An Investigation of Attention Mechanisms in Graph Convolutional Networks applied to Link Prediction Problems

by

Rui Li

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements for the degree of Master of Computer Science with Specialization in Data Science

Department of Computer Science
Carleton University
Ottawa, Ontario, Canada
August 2020

Copyright ©
2020 - Rui Li
The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the Thesis

An Investigation of Attention Mechanisms in Graph
Convolutional Networks applied to Link Prediction Problems

Submitted by Rui Li
in partial fulfilment of the requirements for the degree of
Master of Computer Science

Dr. Anthony White, Supervisor

Carleton University
2020
Abstract

The link prediction problem is fundamental to many application domains. Recently, deep learning-based models have been proposed to tackle this kind of problem. Graph auto-encoder (GAE) is a framework for unsupervised learning on graph-structured data. By using a graph convolutional network (GCN) encoder and a simple inner product decoder, GAE achieves competitive results in link prediction tasks on citation networks. Another important problem on graph-structured data is node classification. Graph attention mechanism has been shown to have good performance in these tasks.

This research investigates whether graph attention mechanisms can achieve good performance in link prediction tasks. We propose the attentive graph auto-encoder (AGAE) model, which incorporates GAE with the graph attention mechanism. The model is compared with GAE on both real-world citation networks and synthetic datasets. Empirical analysis of the AGAE components is included in this research. Investigations on how the model performs on networks with different characteristics is also included. In general, AGAE achieves competitive performance with GAE on citation networks while it outperforms GAE on certain synthetic networks.
Acknowledgments

I, Rui Li, would like to thank my supervisor, Professor Tony White, for his advice, patience, and friendly discussions throughout my thesis. His expertise was invaluable in formulating the research topic and methodology in particular. I would like to thank my parents for their tremendous support and encouragement. I would like to thank all the members of Carleton’s Complex Adaptive System Lab for their kindness and help throughout this journey of my graduate study.
Table of Contents

Abstract iii

Acknowledgments iv

Table of Contents v

List of Tables viii

List of Figures ix

Nomenclature xi

1 Introduction 1
   1.1 Thesis Objectives ........................................ 2
   1.2 Problem Statement ........................................ 2
   1.3 Motivation .............................................. 2
   1.4 Summary of Contributions ................................. 3
   1.5 Thesis Organization ...................................... 4

2 Background 5
   2.1 Network Embedding ....................................... 5
   2.2 Neural Networks Basics .................................. 8
      2.2.1 Artificial Neural Networks ......................... 8
# List of Tables

1. Summary of real-world datasets. ........................................ 45
2. Compare the performance of GAE and AGAE on Cora. .............. 51
3. Compare the performance of GAE and AGAE on Citeseer. .......... 51
4. Compare multi-headed attention with single-headed attention. ... 56
5. Parameter settings for the synthetic data generator. ............... 59
6. Evaluation of GAE and AGAE on synthetic networks. ............. 59
7. Parameter settings for generating the small-world networks. .... 64
8. Evaluation of GAE and AGAE on small-world networks. .......... 64
List of Figures

1. Application of DeepWalk on a small social network to generate embedding in $\mathbb{R}^2$ [1]. .................................................. 7
2. Example of a multilayer perceptron with three layers. .............. 8
3. Example of an artificial neuron. ........................................... 9
5. Illustration of the convolution operation. ............................... 12
6. An example of the traditional autoencoder. ............................. 14
7. Attention mechanism. ....................................................... 16
8. Self-attention ................................................................. 17
9. Comparing 2D convolution with graph convolution. ............... 19
10. Illustration of how a GAT layer works. ................................ 25
11. Aggregation process in an encoder layer. .............................. 37
12. An illustration of multi-headed attention. .............................. 39
13. AGAE architecture .......................................................... 40
14. Example ROC Curve. ....................................................... 48
15. Example precision-recall curve. ........................................... 49
16. First encoder layer attention coefficients vs the non-parametric coefficients. ................................................. 52
17. Second encoder layer attention coefficients and the non-parametric coefficients. .............................................. 53
<table>
<thead>
<tr>
<th></th>
<th>18 Compare input feature with the attentive encoder output.</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>19 Compare attentive encoder output with the non-attentive encoder output.</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>20 Degree distribution of a network generated by our implementation of preferential attachment.</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>21 Evaluation of GAE and AGAE on scale-free networks with different link density.</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>22 Degree distribution of a network generated by Algorithm 3 with 500 nodes and ( m = 100 ).</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>23 Example small-world network with 10 communities and 20 nodes in each community.</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>24 Evaluation of GAE and AGAE against different topological randomness levels.</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>25 Evaluation of GAE and AGAE against attribute noise levels.</td>
<td>67</td>
</tr>
</tbody>
</table>
Nomenclature

Abbreviations

This thesis uses some common abbreviations existing in the domain of machine learning. The following table lists the abbreviations and their expansion.

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>Graph Convolutional Network</td>
</tr>
<tr>
<td>GAT</td>
<td>Graph Attention Network</td>
</tr>
<tr>
<td>GAE</td>
<td>Graph Autoencoder</td>
</tr>
<tr>
<td>AGAE</td>
<td>Attentive Graph Autoencoder</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-layer Perceptron</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The graph attention mechanism has shown promising performance in node classification tasks on graphs. The graph neural network model which implements the graph attention mechanism is called GAT [3]. GAT extends Graph Convolutional Neural Networks (GCN) by enabling implicit weight assignment, meaning that the weights between a node and its neighbourhood are implicitly captured via attention mechanism during training. GAT outperforms GCN in the graph node classification task on citation networks. On the other hand, GAE [4] provides a framework for link prediction tasks on graphs. The encoder of GAE is a GCN and the decoder of GAE is simply the inner product. GAE can be trained end-to-end on graph-structured data to predict missing edges of the same graph or some potential edges of the graph in the future.

The goal of this research is to explore if the link prediction tasks can also benefit from the graph attention mechanism. We propose the AGAE model for link prediction tasks on attributed graph-structured data. AGAE is built on top of GAT and GAE. It simply replaces the GCN encoder layers in GAE by the GAT layers and keeps the inner product decoder unchanged. The performance of AGAE is compared with GAE on both real-world citation networks and synthetic networks. On the citation networks, extensive analyses are performed on the trained AGAE models.
These analyses include the distribution of learnt edge weights, encoder outputs, and the effectiveness of multi-headed attention. Multi-headed attention is basically the aggregation of multiple independent single-headed attention layers to form a single layer. On synthetic networks, we have control over the network generation algorithms so that the model can be evaluated on networks with different characteristics. We vary one network characteristic at a time to see how the performance of the model changes. These network characteristics include the type of network model, topological randomness level, attribute randomness level, and link density. The detail of these analyses and results are described in Chapter 5.

