our model against various traditional generative models for graphs, as well as some recently proposed deep graph generative models. For traditional baselines, we compare against the Erdös-Rényi model (E-R) [18], the Barabási-Albert (B-A) model [2]. We also compare against popular generative models that include learnable parameters mixed-membership stochastic block models (MMSB) [1]. Then we also compare with some recently proposed deep graph generative models such as the DeepGMG [49] and GraphRNN [92]. Few current approaches are designed to generate labeled graphs, with the exception of MolGAN [14], which is designed to generate molecular graphs and needs specialized evaluation methods specific to that task, so we are not going to compare with them since they can not be applied to other graph data. Moreover, we also not compare with NetGAN [7] since it's framework is constrained to learn from one single graph and generate a new one.

4.4.2 Datasets

We perform experiments on different kinds of datasets, with varying sizes and characteristics, such as the Enzymes, Protein, D&D dataset [15, 42], and also datasets of citation graphs

such as Cora and CiteSeer [76]. The summary of the statistics for these datasets is shown in Table 4.1

Table 4.1: Details of the graph datasets.

Graph Types	Datasets	# Graphs	# Graph classes	Avg. $\|V\|$	Avg. $\|E\|$	# Node labels
Citation graphs	Cora_small	256	7	38.7	61.6	7
	Citeseer_small	256	6	44.2	82.7	6
	Cora_large	128	7	175.3	326.3	7
	Citeseer_large	128	6	172.5	414.7	6
Protein graphs	PROTEINS	384	2	28.1	53.4	3
	ENZYMES	256	6	39.4	77.7	3

Citation graphs For the citation networks, we used Cora and Citeseer dataset. The Cora dataset is a collection of 2,708 machine learning publications categorized into seven classes and the CiteSeer dataset is a collection of 3,312 research publications crawled from the CiteSeer repository [76]. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. In order to testing the scalability of LGGAN, we extracted different subsets with different graph size by constraining #nodes in graph $|V|$. For Cora_small and CiteSeer_small dataset, We extract 2-hop and 3-hop ego networks with $30 \leq |V| \leq 50$ and for the Cora and CiteSeer dataset, we extract 3-hop ego networks with $150 \leq |V| \leq 200$. In the AC-GAN framework, we set the graph label as the node label of the center node in the graph.

Protein graphs For the protein graphs, we adopt the D&D, ENZYMES and Protein dataset. The D&D is a dataset of 1178 protein structures [15]. Each protein is represented by a graph and labeled into two categories, enzymes and non-enzymes. The ENZYMES dataset consists 600 enzymes [74]. Each enzyme in the dataset are labeled with one of the six EC top-level classes. The protein dataset include proteins from the dataset of enzymes and non-enzymes created by Dobson and Doig [15]. There are two graph labels, enzymes

versus non-enzymes.

4.4.3 Evaluation

To evaluate the quality of the generated graphs, we follow the work in GraphRNN [92] and compare generated graphs with the real ones based on variants of the *maximum mean discrepancy* (MMD) [34] of some graph statistics.

In the experiments, we use four graph statistics to evaluate the generated graphs: degree distribution, clustering coefficient distribution, node label distribution and average orbit counts statistics to further quantitatively evaluate the generated graphs. Moreover, since we are generating labeled graphs, we also want to evaluate the graph distribution in each class. In order to do this, we extract subgraphs for each class from both training graphs and generated graphs and evaluate based on the three metrics we mentioned before (except for the label distribution) to further justify whether the model just simply assign the class based on the label distribution without considering the underling graph structure. Since these evaluation metrics are only applicable to labeled graphs, we only apply it to MMSB and LGGAN.

In order to better evaluation the structure of the labeled graphs being generated besides just the distribution of the labels, we also calculate MMD of those three graph statistics for the sub-graphs in each class and take the average value as the final results. Since only MMSB can be used to directly generate labels, we compare LGGAN to it using the ENZYMES datasets, the results are shown in Table 4.2. We can see that, LGGAN can not only learns a good distribution of the labels but also able to learn the structure within each class and it does a much better job than the MMSB model.

Table 4.2: Comparison of LGGAN with the other labeled graph generation model MMSB on both the graph statistics and average sub-graphs statistics of different classes using MMD evaluation metrics on ENZYMES dataset.

	Graph statistics				Sub-graph statistics		
	Degree	**C**lustering	**O**rbit	**L**abel	Avg. **D**	Avg. **C**	Avg. **O**
MMSB	0.55	1.08	0.05	0.92	0.14	0.20	0.03
LGGAN	**0.09**	**0.17**	**0.03**	**0.01**	**0.13**	**0.15**	**0.01**

4.4.4 Evaluating the Residual GCN Discriminator

To evaluate the influence of residual connections and the depth of the GCN discriminator, we report the results for graph generation on the ENZYMES dataset based on the four evaluation metrics mentioned in Section 5.3.2. Through these tests, we aim to investigate the following design aspects: (1) the GCN depth, (2) residual connections, and (3) different aggregate functions (i.e., max-pooling and concatenation). We plot the results in Figure 4.4.

According to the plots, the performance does not have noticeable improvement with more than two or three GC layers unless we include residual connections. However, when adding the residual connections, using either aggregate function can train deeper GCNs with more than five or six layers, achieving high quality results that outperform other baseline models we compare to in Section 4.4.6. There is no notable difference between the aggregate functions. Since max-pooling does not introduce any additional parameters to learn, we use max-pooling as the aggregation function for residual connections in the remaining experiments.

To further evaluate the performance of GCN based discriminative model, we also compare LGGAN with a simple version where we use MLP as the discriminator. Due to space, please find the details in the supplementary materials.

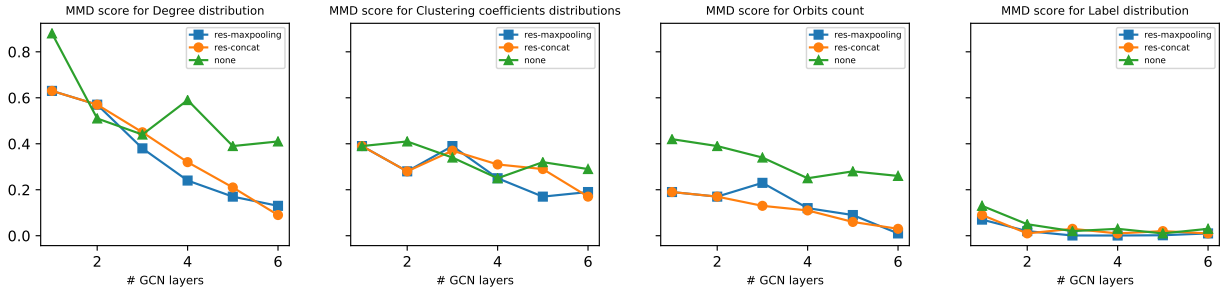


Figure 4.4: Comparison of the results with different GCN layers with or without residual connections.

4.4.5 Comparing different frameworks of LGGAN

LGGAN is a quite flexible framework which can adopt different setting for each part, such as the GAN framework and the model for generator and discriminator. Therefore, at the beginning, we compare different settings to see how they perform on different graph-structured datasets.

Recently variants of GAN have been proposed and claimed that to be able to achieve better results under certain condition, so first of all we adopt three popular ones that are widely known and used as different GAN settings for our model and make a comparison. They are LGGAN_GAN which uses the original GAN framework [31], LGGAN_CGAN which uses the conditional GAN framework [59] and LGGAN_ACGAN which uses the AC-GAN [66] framework. The architecture of them are shown in Figure 4.1. Moreover, we also run experiments to compare two different discriminative models we described in section 4.3.1. The first one is the simple version namely LGGAN_s, which uses multi-layer-perceptron (MLP) to generate graph embedding. Then the other model uses the Jumping knowledge networks (JK-Net) based on multiple layers of GCN to generate graph embedding.

We evaluate those six frameworks on different graph-structured data to compare the quality of the generated graphs and test the generality. We run experiments on two datasets, the

citation dataset: cora_small and the protein dataset: ENZYMES. The results are shown in Table 4.3. From the table we can see that among all those three GAN frameworks, LGGAN_ACGAN achieves the best results on both datasets no matter which discriminative model is used. This matches with our expectation, since with AC-GAN framework, the data that fed into the discriminator also includes the class information which allows it to learn a better embedding and then propagate that information to the generator.