1.1 Thesis Objectives

This thesis has two primary objectives. The first objective is to propose a model with the graph attention mechanism implemented for link prediction problems. The second objective is to analyze the trained model itself as well as how the model performs on the networks with different characteristics.

1.2 Problem Statement

Can the graph attention mechanism learn good representations for link prediction tasks and how well does it work on networks with different characteristics?

1.3 Motivation

To answer the problem statement and achieve the thesis objectives, several relevant questions must be answered. The list below summarizes these key questions:

- RQ1: How can the graph attention mechanism be modelled for the graph link
prediction tasks?

- RQ2: Does the proposed attentive model work better than the non-attentive model on link prediction tasks?

- RQ3: How does the model perform on networks with certain characteristics?

To answer the first question, we propose the AGAE model with the graph attention mechanism implemented. To answer the second question, we compared the performance of GAE and AGAE on both citation networks and synthetic networks. To answer the third question, we implemented an algorithm that can generate synthetic network data with specific properties. We vary one property at a time to observe how the performance of the model changes.

### 1.4 Summary of Contributions

This thesis makes the following contributions:

- Proposal and implementation of the AGAE model, which has the graph attention mechanism implemented. (Section 4.3)

- Evaluation of AGAE on real-world citation networks. Analysis of the attentive encoder outputs, the attention coefficients distributions, and the effectiveness of multi-headed attention. (Section 5.4)

- Investigation of the use of synthetic network data to evaluate the model. Identification of effects that certain network characteristics have on the performance of AGAE.
1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides the introductory background material to facilitate understanding of the rest of this thesis. It covers the topics of network embedding, neural network basics, convolutional neural networks, and conventional autoencoders. Chapter 3 discusses existing research within the relevant areas. Topics covered in this chapter are attention mechanism, graph convolution networks, attentive models in graphs, and the link prediction problem in graphs. Chapter 4 describes the proposed AGAE model and its vectorized implementation. This is followed by Chapter 5, which details experiment settings, results, and analyses. Finally, Chapter 6 summarizes the significant results carried out in this research. It also discusses the potential applications and future research directions that relate to the topic of this thesis.
Chapter 2

Background

This chapter provides the background knowledge which is needed to understand the rest of this thesis. Section 2.1 gives a high level description of network embedding. Section 2.2 introduces basic concepts of neural networks, including multi-layer perceptrons, the backpropagation training algorithm, and the dropout regularization technique. Section 2.3 gives an introduction to convolutional neural networks, emphasizing the importance of the convolution operation. Section 2.4 briefly discusses recurrent neural networks. Section 2.5 introduces the traditional autoencoder and the variational autoencoder. Those who are familiar with some or all of these topics can skip the corresponding sections without loss of understanding of the research described in subsequent chapters.

2.1 Network Embedding

Networks can be used to model many real-world data structures such as the World Wide Web, social networks, and communication networks. The purpose of network embedding is to learn low-dimensional latent representations of nodes in a network. These representations can then be used for solving downstream machine learning problems, which include node classification and link prediction.
Network embeddings should reflect the similarity between each pair of nodes in a network. In other words, the distance between embeddings should represent a metric for evaluating the similarity between the corresponding members of the network. In addition to this key characteristic, a good network embedding should also possess the following characteristics according to the work of [1]:

- **Adaptability** - The learned embeddings should work for networks that are constantly evolving without repeating the learning process all over again.

- **Scalability** - The embeddings should be generated in a reasonable period of time for large-scale networks.

- **Low dimensionality** - The embeddings should be low-dimensional since when labelled data is scarce; low-dimensional models generalize better and converge faster.

Traditionally, dimensionality reduction techniques such as principal component analysis [6] are often used to generate network embeddings. These dimensionality reduction techniques can represent an $n \times m$ matrix $M$ as a $n \times k$ matrix where $k << n$. In the case of network embedding, $M$ should be a matrix (e.g., adjacency matrix and normalized Laplacian matrix) that can represent the structural information of the network. The dimensionality reduction-based graph embedding methods are simple but usually have poor performance.

Recently, deep learning-based network embeddings methods have drawn a lot of research attention. DeepWalk [7] is one of these network embedding methods from the deep learning community. The idea of Deepwalk is similar to the word embedding technique in natural language processing. The nodes can be thought as words and a short random walk from each node can be thought as a sentence. The neural language model skip-gram is then applied on these random walks to obtain the
Figure 1: Application of DeepWalk on a small social network to generate embedding in $\mathbb{R}^2$ [1].

network embedding. Due to its performance and scalability, Deepwalk has become arguably the most popular network embedding method [1].

An example of applying RandomWalk on a small social network Zachary’s Karate graph [8] is illustrated in Figure 1. Figure 1a is the network with color-coded labels and Figure 1b is the embedding of the network into the $\mathbb{R}^2$ space by DeepWalk. It can be seen from Figure 1b that the similarities between nodes are preserved in the embedding given the assumption that nodes with the same label are similar to each others. In essence, DeepWalk has computed identifiable clusters for the network nodes.

The aforementioned methods only make use of network structural information to obtain the network embedding. However, nodes in real-world networks are often associated, or tagged, with attributes. Some graph neural network models such as GCN can simultaneously process both network structural information as well as nodes’ attributes to generate embedding. We will discuss more on GCN and its variants in section 3.2. A more detailed introduction to the topic of network embedding in general can be found in [1].
2.2 Neural Networks Basics

2.2.1 Artificial Neural Networks

Artificial neural networks are computation models inspired by animal brains. The building blocks of an artificial neural network are the artificial neurons. Each neuron performs some simple computations as part of a large computation performed by the network as a whole. The most basic artificial neural network is the multi-layer perceptron, where the neurons are organized into layers as shown in Figure 2. The first layer of an multi-layer perceptron (MLP) is the input layer. It takes input values and passes them to the hidden layer. There could be one or more hidden layers that process the input data and present the result to the output layer. In a multi-layer perceptron, the network is feed-forward, meaning that signals always travel from one layer to a subsequent layer.