And for different dicriminative models, we can see from the table that using JK-Nets can improve the quality of generated graphs no matter which GAN framework is used. But an interesting thing is that we can see the gap between LGGAN_s and LGGAN on ENZYMES dataset is much larger than the citation networks. And this is because for the cora_small dataset, it is composed of many small two-hop and three-hop ego-networks where the structure is quite simple and uniform so that it is easier to learn. However, with the ENZYMES dataset, the structure is more complicated and diverse, therefore it reveals that the LGGAN_s is hard to generalize to complex data. In contract, the quality of generated graphs with LGGAN using JK-Net is more consistent among different datasets which proves that LGGAN can adaptively adjust to different graph-structured data however the LGGAN_s only performs well on simple graphs such as the ego networks. Therefore, we use the most powerful model LGGAN_ACGAN in the following experiments and refer to it as LGGAN to compare with other baselines.

4.4.6 Comparing to Other Models

For the second experiment, we compare LGGAN to other models for generating graphs, both traditional generative models such as the E-R model, B-A model, MMSB model and deep generative models that were proposed recently, such as GraphRNN and DeepGMG.

Table 4.3: Comparison of LGGAN with different GAN frameworks and discriminative models on Cora-small and ENZYMES dataset.

GAN frameworks	Cora_small				ENZYMES			
	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label
LGGAN_GAN_s	0.27	0.18	0.03	0.37	0.67	0.88	0.004	0.01
LGGAN_GAN	0.21	0.14	0.007	0.15	0.31	0.20	0.01	0.008
LGGAN_CGAN_s	0.18	0.18	0.006	0.35	0.53	0.69	0.04	0.004
LGGAN_CGAN	**0.10**	0.24	0.01	0.19	0.23	**0.13**	0.02	0.01
LGGAN_ACGAN_s	0.14	**0.009**	0.06	0.13	0.51	0.29	0.03	0.01
LGGAN_ACGAN	0.13	0.08	**0.03**	**0.11**	**0.09**	0.17	**0.005**	**0.01**

Notice that DeepGMG can not be used to generate large graphs due to the computational complexity, so the results of DeepGMG on large graph dataset are not available. We compare them on three important aspects for generating graphs. The first one is the quality of the generated graphs which should be able to capture the topology of the training graphs. The second one is the generality, where a good generative model should be able to generalize to different and complex graph-structured data. Then the last one is the scalability where we want the model to be able to scale up to generate large networks instead of restricting to relative small graphs to enable the possibility for more applications.

From Table 4.4 we can see that LGGAN achieves the best performance on all datasets, with 90% decrease of MMD on average compared with traditional baselines, and 30% decrease of MMD compared with the state-of-the-art deep learning baseline GraphRNN. Although GraphRNN performs well on those two protein-related datasets, ENZYMES and protein, it does not maintain the same performance on large datasets, such as cora and D&D.

4.4.7 Downstream Task: Graph Classification

To further evaluate the quality of LGGAN's generated graphs, we extract the generated examples and apply it to a downstream task. We use the synthetic graphs to train a model

Table 4.4: Comparison of LGGAN and other generative models on different graph structured data using MMD evaluation metrics.

	Cora_small				Citeseer_small				Cora				Citeseer			
	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label
E-R	0.68	0.94	0.48	N/A	0.63	0.86	0.12	N/A	0.88	1.45	0.27	N/A	0.82	1.57	**0.06**	N/A
B-A	0.31	0.53	0.11	N/A	0.37	0.18	0.11	N/A	0.54	1.06	0.16	N/A	0.32	1.04	0.08	N/A
MMSB	0.21	0.68	0.07	0.48	**0.17**	0.50	0.11	0.32	**0.12**	0.68	0.09	0.49	0.08	0.50	0.11	0.32
DeepGMG	0.34	0.44	0.27	N/A	0.27	0.36	0.20	N/A	-	-	-	-	-	-	-	-
GraphRNN	0.26	0.38	0.39	N/A	0.19	0.20	0.39	N/A	0.20	0.46	0.11	N/A	**0.20**	1.15	0.14	N/A
LGGAN	**0.13**	**0.08**	**0.03**	**0.11**	**0.17**	**0.13**	**0.04**	**0.09**	0.15	**0.21**	**0.06**	**0.009**	0.25	**0.12**	**0.06**	**0.15**

	Protein				ENZYMES				D&D			
	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label	Degree	Clustering	Orbit	Label
E-R	0.31	1.06	0.28	N/A	0.38	1.26	0.08	N/A	0.15	1.78	**0.03**	N/A
B-A	0.93	0.88	0.05	N/A	1.17	1.08	0.51	N/A	1.08	1.62	0.20	N/A
MMSB	0.46	1.05	0.21	0.01	0.55	1.08	0.05	0.92	0.49	1.77	0.31	1.47
DeepGMG	0.96	0.63	0.16	N/A	0.43	0.38	0.08	N/A	-	-	-	-
GraphRNN	**0.04**	0.18	0.06	N/A	**0.06**	0.20	0.07	N/A	**0.09**	1.13	0.61	N/A
LG-GAN	0.18	**0.15**	**0.02**	**0.005**	0.09	**0.17**	**0.03**	**0.01**	0.13	**0.35**	0.21	**0.01**

for graph classification. We first compute a kernel matrix $K \in \mathbb{R}^{n*n}$ for each of a set of graph kernels, where K_{ij} represents an inner product between representations of G_i and G_j. Then we can train kernel support vector machines (SVM) to classify the graphs. In our experiment, we choose three popular graph kernels: the graphlet kernel (GK) based on subgraph patterns, the shortest-path kernel (SP) based on random walks, and the Weisfeiler-Lehman subtree kernel (WL) based on subtrees.

We compare performance when training on synthetic graphs from LGGAN, MMSB (the only baseline model that can generate labeled graphs), and real graphs. We run this procedure with three datasets: Cora_small, ENZYMES and PROTEINS. For each dataset, we run ten trials and calculate the average value of the accuracy.

We list results in Table 4.5. The accuracy of models trained with graphs generated by LGGAN is close to those trained using the real graphs, especially compared to the models trained with graphs generated by MMSB. These results suggest that the graphs generated by LGGAN can better capture the important aspects of graph structure.

Table 4.5: Comparison of graph classification accuracy with different kernels: graphlet kernel (GK), shortest-path kernel (SP) and Weisfeiler-Lehman subtree kernel (WL) on citation and protein datasets with the other labeled graph generation model MMSB.

	Cora_small			ENZYMES			PROTEINS		
	GK	SP	WL	GK	SP	WL	GK	SP	WL
gen_MMSB	22.62	21.34	23.15	15.38	14.29	17.86	64.32	65.61	64.29
gen_LGGAN	23.44	26.56	26.92	23.08	26.92	30.19	65.61	66.54	69.23
real_graphs	26.56	29.69	34.32	23.08	29.68	35.71	69.05	73.81	76.19

4.4.8 Scalability

To evaluate the scalability of these methods, we perform experiments on two different subsets of the Cora dataset with different graph sizes: the Cora_small and Cora datasets. As listed in Table 4.4, the traditional models all create a large gap between these two datasets in terms of three evaluation metrics. For the deep generative models, DeepGMG cannot generate large graphs due to the computational complexity of its generation procedure which tries to add node one by one increasingly. And compared to GraphRNN, LGGAN MMD scores barely increase compare to smaller dataset, suggesting that our model is more reliable and has the best ability to scale up to large graphs.

4.4.9 Generality

To evaluate the ability of LGGAN to adapt to different graph-structured data, we evaluate the results of all methods on the different domains of citation ego-networks (Cora) and molecular protein graphs (ENZYMES). From Table 4.4, LGGAN achieves more consistent results on various datasets compared to other models, where some of them suffer from the issue of generalization such as MolGAN which can only be used to generate specific or limited types of graph-structured data.

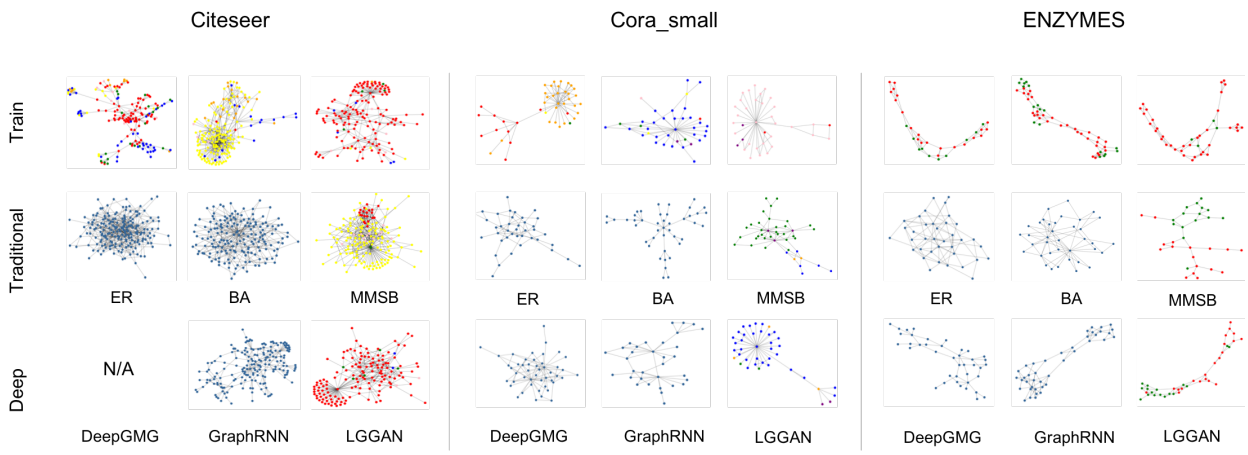


Figure 4.5: Visualization of training graphs (first row); graphs generated by traditional models (second row): E-R model, B-A model, MMSB model; graphs generated by deep models (third row): DeepGMG, GraphRNN, LGGAN for different datasets.

Some examples are visualized in Figure 4.5, which contains graphs generated by our model and the baselines. Although it is not as intuitive for humans to assess as, e.g., natural images, one can still see that LGGAN appears to capture the typical structures of datasets better than other models.

4.4.10 Diversity

A good labeled graph generative model should generate diverse examples. Two types of diversity are important: (1) **diversity among generated examples** would capture the natural variations in real graphs; and (2) **diversity compared to training examples** ensures that the generative model is doing more than exactly memorizing some training examples and outputting copies of them. Generative models should balance the need for generated outputs to be new graphs unseen during training while retaining important properties of the real data.

Therefore, to investigate to what extent our model can maintain these types of diversity, we

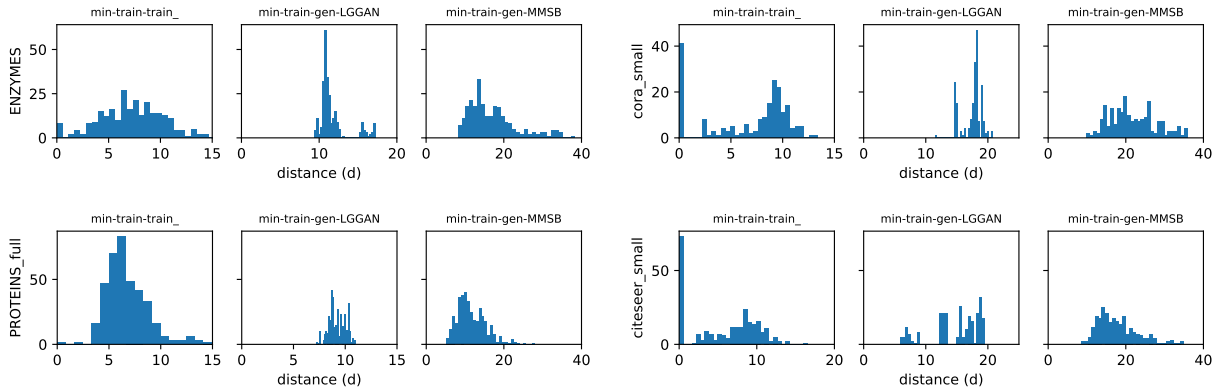


Figure 4.6: Histogram of the distances between training graphs and graphs generated by LGGAN and MMSB.

calculate the Weisfeiler-Lehman kernel value for both the training graphs and the generated graphs (by MMSB and LGGAN) and compute the kernel distance d between any graphs g_i and g_j as

$$d_{ij} = \sqrt{K(g_i, g_i) + K(g_j, g_j) - 2K(g_i, g_j)}. \tag{4.10}$$

We plot histograms of the minimum distances between each generated example and the training set in Figure 4.6 for four datasets, Cora_small, Citeseer_small, PROTEINS and ENZYMES. In each plot, the left column shows the minimum distance of each training graph to any other graph; the middle column shows the minimum distance for each graph generated by LGGAN to the training graphs; and the right column shows the same for MMSB. These plots suggest that the graphs generated by our model are more similar to training graphs than the examples generated by MMSB, yet they are not exact copies of training graphs and have a similar diversity of graph distances as the real data.

4.5 Conclusion

In this work, we proposed a deep generative model using a GAN framework that generates labeled graphs. These labeled graphs can mimic distributions of citation graphs, knowledge graphs, social networks, and more. We also introduced an evaluation method for labeled graphs to measure how well the model learns the sub-structure of the labeled graphs. Our model can be useful for simulation studies, especially when access to labeled graph data is limited by access or privacy concerns. We can use these models to generate synthetic datasets or augment existing datasets, to do graph-based analyses such as communication segmentation, node classification, anomaly detection, and link prediction. The experiments show that it outperforms other state-of-the-art models for generating graphs while also being capable of the previously unaddressed task of generating labels for nodes.

Chapter 5

Attention-based Graph Evolution

5.1 Introduction

Graphs are complex and versatile data structures that can be used to represent various kinds of real-world data with complex relationships. However, some special properties of graphs, such as discrete form and order-invariance, make generation of graphs a harder problem than that for other data types such as images and natural language. Naive approaches that represent graphs as standard data types such as adjacency matrices and label lists can cause loss of some structural information.

As an fundamental topic in graph modeling, graph generation has a long history that began as early as the 1950s [18]. However, most traditional methods rely on prior knowledge of the graph topology and are limited in capability of learning generative properties from observations. To solve this problem, researchers have recently been exploring trainable deep models for graph generation based on the effectiveness of graph neural networks—e.g., graph convolutional networks [43]—which have been applied to various kinds of data describing, for example, molecular chemicals for drug design and scientific publications for predicting citations [78, 92]. However, these approaches are unconditional generative models, which limits their control over the generating procedure and makes them unable to produce graphs in context. These limitations restrict the applicability of these approaches to real world settings where graphs transform from one state to another and evolve in dynamic network

settings.

Modeling graph evolution is an important task that can be applied to various practical applications. A model of graph evolution would be a powerful tool for both predicting the future and the transformation of networks. For example, a marketer aiming to post an advertisement on an online social network may only have access to short-hop ego networks around users, but they need to know how the information would spread into the extended network beyond these ego networks. In disease control and prevention, when an infectious disease emerges and starts to spread, it is important to understand how it may spread beyond the visible network. Because graph data represents real-world phenomena that is changing or incompletely observed, there are many other examples of problems that could benefit from new tools for modeling graph evolution. Yet existing methods lack the flexibility of deep generative models or the ability to condition on previous graph states.

To provide this missing capability, we introduce an attention-based graph evolution model (AGE). AGE is a model for conditional graph generation based on the attention mechanism that allows consideration of global information with parallel computation across all graph nodes. AGE adopts the encoder-decoder structure, where the encoder tries to learn the representation of conditioned graphs using a self-attention mechanism, and the decoder tries to generate the representation of the target graphs using the correlation with the conditioned graphs and also with itself. The decoder can thus capture both global and local information. This graph-conditioned generation framework greatly enriches the potential applications for graph generation. AGE can be used to model not only graph evolution in space and in time, but also the transformation between graphs from one state to another.