Each artificial neuron has weights associated with all the artificial neurons in the previous layer. An artificial neuron computes the weighted sum of outputs from
the previous layer and applies a non-linear activation function to that sum before outputting the result. Figure 3 shows an example of an artificial neuron. In this example, the output $y$ of the neuron is computed as:

$$y = \sigma(x_1w_1 + x_2w_2 + x_3w_3),$$

where $\sigma$ is the non-linear activation function.

Rectified Linear Unit function (ReLU) and LeakyReLU [9] are two non-linear activation functions that work well with neural networks. The ReLU is defined as $\text{ReLU}(z) = \max(0, z)$. It has the advantage of being fast to compute in practice. However, ReLU suffers from the problem that during training, some neurons stop outputting anything other than 0. This could happen since when the weighted sum of a neuron is negative, ReLU outputs 0. LeakyReLU is a variant of ReLU that solves this problem. It is defined as $\text{LeakyReLU}(z) = \max(\alpha z, z)$. The hyperparameter $\alpha$ is the slope of the function for $z < 0$ and is usually set to 0.01. This small slope ensures that neurons do not always output 0.
2.2.2 Back-propagation

Back-propagation \cite{10} is an efficient training algorithm for neural networks. Each step of back propagation consists of a forward pass and a backward pass. The forward pass is the same as making a prediction except all the intermediate results are preserved as they are needed for the backward pass. During a backward pass, the algorithm first measures the network’s output error. Then it computes the error gradients (e.g., how much each weight of the output layer contributes to the output error) by the chain rule and recursively repeats this through hidden layers until it reaches the input layer. Finally, it uses the error gradients to update all the weights in the model.

It is worth noting that ensuring all the weights are initialized randomly is important when using back-propagation for training. Otherwise, the training may fail since back-propagation will update weights the same way and they will remain the same to each other. This is often referred to as the vanishing gradients problem. The Glorot initialization is often used for randomly initializing network weights \cite{11}. It can significantly alleviate the vanishing gradients problem.

2.2.3 Dropout

Dropout \cite{2} is a powerful regularization technique for deep neural network. It can make the network generalize more to unseen data. While training a neural network with dropout, every node except for the nodes in the output layer has a probability \( p \) of being dropped temporarily at each training step. Figure 4 illustrates how the nodes along with all their incoming and outgoing connections are dropped out. The parameter \( p \) is called the dropout rate, which is usually set to around 50\%. Note that the dropout only happens during training. After the model is trained, the standard network without dropout is used for inference.

Dropout forces each neuron not to co-adapt with their neighbouring neurons so
that each neuron has to be as useful as possible on itself. Dropout can also make
the network not to rely excessively on just a few input neurons so that the network
becomes less sensitive to slight changes in the inputs. Therefore, deep neural networks
usually generalize better by applying the dropout technique.

2.3 Convolutional Neural Network

Convolutional neural networks (CNNs) are a specialized kind of neural networks de-
signed for processing grid-like data [12]. The basic building block of a CNN is the
convolutional layer. The convolutional layer employs a mathematical operation called
convolution, which slides one function over another and measures the integral of their
pointwise multiplication. In the context of a convolutional layer, one function is the
input and the other function is the kernel. We slide the kernel over input to compute
the output feature map. Figure 5 illustrates how convolution works on 2-dimensional
input [13]. In this case, the input is a $3 \times 4$ matrix and the kernel size is $2 \times 2$.
Starting form the top-left location, we keep moving the kernel right until it reaches
the right border of the input matrix. Then, we return the kernel to the left border,
Figure 5: Illustration of the convolution operation.

and move the kernel down by one element. We repeat this process for every possible location on the input matrix until the kernel reaches the bottom-right corner of the input matrix. For each location, we can compute the element-wise product of the kernel and the overlapped part of the input matrix. For instance, the convolution result at the top-left location is: $1 \times 1 + 1 \times 4 + 1 \times 2 + 1 \times 5 = 12$. The output feature map consists of convolution results of all possible location as shown in Figure 5b.

Unlike the multi-layer perceptron, each neuron in a convolutional layer is only connected to neurons in its receptive field. This is usually referred to as sparse connectivity. Sparse connectivity means the model has far fewer parameters and the efficiency of the model is improved. In a convolutional layer, each kernel is used at every position of the input. This is called parameter sharing. Parameter sharing allows CNN to capture spatially invariant features. For instance, when processing image, the same edges appear more or less everywhere in the image. Therefore, it is practical to share parameters across the entire image [14]. There are also some other concepts in CNNs such as the pooling operation. We mainly introduce the convolution operation in this section since it is the key concept to CNN and helps to understand the graph convolution in GCN. A comprehensive discussion of CNNs can be found in Chapter 9 of [12].
2.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are proposed for processing sequential data \[10\]. Just as CNNs are specialized for processing images of large size, RNNs can scale to much longer sequences than other networks without sequence-based specialization. A RNN layer is composed of recurrent neurons. At each time step $t$, every neuron receives the input from time $t$ and the output from time $t - 1$. Therefore, each recurrent neuron has two sets of weights. One set of weights is for the inputs and the other set is for the outputs of the previous time step. These weights are shared across several time steps. RNNs are often used in natural language processing tasks. For instance, it can be used to build an encoder-decoder architecture (e.g., seq2seq model \[15\]), which is capable of performing neural machine translation.

2.5 Autoencoder

An autoencoder is the neural network that can learn meaningful embedding of input data in an unsupervised manner \[14\]. It is trained by simply learning to copy its inputs to its outputs. However, it will be trivial if the input is just perfectly copied to the output. Instead, the autoencoders are often restricted to copy only approximately. These restrictions can force the model to learn efficient ways of representing data.

The most straightforward restriction we can impose on an autoencoder is to limit the dimensionality of the embedding. The learned embedding should have much lower dimensionality than the input data. Therefore, the autoencoder is often used as a useful dimensionality reduction method for embedding input data. The traditional autoencoder contains an encoder and a decoder. The encoder and decoder are often neural networks with symmetrical architectures. Figure 6 shows an example architecture of a traditional autoencoder. We can see from the figure that the encoder takes
Figure 6: An example of the traditional autoencoder.

The input data $X$ and converts it to a lower-dimensional embedding $Z$. The decoder takes the lower-dimensional embedding $Z$ and returns the reconstruction of the input $\hat{X}$. There will be an encoding error associated with the difference between $X$ and $\hat{X}$.

Another important type of autoencoder is the variational autoencoder [16]. The difference between the variational autoencoder and the traditional autoencoder resides at the encoder. Instead of directly generating an embedding, the encoder of VAE produces a mean embedding $\mu$ and a standard deviation $\sigma$. The actual embedding is then sampled randomly from a Gaussian distribution with mean $\mu$ and standard deviation $\sigma$. After the actual embedding is sampled, the decoder just decodes the embedding normally. Since the outputs of the variational autoencoder is partly determined by chance, the variational autoencoder can act as a generative model to generate new data that are similar to the data in the original dataset.
Chapter 3

Related Work

This chapter provides an overview of recent research literature that relate to the application of the graph attention mechanism to the link prediction problem. Section 3.1 discusses the attention mechanism in general. Section 3.2 introduces the graph convolutional network and its variants. Section 3.3 discusses the graph models that incorporate the attention mechanism. The GAT is described in more detail since the method proposed in Chapter 4 is mainly based on this model. Section 3.4 discusses the link prediction problem in networks. It first gives a formal definition of the link prediction problem in networks. Then it introduces several link prediction methods, which are categorized into local similarity-based methods, global similarity-based methods, and Supervised learning methods.

3.1 Attention Mechanism

The attention mechanism has shown promising performance within diverse application domains especially in Natural Language Processing [17]. Before the attention mechanism was proposed, the seq2seq model [15] is commonly used in the field of language modeling. It has an encoder-decoder architecture that can transform arbitrary length input sequence to an arbitrary length output sequence. Both encoder and
The encoder compresses the input sequence into a fixed length latent embedding vector called the context vector and then the decoder transforms the context vector into a sequential output. The most apparent drawback of this seq2seq model is that the fixed length context vector is incapable of remembering a very long sequence. The attention mechanism was developed in order to resolve this problem.

The attention mechanism was first introduced for machine translation in [19] which makes improvements to the Sequence to Sequence (seq2seq) model. As shown in Figure 7, instead of building a context vector solely based on the encoder’s last hidden state, the attention mechanism creates connections between the entire input sequence and the context vector. The weights of these connections can be learned jointly with the rest of the model by training. During inference, the context vector is computed as the weighted sum of encoder’s hidden states of all time steps so that we do not need to worry about forgetting anymore.

Another impactful attention model is the transformer model which is proposed in [20]. It can do sequence-to-sequence modeling without the recurrent architecture...
Figure 8: Self-attention

and significantly improved state of the art in neural machine translation. The attention mechanism used in the transformer model is called multi-headed self-attention. Self-attention, as the name suggests, relates different positions of a single sequence to compute a representation of the same sequence. As the model processes each word in the input sequence, self-attention allows it to look at other positions in the input sequence as shown in Figure 8. Instead of only computing the attention once, the multi-headed mechanism runs through the attention computation multiple times independently and the outputs are simply concatenated and linearly transformed into the expected dimension. Empirically, the multi-headed mechanism allows the model to learn better attention weights and to achieve better performance. Because the transformer model does not rely on RNN anymore, the idea can be easily applied to data structures other than a sequence. For instance, on graph-structured data, the self-attention strategy can be used to compute the embedding of each node by attending over its neighbours. Also, the model is more efficient in training since it can be better parallelized without the recurrent architecture.

Since the Transformer was introduced, it has become the building block of many state-of-the-art NLP architectures. In the Generative Pre-Training (GPT) paper [21], the authors demonstrated the effectiveness of performing unsupervised pre-training on a Transformer-like architecture. The architecture is composed of 12 Transformer
modules and is trained on a large dataset in an unsupervised manner. Then the pre-trained model can be fine-tuned on various downstream tasks such as text classification, textual entailment, and question answering. The model achieves significant improvement when applied to these NLP tasks. The follow-up work GPT-2 model has a similar architecture to GPT but with larger number of parameters [22]. The most important innovation of this model is that it can achieve good performance on many NLP tasks without any fine-tuning. In addition to NLP, the attention mechanism can also be applied to other machine learning problems. In computer vision, attention mechanism can be applied to images to generate captions [23]. The image is first passed through a CNN to extract feature maps. Then a LSTM decoder consumes those feature maps one by one to produce descriptive words, where the attentive weights from all the feature maps to each word are learned through attention.

3.2 Graph Convolutional Networks

Many real-world data can be naturally modelled as graph including the World Wide Web, social networks, scientific citation networks, urban computing, recommender systems, epidemiology, and anomaly and fraud analysis [24]. For instance, in a social network, the users can be represented as nodes and their interactions can be denoted as links [25]. In a citation network, documents can be represented as nodes and a citation from one document to another can be represented as an edge.

In recent years, deep learning has achieved state-of-the-art in many machine learning tasks such as image classification, speech recognition, and machine translation. The conventional deep neural networks require input data to be represented in Euclidean space. Therefore, it can not process graph-structured data directly. Recently, many studies that managed to extend deep learning approaches for graph data have emerged, and the term graph neural networks is used for all these deep learning
approaches. Early proposed Graph Neural Network models \cite{26,28} propagate information via the recurrent neural architecture in an iterative manner until it reaches a stable state. However, these methods are computationally expensive.

Since CNN has achieved huge success in computer vision, researchers generalize the convolution operation for graph-structured data and name it graph convolution. Graph neural network models based on graph convolution are called graph convolutional networks. Figure 9 compares graph convolution with the traditional 2D convolution \cite{29}. In this 2D convolution example, the new representation of the red node is computed by taking a weighted sum of the red node and its neighbours. Similarly, in the graph convolution example, the new representation of the red node can also be computed as the weighted sum of the red node and its neighbours. Unlike grid-like data, the neighbours of the node in a graph are unordered and variable in size. The graph convolution provides a way to aggregate each node and its neighbours in the graph-structured data.

\cite{30} proposed the first graph convolutional network based on the spectral graph theory. It uses graph Fourier transformation to transform the graph signal to its spectral domain, and then perform the graph convolution in the spectral domain. However, this method is computation-intensive since it requires the graph Fourier
transformation and the inverse graph Fourier transformation. ChebNet [31] was then proposed to simplify the computation. It uses Chebyshev polynomials to approximate the spectral graph convolution operation. Essentially, this technique is equivalent to use polynomials to provide a series approximation to an arbitrary function. ChebNet implicitly avoids the computation of the graph Fourier transformation, reducing the computation complexity. [32] introduces the first-order approximation of ChebNet (1stChebNet) which further simplifies the spectral graph convolution. Due to its simplicity and impressive performance, the 1stChebNet has been referred to as GCN in many literature. It is also used as a strong baseline in many research experiments. In the following of this thesis, GCN will refer to the graph convolution networks model proposed in [32].