To evaluate how AGE performs on this problem setting, we perform experiments on datasets in various areas such as computer networks, citation networks, social networks and rating networks. The experiment results in terms of both the evaluation metrics and the graph

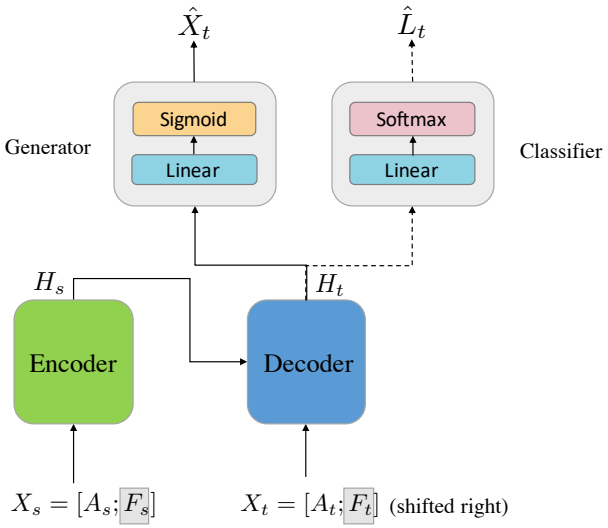


Figure 5.1: The model architecture of AGE.

visualization, show that AGE can not only generate extremely realistic graphs, but also has the strong ability to model the evolution of graphs as a powerful conditioned graph generative model. Also experiments on different applications and network datasets also prove that AGE can adapt to various kinds of evolution and transformation between graphs and performs consistently well across different evaluation metrics and schemes.

5.2 Attention-based Graph Evolution Model

We define the prediction of graph evolution as taking an existing *source* graph with nodes (with or without label) as input and predicting, or generating, a transformed version of the graph, or *target* graph. The transformation can represent change over time in a dynamic graph, or expansion in space, such as how ego networks change as we expand out to more steps. Many powerful graph generation models are autoregressive, meaning they generate graphs by sequentially adding new nodes and evaluating relevant possible edges [49, 92]. We also adopt this approach and further incorporate an attention-based transformer [83] to

process a source graph, and model its evolution. We use an attention mechanism instead of graph convolutional network [43] because attention models can overcome depth limitations of GCNs. Therefore, they can learn more powerful embeddings based on global context.

We model the graph generation procedure as a sequential problem by adding new nodes step by step. Many other graph generative models such as GraphRNN [92] and DeepGMG [49] proposed the same structure before; however, they all suffer from the efficiency problem since the sequentiality prohibits parallelization within instances during the training procedure. This drawback limits their applications on large graphs especially for DeepGMG which can only be applied to graphs with less than 30 nodes. To avoid these issues with long sequences, we adopt the transformer framework, which instead processes the nodes ordered in the sequence in parallel while using the attention mechanism to incorporate information from all other nodes, even those far away in the sequence. By processing the nodes in parallel, it also significantly shortens the training time, making it much faster than other models.

The architecture of AGE is shown in Figure 5.1. As in the standard transformer framework, AGE consists of two main components: an encoder E and a decoder D. The encoder learns to represent source graphs through a multi-head attention mechanism. The decoder, which is an autoregressive model, then sequentially generates one new node at a time, with possible edges connecting to existing nodes (i.e., the nodes in source graphs and the ones generated previously). In our model, a graph is represented as $G = (\boldsymbol{F}, \boldsymbol{A}, \boldsymbol{L})$ where $\boldsymbol{F}$ is the feature matrix of nodes in source graphs (if one is given), $\boldsymbol{A}$ is the adjacency matrix of source graphs, and $\boldsymbol{L}$ is the label matrix of the nodes in the graph (if is available). Among these three components, the adjacency matrix is essential. In some settings, we can leave out the features and labels if we do not have this information. The goal of AGE is therefore to learn a mapping from a source graph G_s to a target graph G_t.

5.2.1 Attention

We represent the hidden states of the encoder as a set of key-value pairs $(\boldsymbol{K}, \boldsymbol{V})$ and the goal of the encoder is to map a query $\boldsymbol{Q}$ and the key-value pairs to the output, which is the encoded representations of the input. We adopted the scaled dot-product attention mechanism, where the output is the weighted sum of the values and the weight is determined by the dot-product of the query $\boldsymbol{Q}$ with the keys $\boldsymbol{K}$:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{5.1}$$

We use a multi-head attention [83] by concatenating multiple attention modules, which allows the model to jointly pay attention to information from different representation subspaces.

$$\begin{aligned} \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}^O \\ \text{where} \quad \text{head}_i &= \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \end{aligned} \tag{5.2}$$

5.2.2 Self Attention

In AGE, the encoder and the decoder each have their own self-attention block, which is designed to learn high-level node representations based on other nodes within the same graph. In the encoder, the representation of node i in source graphs is updated based on the following rule:

$$\boldsymbol{h}_i^{t+1} = \boldsymbol{h}_i^t + \sigma(\sum_{j=1}^{N_s} a_{i,j}^t \times \boldsymbol{W}_s^s \boldsymbol{h}_j^t) \tag{5.3}$$

where $a_{i,j}$ is the normalized weights the model learns between node v_i and v_j, $\boldsymbol{h}_i$ is the hidden node feature of node i, N_s is the number of nodes in the source graph, σ is a nonlinear activation and $\boldsymbol{W}_s^s$ is the linear transformation where the weights are learnable parameters

separately instantiated for each attention step in the model. The edge weights between two nodes are computed based on the attention mechanism:

$$e_{i,j}^{t+1} = \text{Attention}(\boldsymbol{W}_s \boldsymbol{h}_j^{t+1}, \boldsymbol{W}_s' \boldsymbol{h}_i^{t+1}) \tag{5.4}$$

where $e_{i,j}$ is the attention weight of edge from node i in source graph to node j in target graph, and $\boldsymbol{W}_s$ and $\boldsymbol{W}_s'$ are linear transformations. The weights are again learnable parameters separately instantiated for each attention step in the model. We normalize the attention weights of node j with all the other nodes in the graph:

$$a_{i,j}^{t+1} = \frac{\exp(e_{i,j}^{t+1})}{\sum_{k=1}^{N_s} \exp(e_{k,j}^{t+1})}. \tag{5.5}$$

5.2.3 Source-Target Attention

To learn the correlations between the nodes in source graph and the ones to be generated by decoder, we apply a source-target attention block after the self-attention operations. The representation of a predicted node j in generated graph is updated based on the learned embeddings of all nodes in the source graph using the following rule:

$$\boldsymbol{h}_j = \boldsymbol{h}_j + \sigma \left(\sum_{i=1}^{N_s} a_{i,j} \times \boldsymbol{W}_s^t \boldsymbol{h}_i \right) \tag{5.6}$$

where σ is a nonlinear activation function, $\boldsymbol{W}_s^t$ is a learnable linear transformation and $a_{i,j}$ is the normalized weights the model learned between node v_i and v_j. The edge weights between two nodes in different graphs are typically calculated in the same way as shown in Equation 5.4 and 5.5.

Table 5.1: Attributes of the graph datasets and evolution settings. Some graphs have features, some have labels, and some have time stamps that enable us to study different types of evolution in space and time.

Graph Types	Datasets	# S-T Graph pairs	Avg. $\|V_t\|$	Avg. $\|E_t\|$	features	labels	time-stamp
Graph Evolution in Space	Cora_small	462	24.6	64.4	✓	✓	✗
	Citeseer	444	83.0	234.0	✓	✓	✗
Graph Evolution in Time	Cit-HepTh_small	706	26.5	96.1	✗	✗	✓
	Facebook-friend	129	96.2	332.8	✓	✓	✗
	Cit-HepPh	525	65.9	240.7	✗	✗	✓
	Bitcoin-OTC	130	56.7	186.8	✗	✓	✓
Graph Evolution in Time with Deletion	Oregon	60	85.5	193.5	✗	✗	✓

5.2.4 Encoder

In AGE, the encoder takes in a source graph G_s represented by its initial representations $\boldsymbol{X}_s = [\boldsymbol{A}_s; \boldsymbol{F}_s]$ (we can leave out $\boldsymbol{F}_s$ if it's not given) and maps it to a high-level embedding. Here $\boldsymbol{A}_s$ and $\boldsymbol{F}_s$ are the adjacency matrix and the feature matrix of the source graph, where the nodes are arranged in a breadth-first-search (BFS) ordering. We concatenate the feature matrix if we have one. When available, we can use features to generate new node features in addition to the graph structure. In our experiments, we focus on undirected, unweighted graphs where the adjacency matrix $\boldsymbol{A}$ is a symmetric binary matrix with each element represents the connectivity of a pair of nodes, but it can be easily extended to both directed and weighted graphs.

We use a fixed maximum number of nodes, N_s for the source graph and N_t for the target graph. AGE can learn about and generate structures with various sizes smaller than these maximums by ignoring isolated nodes in the generated graph. We also define a fixed minimum number of nodes for both source and target graphs to ensure that the input graph is not empty, and to ensure that there are some differences between the source and target graphs.