GCN can be understood from a message passing perspective. Given a node $v$ in a graph, the GCN layer first aggregate $v$ and its neighbours to produce an intermediate representation $h_v$. Then, the aggregated representation $h_v$ is transformed with a linear projection followed by a non-linearity as $\sigma(Wh_v)$. Therefore, the layer-wise propagation rule of GCN can be written as:

$$X^{k+1} = \sigma(\hat{A}X^kW^k)$$ (2)

Here, $X^k$ is the node feature matrix. $\hat{A}$ is the symmetrically normalized adjacency matrix of the input graph. $\hat{A}$ also includes self-loops for each node so that the aggregated representation of a node includes its own features. $\hat{A}$ can be computed as: $\hat{A} = I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. $W^k$ is the trainable weights, and $\sigma$ is a nonlinear activation function. $X^{k+1}$ is the output node feature matrix.

Elements in the normalized adjacency matrix $\hat{A}$ can also be viewed as the edge weights of the input graph. These weights determine the importance between a given node and its neighbouring nodes during the message passing process. Without the
symmetrical normalization, nodes with large degrees will get large values in their representation which can lead to issues such as exploding gradients during training. However, symmetrical normalization is not the only way to compute these edge weights. These weights can also be implicitly learned by training. The next section will give an introduction to the graph attention mechanism, which is another way to compute the edge weights.

GCN is the fundamental building block of many complex graph neural network models. GCN and its variants can be used for different graph analysis tasks, which can be classified into following three categories [29]:

- **Node-level classification** The graph convolutional layers in a GCN model can process graph data directly and output the graph embedding. It is usually followed by a MLP or a simple softmax layer to predict the final label of the nodes. Even if in the semi-supervised learning setting (e.g., partial nodes are labelled while others remain unlabeled.), GCN can learn a robust model that effectively identify the unlabeled nodes.

- **Edge-level classification** This kind of task relates to the prediction of edge labels. In the link prediction settings, the problem is reduced to a binary classification to predict the existence of an edge between two nodes. After the graph embedding is computed by the graph convolutional layers, to predict the label of an edge, an additional function will take two nodes’ latent representation and output the label.

- **Graph-level classification** The idea of graph-level classification is similar to the image classification tasks. In order to obtain a compact representation on graph level, a sum or average pooling layer following each graph convolutional layer is used to coarsen the graph so that nodes on the coarsened graph represent higher graph-level representations. To calculate the probability for each graph
label, the final layer is usually a linear layer with softmax activation.

GCN achieves good performance in the semi-supervised node classification tasks on several citation network datasets. However, GCN has to be trained with all nodes in the graph. Therefore, it is inherently transductive and does not work for dynamic graphs. In reality, we often need to add new nodes to an existing graph. This is quite common in social networks, recommendation systems, and citation networks. The GraphSAGE \cite{33} addresses this problem by introducing a general inductive framework that efficiently generates embeddings for a new node by sampling and aggregating features from the node’s local neighbourhood. On the other hand, GAT makes an improvement to GCN by leveraging masked self-attention mechanism. It outperforms the GCN in the transductive learning setting and achieves state-of-the-art in the inductive learning setting \cite{3}. More discussions on GAT is included in Section 3.3. Graph auto-encoder (GAE) and its variants are often used in edge level classification tasks. In a typical GAE, the encoder transforms attributed input graph into low-dimensional embedding, and the decoder attempts to reconstruct the structure of the graph based on the embedding. More discussions on GAE is included in Section 3.4.

In addition to GCN and its variants already mentioned, there are some other graph convolutional models that are inspired by existing deep learning models. Molecular Generative Adversarial Networks adapts generative adversarial networks \cite{34} to work directly on graph-structured data. It can be used to discover new chemical compound which possess certain chemical and physical properties. Experiments on QM9 chemical database show that this model is capable of generating close to 100% valid compounds \cite{35}. Graph spatial-temporal networks is an other variants of GCN that can be used to forecasting future node values. The application domain of the graph spatial-temporal networks including neuroscience, human action recognition, and traffic forecasting. Diffusion convolutional recurrent neural network (DCRNN) \cite{36} is a graph
spatial-temporal model proposed for traffic forecasting by incorporating both spatial and temporal dependency in the traffic flow. It captures spatial dependency using graph convolution and temporal dependency using a recurrent architecture. DCRNN have achieved significant improvement over state-of-the-art baselines on large scale road network traffic datasets.

### 3.3 Attentive Models in Graphs

As we discussed in the Section 3.1, attention mechanisms have become the state-of-the-art in natural language processing tasks. Recently, many new approaches have been proposed to incorporate the attention mechanism into graph models. Graphs that are extracted from the real-world data can be structurally large and noisy. The attention mechanism can help the model to focus on the most relevant parts of the input graph. Therefore, it has the potential to allow the model to avoid noisy parts of the graph, thus improving the signal-to-noise ratio [37]. Furthermore, analyzing the learned attention weights may also provide a way to make the results more interpretable [38].

Although there are several types of graph attention mechanism, they all share the same purpose. In general, given a target graph object $v_0$ and a set of graph objects in $v_0$’s neighbourhood $N_{v_0}$, graph attention is defined as a function $f : \{v_0\} \times N_{v_0} \rightarrow [0, 1]$ that maps each of the graph objects in $N_{v_0}$ to a relevance score [24]. The graph object mentioned here can be node, edge, or sub-graph. All types of graph attention mechanisms differ in how the attention function $f$ is defined.

Graph attention network (GAT) [3] extends GCN by leveraging an explicit self-attention mechanism. The key difference between GAT and GCN is how the information from the one-hop neighbourhood is aggregated. Instead of producing the
normalized sum of neighbouring node features, GAT introduces the attention mechanism as a substitute for the normalized convolution operation. The importance of a node to another node is determined jointly with other parameters during training.

Suppose $i$ and $j$ are two nodes in a graph. $h_i$ and $h_j$ are the feature vectors of the two nodes. Each feature vector represents attributes of the corresponding node. The intention here is to find a function that can determine how important is $j$ to $i$ based on their feature vectors $h_j$ and $h_i$. The importance score of $j$ to $i$ is computed as:

$$e_{i,j} = \text{LeakyReLU}(a^T [Wh_i || Wh_j])$$  \hspace{1cm} (3)

where $W$ is the weight matrix shared over all the nodes in the graph. $a$ is the learnt attention weights vector. The first step performed by the graph attention mechanism is to linearly transform the feature vectors by $W$. Then, the two transformed feature vectors are concatenated and mapped by $a^T$ to a single attention coefficient score $e_{i,j}$ which tells how important node $j$ to node $i$.