5.2.5 Decoder

The decoder is composed of several stacked attention modules that alternate self-attention and source-target attention layers. The input for the decoder includes two parts: the target graphs G_t and the learned embeddings $\boldsymbol{H}_s$ of the source graph (provided by the encoder). The target graph G_t is represented by the shifted node representations $\boldsymbol{X}_t = [\boldsymbol{A}_t; \boldsymbol{F}_t]$ (we can leave out $\boldsymbol{F}_t$ if it's not given), with a start token and an end token filled at the beginning and appended to the bottom to ensure that the decoder predicts the next node based on the previously generated set. Like the source graph, the nodes in the target graph are also arranged in a breadth-first-search (BFS) ordering at training time, and the model is expected to learn to generate BFS orders.

Given $\boldsymbol{H}_s$, the autoregressive decoder generates an output sequence of nodes one at a time, where each step is also conditioned on the previously generated nodes.

The decoder maps the embedding to the space of adjacency matrices and space of label matrices (if the data has label information) to reconstruct the generated graphs. We use a generator which is a combination of a linear transformation and the sigmoid activation function to map $\boldsymbol{H}_t$ to the adjacency matrix $\hat{\boldsymbol{A}}_t$:

$$\hat{\boldsymbol{A}}_t = \text{sigmoid}(\boldsymbol{W}_g \boldsymbol{H}_t). \tag{5.7}$$

We use a classifier which is a combination of a linear transformation and the softmax activation function to map H_t to the label vector $\hat{\boldsymbol{L}}_t$:

$$\hat{\boldsymbol{L}}_t = \text{softmax}(\boldsymbol{W}_c \boldsymbol{H}_t) \tag{5.8}$$

For the predicted adjacency matrix, we use the binary cross entropy loss function to measure

the differences:

$$L_{\text{adj}} = -\sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{A}_{ij} \log(\hat{\boldsymbol{A}}_{ij}) + (1 - \boldsymbol{A}_{ij}) \log(1 - (\hat{\boldsymbol{A}}_{ij})) \tag{5.9}$$

Moreover, if the data has the label information, we also added the loss on labels based on label smoothing using the KL divergence loss.

5.3 Experiments

In this section, we compare AGE with other graph generation methods on various conditioned graph generation problems to demonstrate its wide applicability. The statistics of them are listed in Table 5.1 and the details of how we construct them are explained below. In the following experiments, we extract 70% of the data as training set, 20% for the valid set and 10% for the test sets. We used six attention layers for both self-attention and source-target attention block and within each, we set the number of heads to eight.

5.3.1 Baselines

As we previously discussed, some work have been proposed to generate graphs using deep models, however few of them can condition on existing graphs for general tasks. Therefore, we compare AGE against two categories of other relevant models. The first set consists of methods that can (or can be modified to) generate graphs conditionally, such as the Erdös-Rényi model (E-R) [18] and the Barabási-Albert (B-A) model [2]. These generative models iteratively grow a graph, so they can start from an existing graph. The second set of more recent methods are unconditional graph generation models, such as the mixed-membership

stochastic block models (MMSB) [1], DeepGMG [49] and GraphRNN [92], which include state-of-the-art deep generative models. In our experiments, we train these directly on the target graphs without the source graphs.

5.3.2 Evaluation Metrics

We evaluate the generated graphs in two modes. First, we evaluate whether the distribution of generated target graphs is realistic, which captures how well the conditional generative model captures variation in generated graphs. We compute the distances of the distributions of generated graphs and of the target graphs using *maximum mean discrepancy* (MMD) [34], following the evaluation procedure used by You et al. [92]. We compute MMD for four graph statistics: degree distribution, clustering coefficient distribution, node-label distribution (if labels are available), and average orbit count statistics. A model that faithfully captures the conditional distribution over target graphs should have low MMD with the set of true target graphs.

In our second mode of evaluation, we compute the similarity between the generated graph and the true target graph for each source graph. This metric evaluates the performance of conditional generation. We calculate the graph similarities using three graph kernels: the shortest path kernel [9] (GK_st), the graphlet sampling kernel [77] (GK_gs), and the SVM-θ kernel [41] (GK_svm). A good conditional graph generator should generate graphs with high similarity to the true target graphs.

5.3.3 Graph Evolution in Space

Our first evaluation setting considers the graph evolution problem in space. In real-world networks, graph data is collected by subsampling from larger graphs. Due to resource con-

straints, data collection may not gather as large subsamples as needed. A generative model that can conditionally add nodes in a manner consistent with how graphs grow as one expands the subsample could enable larger analyses of semi-synthetic networks.

Datasets We test this problem setting on citation networks. The problem is to predict the expansion of ego networks with farther-hop neighbors. We used the Cora and Citeseer datasets [76]. The Cora dataset is a collection of 2,708 machine learning publications categorized into seven classes, and the CiteSeer dataset is a collection of 3,312 research publications with six node classes. We evaluated our models with different graph sizes. For small datasets (Cora_small and Citeseer_small), we extract one-hop ($G_s = G_1 = \{V_1, E_1\}$) and two-hop ($G_t = G_2 = \{V_2, E_2\}$) ego networks with $5 \leq |V_1| \leq 20$ and $30 \leq |V_2| \leq 50$ as the source and target graphs. For the large datasets (Cora and Citeseer), we extract two-hop ($G_s = G_2 = \{V_2, E_2\}$) and three-hop ($G_t = G_3 = \{V_3, E_3\}$) ego networks with $10 \leq |V_2| \leq 50$ and $40 \leq |V_3| \leq 170$ as the source and target graphs.

Data construction for this problem is shown in Figure 5.2. The training data consists of graph pairs extracted from the datasets. The source graph G_s is the i-hop ego network where the initial embeddings is constructed by concatenating the adjacency matrix $\boldsymbol{A}_s$ and the feature matrix $\boldsymbol{F}_s$ (if $\boldsymbol{F}_s$ is given). The target graphs G_t are the $(i+1)$-hop ego networks of the same node v where the initial embeddings is constructed by concatenating the adjacency matrix $\boldsymbol{A}_t$ and the feature matrix $\boldsymbol{F}_t$. Notice that in Figure 5.2, $\boldsymbol{A}_s$ is in the top left corner of $\boldsymbol{A}_t$ and $\boldsymbol{F}_s$ is in the top of $\boldsymbol{F}_t$ because we have $G_s \in G_t$ for expansion problem and we arranged the nodes in the BFS order. In this case, if we know there is no node or edge deletion, we can train the decoder only with G_{t-s} and append the source graph G_s only for final evaluations. AGE can also model graph evolution with deletion, details will be shown in section 5.3.5. Here, if the feature information is not available in the data, the embedding would only be

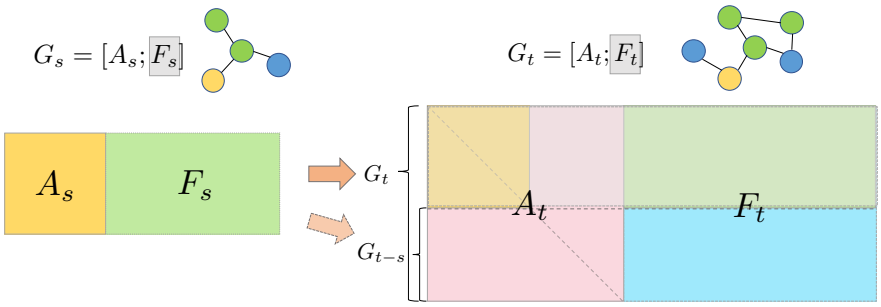


Figure 5.2: Data construction for graph evolution in space. The graph and matrix on the left represents the input source graph, which contains a portion of the full target graph on the right. The full target graph contains the adjacency and feature matrices of the source graph in this setting.

the adjacency matrix. The problem is to model how ego networks expand to the nodes of next-hop distance. We can therefore ignore the possibility of node and edge deletion in this setting.

We compare AGE with other graph generative models listed in section 5.3.1. Due to the computational complexity of the DeepGMG model, we only perform experiments with it on small graphs, such as Cora_small, Cit-HepTh_small and Bitcoin-OTC_small. Also, some methods can not generate node labels while generating the graph structure, we can not evaluate the distance between the label distributions, therefore the metric "MMD_label" is shown as N/A.

Results are listed in Table 5.2 (In all tables, values are rounded to two decimal places). The metrics indicate that AGE is a strong conditional graph generator in both its ability to mimic graph distributions and match the target graphs. Considering the evaluation of the distance between the distributions of generated graphs and target graphs, AGE achieves the best scores. AGE scores less than 0.1 MMD on all cases, with at least a 30% decrease compared to the second best method, GraphRNN on two datasets with different graph sizes. This result corroborates that, as a graph generative model, AGE can generate realistic graphs that appear to be from the same distribution as the true target graphs.

Table 5.2: Comparison of AGE and other generative models on graph evolution in space using MMD evaluation metrics and graph kernel similarities.

	Cora_small							Citeseer						
	Distribution distance				Graph similarity			Distribution distance				Graph similarity		
	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}
E-R	0.33	0.53	0.11	N/A	0.77	0.74	0.93	0.66	0.62	0.21	N/A	0.72	0.74	0.96
B-A	0.35	0.40	0.22	N/A	0.71	0.50	0.53	0.14	0.29	0.14	N/A	0.80	0.78	0.91
MMSB	0.09	0.53	0.14	0.16	0.93	**0.85**	0.98	0.24	1.01	0.15	0.09	0.93	0.84	0.98
DeepGMG	0.37	0.54	0.06	N/A	0.88	0.79	0.90	-	-	-	-	-	-	-
GraphRNN	0.08	0.34	0.09	N/A	0.91	0.76	0.94	0.03	0.23	0.03	N/A	0.86	0.81	0.95
AGE	**0.01**	**0.04**	**0.01**	**0.01**	**0.94**	**0.85**	**0.99**	**0.01**	**0.01**	**0.02**	**0.01**	**0.94**	**0.85**	**0.99**

Considering how well generated graphs match the specific target graphs. We calculate the graph similarities between the generated graphs and the target graphs. The kernel similarity scores are are normalized, so they range from 0 to 1. The graphs AGE generates consistently have the best similarity scores.

5.3.4 Graph Evolution in Time

Many graph generation methods are focused on static graphs. However in practice, many networks are not static. Instead, they change and evolve over time, with the addition of new nodes and edges, such as in citation networks and collaboration networks, and also with the deletion of existing nodes and edges, such as in computer networks and social networks.

Dataset To evaluate the performance of AGE on modeling the evolution of networks, we first study cases where the source graphs evolve by growing, acquiring new nodes and edges. This setting does not include edge or node deletion. We perform experiments on the following three datasets. The first dataset we use is the Facebook Friendship Networks [86] data, which contains friendship data of Facebook users. The data spans from 2006 to 2009. For the experiments, we extract two-hop ($G_2 = \{V_2, E_2\}$) ego networks with $30 \leq |V_2| \leq 120$ at year 2006 and 2009 respectively as the source and target graphs. The second dataset we use

is the Bitcoin Networks [45] data, which is a who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC. It maintain a record of users' reputation to help other users find reliable users to trade with, buyers rate sellers in each transaction with a credibility score from -10 to $+10$. In our experiments, we compute the average ratings of each user as their credibility labels. Since there are few examples with scores below 0, we convert all negative scores to 0. The data spans from 2011 to 2016. We extract two-hop ($G_2 = \{V_2, E_2\}$) ego networks with $30 \leq |V_2| \leq 120$ at year 2011 and 2016 respectively as the source and target graphs. The third dataset we use are two citation networks in Physics from the e-print arXiv: cit-HepPh and cit-HepTh [27]. Cit-HepPh includes 34,546 papers with 421,578 edges and cit-HepTh includes 27,770 papers with 352,807 edges. Both datasets covers papers in the period from 1993 to 2003. We evaluated our models with different graph sizes. For small datasets (cit-HepTh_small), we extract two-hop ($G_2 = \{V_2, E_2\}$) ego networks with $20 \leq |V_2| \leq 50$ at year 1993 and 2003 respectively as the source and the target graphs. For the large datasets (denoted cit-HepPh and cit-HepTh), we extract two-hop ego networks with $30 \leq |V_2| \leq 120$ instead.

The settings here is similar to previous problem setup, but the key difference is that now each source graph G_t^v is the i-hop ego network of node v at time t, and the target graph is the i-hop ego networks G_{t+1}^v of the same node at time $t+1$. Here, we have $G_t^v \in G_{t+1}^v$, and the problem is to model how networks evolve (or grow) with actual time.

We compare AGE with other graph generative models and the results are shown in Table 5.3. The evaluation results show that AGE can accurately model the graph evolution or graph growing procedure over time. We compute the distance between the distributions of generated graphs and target graphs, and, as before, AGE achieves the best scores among all the generative models regarding the realism of the generated graphs. Again, this is strong evidence that AGE can generate realistic graphs that appear to be from the same distribution

Table 5.3: Comparison of AGE and other generative models on graph evolution in time using MMD evaluation metrics and graph kernel similarities.

	Facebook-friend							**Cit-HepPh**						
	Distribution distance				Graph similarity			Distribution distance				Graph similarity		
	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}
E-R	0.54	1.25	0.32	-	0.50	0.55	0.98	0.43	1.15	0.27	-	0.55	0.56	0.93
B-A	0.49	1.08	0.34	-	0.78	0.85	0.78	0.43	0.65	0.16	-	0.71	0.85	0.94
MMSB	**0.09**	0.53	**0.14**	-	0.89	0.85	0.98	0.19	1.20	0.14	-	0.84	0.59	0.99
GraphRNN	0.17	0.18	0.21	-	0.76	0.64	0.95	**0.08**	0.81	0.08	-	0.86	0.69	0.92
AGE	**0.09**	**0.01**	0.19	-	**0.93**	**0.88**	**0.99**	0.10	**0.01**	**0.04**	-	**0.94**	**0.89**	**0.99**
	Bitcoin-OTC							**Cit-HepTh_small**						
	Distribution distance				Graph similarity			Distribution distance				Graph similarity		
	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}
E-R	0.63	1.12	0.21	N/A	0.57	0.43	0.98	0.33	0.81	0.22	-	0.64	0.24	0.93
B-A	0.40	0.46	0.14	N/A	0.68	0.90	0.95	0.37	0.71	0.28	-	0.63	0.57	0.86
MMSB	0.30	1.17	0.12	0.15	0.80	0.59	0.98	0.28	0.83	0.42	-	0.82	0.36	0.98
DeepGMG	-	-	-	-	-	-	-	0.12	0.68	0.20	-	0.93	0.56	0.92
GraphRNN	0.16	0.43	0.20	N/A	0.84	0.64	0.94	0.05	0.27	0.07	-	0.96	0.80	0.95
AGE	**0.08**	**0.04**	**0.10**	**0.01**	**0.97**	**0.92**	**0.99**	**0.04**	**0.01**	**0.04**	-	**0.99**	**0.94**	**0.99**

of the target graphs.

Considering the graph similarities between the generated graphs by all models and the target graphs, Table 5.3 shows that among all models, AGE is the only one that can reach similarity 0.9 for all three graph kernels, while the other methods cannot consistently score high across different kernels. This suggests some aspect of graph similarity is not satisfied by these other generation procedures. These results again demonstrate that AGE represents a significant step in our ability to model the evolution of graphs in time.

5.3.5 Graph Evolution in Time with Deletion

To evaluate the performance of AGE on modeling the evolution of graphs with deletion, study cases where the source graphs evolves with not only addition of new nodes and edges, but also allows the deletion of existing nodes and edges. We perform experiments on the following dataset with varying sizes and characteristics.

Table 5.4: Comparison of AGE and other generative models on graph evolution in time with deletion using MMD evaluation metrics and graph kernel similarities.

	Oregon						
	Distribution distance				Graph similarity		
	Degree	Clustering	Orbit	Label	GK_{st}	GK_{gs}	GK_{svm}
E-R	0.51	0.37	0.25	-	0.55	0.63	0.96
B-A	0.11	0.35	0.21	-	0.85	0.98	0.92
MMSB	0.54	0.39	0.29	-	0.71	0.45	0.93
GraphRNN	0.14	0.12	0.20	-	0.91	0.88	0.93
AGE	**0.01**	**0.01**	**0.01**	-	**0.99**	**0.99**	**0.99**

Datasets For this problem setting, we use the Computer network dataset [46], which is a network describing peering information inferred from Oregon route-views with nine different timestamps in total. we extract two-hop ($G_2 = \{V_2, E_2\}$) ego networks with $30 \leq |V_2| \leq 120$ at the first and last timestamp respectively as the source and target graphs.