In its most general form, every node can attend on every other node, dropping all structural information. However, in GAT, the graph structure is injected into the attention mechanism by performing masked attention. In other words, it only computes $e_{ij}$ for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ are the 1st order neighbours of node $i$. The attention coefficients are then normalized using the softmax function:

$$a_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$  \hspace{1cm} (4)

The reason to normalize the attention coefficients is due to the varied structure of graphs. Each node in a graph can have highly variable node degrees. Normalization allows all the attention coefficients to have a common scaling of 0 to 1.

Once the attention coefficients of node $i$ and its neighbours are obtained, they are
Figure 10: Illustration of how a GAT layer works.

used to compute the new representation $h'_i$ of $i$ as:

$$h'_i = \sigma\left(\sum_{j \in N_i} a_{ij} Wh_j\right),$$  \hspace{1cm} (5)$$

where $\sigma$ is a non-linear activation function. $h'_i$ is the aggregated representation of $i$’s neighbours weighted by the attention coefficients.

Figure 10 gives an example to illustrate how a GAT layer works. To compute the new representation $h'_i$ of node $i$, it first uses Equation 3 and 4 to compute the attention coefficients between $i$ and its neighbours, which in this example are $a_{i,k}$, $a_{i,l}$, $a_{i,m}$, and $a_{i,n}$. Then, Equation 5 is applied to compute $h'_i$. The new representations of other nodes in the graph are computed the same way. According to the experimental results, GAT has significant better performance than GCN on the node classification task on citation networks.

Attention-based Graph Neural Network (AGNN) \cite{39} proposes a linear graph neural network model that removes all the intermediate fully-connected layers, and replaces the propagation layers with the attention mechanism. AGNN differs from GAT
in two ways. First, AGNN does not have intermediate fully connected layers except for the first layer. The first layer of AGNN is a typical dense layer which transforms the input node features into \( n \)-dimensional representations:

\[
H^{(1)} = ReLU(XW^{(0)}).
\]  

(6)

This is followed by layers of attention-guided propagation layers:

\[
H^{(t+1)} = P^{(t)}H^{(t)},
\]

(7)

where \( P^{(t)} \in \mathbb{R}^{n \times n} \) is the propagation matrix containing attention coefficients. The propagation layer in Equation 7 significantly reduces the number of trainable parameters, which arguably makes the model more efficient and provides better generalizability. Second, the attention coefficients of AGNN are computed based on the cosine similarity comparing to the attention coefficients of GAT which is based on the dot product. Given a node \( i \) and its neighbouring node \( j \), the attention coefficient \( a_{ij} \) that represents the relevance of node \( j \) to node \( i \) is computed as:

\[
a_{i,j} = \frac{\exp(\beta \cdot \cos(Wh_i, Wh_j))}{\sum_{k \in \mathcal{N}_i} \exp(\beta \cdot \cos(Wh_i, Wh_k))}
\]

(8)

where \( \beta \) is a scalar trainable bias. Similar to GAT, \( W \) is a trainable weight matrix and \( \mathcal{N}_i \) is the set of neighbouring nodes of node \( i \).

In conclusion, AGNN significantly reduces the model complexity by removing the non-linear layers and uses a cosine similarity based attention mechanism. The paper claims that this method outperforms competing methods on the standard benchmark citation network datasets.

While the attention in GAT and AGNN focuses on integrating information from neighbouring nodes into target node, Graph Attention Model (GAM) takes a
series of steps on an input graph and encodes information from visited nodes using
an RNN to construct a sub-graph embedding. Assuming the RNN’s hidden state at
step \( t \) is \( h_t \), attention is then defined as a function \( f : \mathbb{R}^h \rightarrow \mathbb{R}^k \) that maps \( h_t \) to
a \( k \)-dimensional rank vector where \( k \) is the number of node types. The model will
then take the neighbouring node of a particular type with the highest rank score
for the next step. The attention-guided walk allows the model to focus only on the
informative part of the graph and avoid noise in the rest of the graph. Experimental
results show that GAM is competitive against various well-known methods in graph
classification tasks although this method is limited to only a portion of the graph.

AttentionWalks [40] is another representative graph attention model that was
inspired by DeepWalk. The problem with Deepwalk is that it has many hyper-
parameters such as the length of a random walk. These hyper-parameters need
be tuned manually for different graphs. AttentionWalks replaces these fixed hyper-
parameters with trainable attention parameters that can be automatically learned
via backpropagation. These attention parameters are utilized exclusively on the data
traversal and not used by the model for inference. Expereiments on link prediction
tasks show that the automatically-learned attention parameters correspond to the
optimal choice hyper-parameters if they are tuned manually in existing methods.

A graph with a single type of node and a single type of edge is called homogeneous.
Otherwise, the graph is heterogeneous. The aforementioned graph attention models
work on homogeneous graphs. However, they can be easily generalized to work on
heterogeneous graphs that have a single type of node but multiple types of edge.
Edge Attention-based Multi-Relational Graph Convolutional Networks (EAGCN) [41]
propose to use multi-headed attention similar to GAT, but each of the attention head
considers neighbours defined only by a particular edge type. For each node, the
learned embeddings of attention heads are then concatenated or summed to form
the final embedding of the node. Generalizing the graph attention model to work
on graphs with multiple edge types is beneficial to many real-world problems. For instance, in chemical graph theory, compound bonds are often represented by edges of different types. In a recommender system, different user-item ratings can also be represented by edges of different types.

### 3.4 Link Prediction in Graphs

Link prediction is an important problem for network-structured data. Given a snapshot of an undirected network at time $t$, the link prediction task can be defined as inferring the subset of missing links in the current snapshot or that will be formed in a later time $t + \Delta$. Each link in the network represents an interaction or relationship between the pair of nodes connected by the link $[42]$.

According to the work of [42], most link prediction methods are similarity-based. Similarity-based methods assume that nodes are more likely to form links with other similar nodes. Therefore, the key to these kinds of methods is to define a similarity function $s(x, y)$ that assigns a score for every pair of nodes $x$ and $y$. Similarity-based methods can be further categorized into local approaches and global approaches. We adopt the taxonomy from [42] to categorize the link prediction tasks into local similarity-based approaches, global similarity-based approaches, and the supervised learning approaches.