In this experiment, we focus on the more difficult problem of modeling the evolution of graphs with deletion where the source graph G_t^v is the i-hop ego networks of node v at time t and the target graph is the i-hop ego networks G_{t+1}^v of the same node at time $t+1$. The difference with second experiment is that in this case, the condition that $G_t^v \subseteq G_{t+1}^v$ does not hold in this case.

We compare AGE with other graph generative models and list the results in Table 5.4. The evaluation results show even for this more complex problem, AGE still maintains a high-level of performance compared to all the other generative models in terms of both the realism of generated graphs and the similarity of the generated graphs to the target ones. Therefore, together with the second experiment, AGE is able to learn not only graph evolution through growth, but also the more complex setting of volatile evolution.

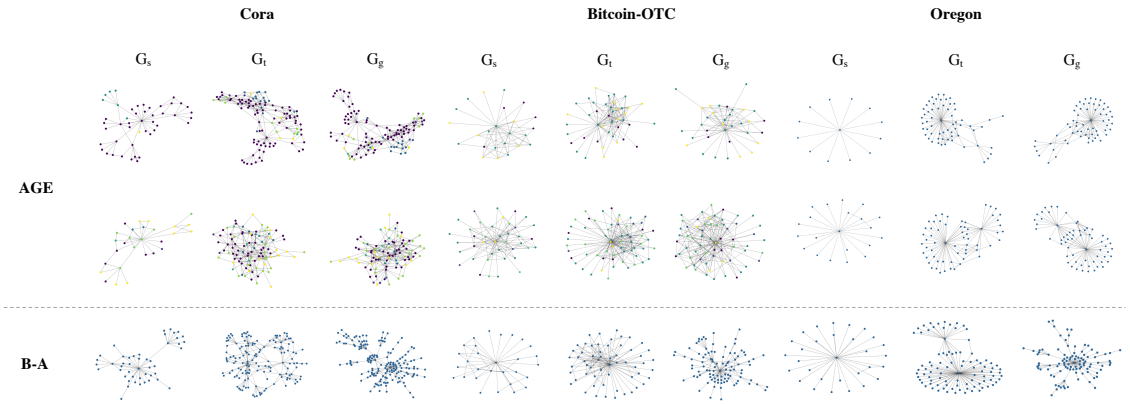


Figure 5.3: Visualization of the graphs generated by two conditioned graph generators, B-A and AGE, on three datasets. For each dataset, we visualize the source graphs (G_s), target graphs (G_t), and generated graphs (G_g). The graphs generated by AGE better mimic the structure of the true target graphs than the preferential-attachment B-A predictions.

5.3.6 Visualization

Some examples are visualized in Figure 5.3, which contains graphs generated by AGE and another conditioned graph generator: the B-A model [2]. Although compare to images, it is harder for human to evaluate these generated graphs through visualization, we can still tell that the graphs generated by AGE can capture the structures of those three datasets better than the B-A model.

5.4 Conclusion

In this work, we proposed attention-based graph evolution (AGE), a conditioned generative model for graphs based on the attention mechanism, which can model graph evolution in both space and time. AGE is capable of generating graphs conditioned on existing graphs. Our model can be useful for many applications in various domains, such as for predicting information propagation in social networks, disease control for healthcare, and traffic prediction in road networks. We model graph generation as a sequential problem, yet we are able

to train AGE models in parallel by adopting the transformer framework. Our experimental results demonstrate that AGE is a powerful and efficient conditioned graph generative model, which outperforms all the other state-of-the-art deep generative models for graphs. In our several experiments on various datasets, AGE is to be able to adapt to various kinds of evolution or transformations between graphs, and it performs consistently well in terms of both the realism of its generated graphs and the similarity to ground-truth target graphs. Finally, AGE has a flexible structure that can be used to generate graphs with or without features and labels. This flexibility thus enables a wider range of applications by allowing it to model many forms of graph evolution.

Chapter 6

Conclusion and Future Works

6.1 Contribution

Graphs are one of the most important and powerful data structures for conveying the complex and correlated information among data points. In this research, we aim to provide more robust and accurate models for some graph specific tasks, such as collective classification and graph generation, by designing deep learning models to learn better task-specific representations for graphs. First, we studied the collective classification problem in graphs and proposed recurrent collective classification, a variant of the iterative classification algorithm that uses differentiable operations, enabling back-propagation of error gradients to directly optimize the model parameters. The concept of collective classification has long been understood to be a principled approach to classifying nodes in networks, but in practice, it often suffers from making only small improvements, or not improving at all. One cause for this could be the faulty training procedure that we correct in this paper. Our experiments demonstrate dramatic improvements in training accuracy, which translate to significant, but less dramatic improvements in testing performance. RCC is a key step toward fully realizing the power of collective classification. Thus, an important aspect to consider to further improve the effectiveness of collective classifiers is the generalization behavior of collective models.

In the next part of the thesis, we studied the problem of graph generation using deep gener-

ative models. We proposed a deep generative model using a GAN framework that generates labeled graphs. These labeled graphs can mimic distributions of citation graphs, knowledge graphs, social networks, and more. Moreover, we also introduced an evaluation method for labeled graphs to measure how well the model learns the sub-structure of the labeled graphs. LGGAN can be useful for simulation studies, especially when access to labeled graph data is limited by access or privacy concerns. We can use these models to generate datasets, or augment existing datasets, to do graph-based analyses such as communication segmentation, node classification, anomaly detection, and link prediction. The experiments show that it outperforms other state-of-the-art models for generating graphs while also being capable of the previously unaddressed task of generating labels for nodes.

For the last part of the thesis, we proposed attention-based graph evolution (AGE), a conditioned generative model for graphs based on the attention mechanism, which can model graph evolution in both space and time. AGE is capable of generating graphs conditioned on existing graphs. AGE can be useful for many applications in various domains, such as for predicting information propagation in social networks, disease control for healthcare, and traffic prediction in road networks. We model graph generation as a sequential problem, yet we are able to train AGE models in parallel by adopting the transformer framework. Our experimental results demonstrate that AGE is a powerful and efficient conditioned graph generative model, which outperforms all the other state-of-the-art deep generative models for graphs. In our several experiments on various datasets, AGE is to be able to adapt to various kinds of evolution or transformations between graphs, and it performs consistently well in terms of both the realism of its generated graphs and the similarity to ground-truth target graphs. Finally, AGE has a flexible structure that can be used to generate graphs with or without features and labels. This flexibility thus enables a wider range of applications by allowing it to model many forms of graph evolution.

6.2 Future Works

Compared to representation learning and generative models for images and text, generative models for graph are relatively newly discovered and are still in an early stage of development. They will remain an active and popular area for researchers to explore. Some directions that are promising and worth further exploration are discussed below.

6.2.1 Scalability and Efficiency

Although many graph representation learning algorithms are scalable, when conducting graph generation, the time complexity of many algorithms are still constrained to $\mathcal{O}(N^2)$, where N is the number of nodes. It is relatively high and makes it hard to scale up to large graphs such as social networks. This constraint comes from the fact that most algorithms for graph generation model the procedure as sequential node generation along with the generation of corresponding edges between existing nodes. Therefore, by evaluating the connectivity of each pair of nodes, the number of generation steps is $\mathcal{O}(N^2)$. Right now, scalability and efficiency are one of the most important tasks for graph generation since most works that used graph neural networks (GNNs) are constrained to generate graphs with less than 100 nodes which severely blocked the applications on large graphs such as social networks. We believe that achievements in this area could highly improve the applications of graph generative models and thus make it possible to have a great impact in real life for tasks such as disease control and prevention.

6.2.2 Evaluation Metrics

It is known that evaluating generative models is very challenging. Normally the performance of a generative model is assessed based on the quality and diversity of the generated data, which make it difficult since it requires a relatively high-level understanding of the data compared to other traditional problems, such as classification and clustering. Moreover, this is even harder for graph generative models compared to generative models for images and text.

The reason is that graphs have a more complex data structure and it is difficult to measure likelihoods for models that relies on an ordering. Most current works (including ours) evaluate the performance of graph generative models by comparing the distributions of some graph statistics, such as degree distributions and clustering coefficient distributions between the real graphs and the generated ones. However, these metrics are constrained to local graph statistics and thus lack a global view of graph properties. Therefore, future work can be done in this area to help develop better evaluation metrics which can in turn provides a better understanding of the generative model and generated synthetic data.