**Local Similarity-based Approaches**

Local similarity-based methods compute the similarity of two nodes using the neighbourhood structural information of the nodes. These methods are usually faster than non-local methods since they are highly parallelizable. They are also efficient to handle online link prediction problems. In the rest of this subsection, we use $\Gamma_i$ to denote the neighbourhood of the node $i$. 
Common neighbours (CM) \cite{43} is the most straightforward local method which defines the similarity score between two nodes as the number of shared neighbours:

\[
s(x, y) = |\Gamma_x \cap \Gamma_y|.
\]  

(9)

Although this method is simple, it works well on many real-world networks. The Jaccard index (JA) shares the similar idea to the CM. JA is inspired by the Jaccard coefficient, which is commonly used in information retrieval. JA measures the ratio of shared neighbours in the set of all neighbours of two nodes:

\[
s(x, y) = \frac{|\Gamma_x \cap \Gamma_y|}{|\Gamma_x \cup \Gamma_y|}.
\]  

(10)

The resource allocation index (RA) \cite{44} is inspired by the resource allocation process. It models the transmission of resources between two unconnected nodes through neighbourhood nodes. Each unit resource from one node is equally distributed to the other node. The similarity function of this method is defined as:

\[
s(x, y) = \sum_{z \in \Gamma_x \cap \Gamma_y} \frac{1}{|\Gamma_z|}
\]  

(11)

It can be seen from Equation 11 that each shared neighbour is penalized by its degree. This makes sense in many real-world networks since the amount of resource a node can send to one of its neighbours decreases as its degree increases. Also, the work from \cite{45} provides an interesting conclusion that the performance of CM based methods increases by penalizing the high-degree nodes in scale-free networks. This conclusion may also explain, in GCN, the symmetric normalization weights the neighbour of a node less if the neighbour has a higher degree.

The preferential attachment index (PA) is directly derived from the famous Barabasi-Albert network formation model \cite{46}. The key idea of the Barabasi-albert
network formation algorithm is the probability of connecting a new node to an existing node increases proportionally as the degree of the existing nodes. Therefore, the resulting scale-free network follows the power-law distribution. Many real-world networks are scale-free such as social networks and airline networks. According to [46], the probability of existing a link between two nodes is correlated with the product of the degrees of the two nodes:

\[ s(x, y) = |\Gamma_x||\Gamma_y|. \]  
\( (12) \)

**Global similarity-based approaches**

Global similarity-based methods can use topological information from the whole network to compute each similarity score. These methods are not limited to measuring the similarity between distance-two nodes. However, the main drawback of the global methods is the difficulty to scale to large networks.

Negated shortest path (NSP) [47] is a simple global similarity-based approach which defines the similarity score as the negation of the shortest path length between a pair of nodes:

\[ s(x, y) = -|\text{Shortest Path}_{x,y}|. \]  
\( (13) \)

The length of the shorted path can be found using Dijkstra’s algorithm. The equation shows that two nodes are more likely to form a link if the the shortest-path distance between them is smaller. Another similarity function based on the path length is the Katz index (KI) [48] which takes into account all possible paths between a pair of nodes. KI sums up the influence of all paths between the pair of nodes and incrementally penalize paths by its length:

\[ s(x, y) = \sum_{l=1}^{\infty} \beta^l |\text{path}_{x,y}^l|, \]  
\( (14) \)
where 0 < β < 1 and $\text{path}^l_{x,y}$ is the set of paths of length $l$ between $x$ and $y$.

In addition to the global methods based on path length, another category of global methods predict missing links using random walks (RW) on a graph \[49\]. Given a graph and a starting node, RW iteratively computes the probability of reaching any other nodes from the starting node. This process can be illustrated as:

$$\vec{p}^2(t) = (D^{-1}A)^t \vec{p}^2(t-1),$$

(15)

where $\vec{p}^2(t)$ is the probability vector and $D^{-1}$ is the inverse of the degree matrix. $D^{-1}$ normalizes the adjacency matrix $A$ by rows. Upon convergence, the similarity score of starting node $x$ and any other node $y$ is defined as:

$$s(s, y) = \vec{p}_y,$$

(16)

where the subscript $y$ means the $y^{th}$ element in the vector.

Random walks with restart (RWR) \[50\] is similar to RW except in each iteration, it has a probability of returning to the starting node. RWR is almost the same as the PageRank algorithm \[51\]. Flow propagation (FP) \[52\] view the RWR from the information propagation perspective and proposes to replace the row-normalized adjacency matrix $D^{-1}A$ by the symmetrically normalized adjacency matrix $D^{-1/2}AD^{-1/2}$. Note that GCN also adopts the exact same information propagation process as FP.

**Supervised learning approaches**

The previously discussed similarity-based methods mainly rely on heuristics to compute similarity scores. However, one heuristic can work well on one network but fail on the other. Therefore, it will be better to learn a suitable heuristic from a given network than using a predefined one. \[53\] developed $\gamma$-decaying heuristic theory which
unifies both local and global heuristics. The theory proves that all these heuristics can be well approximated from a small sub-graph induced by the two nodes of which we want to predict the existence of a link. The $\gamma$-decaying theory implies that the local sub-graph already contains enough information for link prediction. This is desired since learning from the entire network is often infeasible. Based on the $\gamma$-decaying heuristic theory, a link prediction frame called SEAL (learning from Subgraphs, Embeddings and Attributes for Link prediction) is proposed to learn heuristics from local sub-graphs using GCN. SEAL takes local enclosing sub-graph around two nodes as input, and output the likelihood of link existence. The application of GCN in SEAL makes it possible to also take into account each node’s attributes that are available in many real-world datasets.

The graph auto-encoder [4] is another neural network model proposed for link prediction problems on undirected graphs. Graph auto-encoder consists of an encoder and a decoder. The encoder is a graph convolutional network and the decoder can be a neural network or a simple inner product. Since the encoder is GCN, the model can naturally incorporate node attributes, which can arguably improve the predictive power of the model.

There are two versions of graph auto-encoder which are variational graph auto-encoder (VGAE) and non-probabilistic graph auto-encoder (GAE). The encoder of GAE can be GCN or any variants of GCN in order to calculate the network embedding matrix $Z$:

$$Z = GCN(X, A) \quad (17)$$

It followed by a decoder to reconstruct the adjacency matrix $\hat{A}$. The simplest form of the decoder is given by:

$$\hat{A} = \sigma(ZZ^T) \quad (18)$$
The difference between VGAE and GAE resides in the encoder. Instead of directly producing latent embedding for a given input, the encoder produces the mean embedding $\mu$ and the standard deviations $\sigma$. Similar to VAE, the final embedding is sampled from the normal distribution parameterized by $\mu$ and $\sigma$.