6.2.3 Generalizing to Other Applications

Graph generation is a relatively new area in graph analysis. Right now its applications are mainly constrained to drug design. One reason is that there are not that much data in other domains available for this task. Most graph datasets are a single large network, such as social networks, airline networks, and economic networks. To apply deep learning models on these networks for graph generation, we need to create a large training dataset that contains massive subgraphs extracted from that one large network. Another reason is lack of interpretability compared to image and text data. For image and text generation,

there are various real world applications for entertainment. However, most graphs do not have this characteristic due to the complexity of its topological structure. Also, currently most work on graph generation focused on generating graphs randomly or conditioned on an input graph. However most graphs in the real world are dynamic and are changing over the time. Therefore, modeling the evolution of dynamic graphs over time is an really important goal and can have a great impact for tasks such as event forecasting and disease prevention.

Bibliography

[1] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008.

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 2002.

[3] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65, 1994.

[4] Sinan Aral, Lev Muchnik, and Arun Sundararajan. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proceedings of the National Academy of Sciences*, 106(51):21544–21549, 2009.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[6] Mustafa Bilgic, Galileo Mark Namata, and Lise Getoor. Combining collective classification and link prediction. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 381–386. IEEE, 2007.

[7] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*, 2018.

[8] Eran Borenstein and Shimon Ullman. Learning to segment. In *European conference on computer vision*, pages 315–328. Springer, 2004.

[9] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining*, pages 8–pp. IEEE, 2005.

[10] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *Intl. Conference on Learning Representations*, 2014.

[11] Visual Capitalist. Air traffic network map. https://www.visualcapitalist.com/air-traffic-network-map/airports-network/, 2019.

[12] Visual Capitalist. Who you dont know: Stanford economist examines how a weak social network can explain inequality, social immobility. https://news.stanford.edu/2019/03/06/human-networks-drive-inequality-social-immobility/, 2019.

[13] Jerome T Connor, R Douglas Martin, and Les E Atlas. Recurrent neural networks and robust time series prediction. *Neural Networks, IEEE Transactions on*, 5(2):240–254, 1994.

[14] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[15] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[16] Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE transactions on pattern analysis and machine intelligence*, 35(10):2454–2467, 2013.

[17] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, pages 2122–2159, 2011.

[18] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6: 290–297, 1959.

[19] Shuangfei Fan and Bert Huang. Training iterative collective classifiers with back-propagation. In *12th International workshop on mining and learning with graphs, San Francisco, USA*, 2016.

[20] Shuangfei Fan and Bert Huang. Training iterative collective classifiers with back-propagation. In *12th International Workshop on Mining and Learning with Graphs, San Francisco, USA*, 2016.

[21] Shuangfei Fan and Bert Huang. Recurrent collective classification. *arXiv preprint arXiv:1703.06514*, 2017.

[22] Shuangfei Fan and Bert Huang. Recurrent collective classification. *Knowledge and Information Systems*, pages 1–15, 2017.

[23] Shuangfei Fan and Bert Huang. Labeled graph generative adversarial networks. *arXiv preprint arXiv:1906.03220*, 2017.

[24] Shuangfei Fan and Bert Huang. Deep generative models for generating labeled graphs. *ICLR 2019 Workshop on Deep Generative Models for Highly Structured Data*, 2019.

[25] Shuangfei Fan and Bert Huang. Labeled graph generative adversarial networks. *arXiv preprint arXiv:1906.03220*, 2019.

[26] Shuangfei Fan and Bert Huang. Conditional labeled graph generation with GANs. *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[27] Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. Overview of the 2003 kdd cup. *Acm SIGKDD Explorations Newsletter*, 5(2):149–151, 2003.

[28] Lise Getoor and Christopher P Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[29] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, Edoardo M Airoldi, et al. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.

[30] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[32] Alex Graves. *Supervised sequence labelling*. Springer, 2012.

[33] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

[34] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13 (Mar):723–773, 2012.

[35] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein GANs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5767–5777, 2017.

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1024–1034, 2017.

[37] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[38] Masumi Ishikawa. Structural learning with forgetting. *Neural Networks*, 9(3):509–521, 1996.

[39] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–598, 2004.

[40] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecular optimization. In *Intl. Conf. on Learning Representations*, 2019.

[41] Fredrik Johansson, Vinay Jethava, Devdatt Dubhashi, and Chiranjib Bhattacharyya. Global graph kernels using geometric embeddings. In *Proc. of the Intl. Conf. on Machine Learning*, 2014.

[42] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

[43] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[44] Xiangnan Kong, Xiaoxiao Shi, and S Yu Philip. Multi-label collective classification. In *SDM*, volume 11, pages 618–629. SIAM, 2011.

[45] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and

VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proc. of the ACM Intl. Conf. on Web Search and Data Mining*, pages 333–341, 2018.

[46] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[47] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

[48] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, April 2016. URL https://www.microsoft.com/en-us/research/publication/gated-graph-sequence-neural-networks/.

[49] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[50] Jack Lindamood, Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. Inferring private information using social network data. In *Proceedings of the 18th international conference on World wide web*, pages 1145–1146. ACM, 2009.

[51] Ben London and Lise Getoor. Collective classification of network data. In Charu C. Aggarwal, editor, *Data Classification: Algorithms and Applications*. CRC Press, 2013.

[52] Joseph Loscalzo, Isaac Kohane, and Albert-Laszlo Barabasi. Human disease classification in the postgenomic era: a complex systems approach to human pathobiology. *Molecular systems biology*, 3(1), 2007.

[53] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[54] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Neural Information Processing Systems*, 2012.

[55] Luke K McDowell, Kalyan Moy Gupta, and David W Aha. Cautious inference in collective classification. In *Conference on Artificial Intelligence (AAAI)*, volume 7, pages 596–601, 2007.

[56] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, pages 415–444, 2001.

[57] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

[58] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784, 2014.

[59] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[60] John Moore and Jennifer Neville. Deep collective inference. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[61] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[62] Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[63] Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[64] Jennifer Neville and David Jensen. Collective classification with relational dependency networks. In *Proceedings of the Second International Workshop on Multi-Relational Data Mining*, pages 77–91. Citeseer, 2003.

[65] Christos A Nicolaou, Joannis Apostolakis, and Costas S Pattichis. De novo drug design using multiobjective evolutionary graphs. *Journal of Chemical Information and Modeling*, 49(2):295–307, 2009.

[66] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. *arXiv preprint arXiv:1610.09585*, 2016.

[67] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In *Proc. of theIntl. Conf. on Machine Learning*, pages 2642–2651, 2017.

[68] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. *arXiv preprint arXiv:1609.04508*, 2016.

[69] Creative Proteomics. Protein-protein interaction networks. https://www.creative-proteomics.com/services/protein-protein-interaction-networks.htm, 2019.

[70] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p*) models for social networks. *Social networks*, 29(2):173–191, 2007.

[71] Hiroto Saigo, Sebastian Nowozin, Tadashi Kadowaki, Taku Kudo, and Koji Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75(1):69–89, 2009.

[72] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2234–2242, 2016.

[73] Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

[74] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

[75] Frank Schweitzer, Giorgio Fagiolo, Didier Sornette, Fernando Vega-Redondo, Alessandro Vespignani, and Douglas R White. Economic networks: The new challenges. *science*, 325(5939):422–425, 2009.

[76] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[77] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intell. and Stat.*, pages 488–495, 2009.

[78] Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

[79] Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. New specifications for exponential random graph models. *Sociological Methodology*, 36(1): 99–153, 2006.

[80] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning (ICML)*, pages 129–136, 2011.

[81] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

[82] Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 485–492. Morgan Kaufmann Publishers Inc., 2002.

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[84] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[85] Saurabh Verma. Machine learning equations. https://vermamachinelearning.github.io/, 2017.

[86] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proc. of the Workshop on Online Social Networks*, pages 37–42, 2009.

[87] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440, 1998.

[88] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of Wasserstein GANs: A consistency term and its dual effect. In *ICLR*. OpenReview.net, 2018.

[89] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[90] R. Xiang and J. Neville. Relational learning with one network: An asymptotic analysis. In *International Conference on Artificial Intelligence & Statistics (AISTATS)*, 2011.

[91] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018.

[92] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018.

[93] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Conference on Artificial Intelligence (AAAI)*, pages 2852–2858, 2017.

[94] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. StackGAN: Text to photo-realistic image synthesis with

stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

[95] Elena Zheleva, Evimaria Terzi, and Lise Getoor. Privacy in social networks. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(1):1–85, 2012.