Another interesting variant of the graph auto-encoder is the Adversarially Regularized Graph Autoencoder (ARGA) \cite{54} which combines GCN with Generative Adversarial Networks (GAN) \cite{34}. The ARGA paper argues that most existing graph embedding algorithms are unregularized methods. ARGA employs the training scheme of GAN to regularize the learnt hidden representation of nodes to follow a prior distribution. Experimental results shows that ARGA outperforms baselines by a wide margin in link prediction, graph clustering, and graph visualization tasks.

3.5 Summary

This chapter covers works related to the attentive graph model and the link prediction problem. In the first two sections of this chapter, we discuss the attention mechanism in general and the recent development of the graph convolutional networks. Self-attention relates different positions of a single sequence to compute a representation of the same sequence. Since it does not rely on RNN, the idea can be generalized to data structures other than a sequence such as graphs. On the other hand, GCN requires a mechanism to compute the importance scores between a node and its neighbouring nodes so that information can propagate properly in the network. Therefore, it is natural to wonder if the attention mechanism can be implemented into the GCN for learning the importance scores.

The third section introduces existing deep graph models that incorporates the attention mechanism. The formal definition of the graph attention mechanism is also given in this section. The GAT model discussed in this section is especially inspiring
to this research. In the last section, we discuss the link prediction problem. It first
gives the formal definition of the link prediction problem. Our proposed model is
mainly based on the GAE described in the last section.
Chapter 4

Attentive Graph Auto-encoder model

4.1 Introduction

The main goal of this research is to implement the attention mechanism into the graph auto-encoder and evaluate its performance for the link prediction problem. In this chapter, we will describe our proposed model and its implementation in detail. The main idea of our model is to use GAT as the encoder of GAE. This model will be evaluated and analyzed in the next chapter in order to answer the research questions proposed in Section 1.3.

Section 4.2 provides a formal statement of the problem and its scope. Section 4.3 describes our proposed model in detail. Section 4.4 provides the vectorized implementation so that our model can be implemented and trained by deep learning frameworks. Section 4.5 discusses how the model is trained and Section 4.6 describes how we can use the trained model to predict potential links.

4.2 Problem Definition and Scope

Let $G = (V, E)$ be an undirected graph with $n = |V|$ nodes. $A \in R^{n \times n}$ is the adjacency matrix of $G$. Let feature matrix $X \in R^{n \times m}$ where $m$ is the number of
features that each node has. Some of the edges in \( G \) are removed while all node features are kept. Therefore \( A \) may have missing values. The focus of the prediction task is to predict those missing values in \( A \).

4.3 Methodology

Our method is based on the framework of GAE which consists of an encoder and a decoder. The encoder incorporates both graph topological and nodes’ attributes information and outputs latent representations of each node:

\[
Z = f_e(A, X),
\]  
(19)

where \( A \) is the adjacency matrix presenting topological information of the graph. \( X \) is the node-feature matrix where each row corresponding to a node and each column represents a specific feature. \( Z \) is the embedding matrix where each row represents a latent node embedding. In the most general case, the encoder can be any variation of graph neural networks.

Given latent representations of two nodes, the decoder predicts the existence of an edge between the two nodes:

\[
\hat{A}_{i,j} = f_d(z_i, z_j),
\]  
(20)

where \( \hat{A}_{i,j} \) indicates whether there exists a link between node \( i \) and \( j \). \( z_i \) and \( z_j \) are latent representations of node \( i \) and \( j \) outputted by the encoder.

In this research, we propose a GAE model called attentive graph auto-encoder (AGAE). The general idea of AGAE is to construct the encoder using GAT layers. We will describe the encoder and the decoder of AGAE in detail in the following subsections.
4.3.1 Attentive Encoder Layer

An encoder layer generates new representations of nodes by utilizing their neighbors’ representations according to their importance. The new representation of a node is determined by itself and its neighboring nodes as shown in Figure 11. Therefore, the way to compute the importance between two neighboring nodes plays a key role in an encoder layer. In GAE, the importance is explicitly determined by the degree of both nodes. In this research, we add the attention mechanism to each encoder layer in order to implicitly learn the importance during the training process. This is achieved by simply replacing each GCN layer by the GAT layer in the encoder. The formal definition of an attentive encoder layer is given below.

In the $k^{th}$ encoder layer, given a target node $i$, the importance of a neighboring node $j \in \Gamma_i$ is computed as:

$$e_{ij}^{(k)} = \sigma(a_s^{(k)} W^{(k)} h_i^{(k-1)}) + a_r^{(k)} W^{(k)} h_j^{(k-1)}$$  \hspace{1cm} (21)

where $W^{(k)} \in \mathbb{R}^{d(k) \times d(k-1)}$ is the trainable parameters of the $k^{th}$ encoder layer.
\( \mathbf{a}_s^{(k)} \in \mathbb{R}^{d^{(k)}} \) and \( \mathbf{a}_r^{(k)} \in \mathbb{R}^{d^{(k)}} \) are the trainable attention weights. In this model, we assume that the neighborhood of a node is its one-hop adjacent nodes. \( \sigma \) denotes the activation function such as LeakyReLU. In order to make the importance coefficients of a node comparable, they are normalized by softmax function as follows:

\[
\alpha_{ij}^{(k)} = \frac{\exp(e_{ij}^{(k)})}{\sum_{l \in N_i} \exp(e_{il}^{(k)})} \tag{22}
\]

where \( N_i \) represents the neighboring nodes of node \( i \). Equation 22 ensures that the sum of the importance scores of neighboring nodes to node \( i \) equals to 1. The final representation of node \( i \) output by the \( k^{th} \) encoder layer can be shown as:

\[
h_i^{(k)} = \sum_{j \in N_i} \sigma(\alpha_{ij}^{(k)} W_r^{(k)} h_j^{(k-1)}) \tag{23}
\]

### 4.3.2 Multi-headed Attention

The graph attention mechanism described in Section 4.3.1 is the single-headed attention mechanism. Multi-headed attention means aggregation of multiple independent graph attention layers to form a single layer [3]. In the case of K-headed attention, K independent attention mechanisms execute the transformation of Equation 23, and then their features are concatenated or averaged.

Figure 12 is an example of multi-headed attention with the number of heads equals to 2. Each arrow colour denotes an attention head which also means an independent attention computation. Each attention head outputs a new feature representation. In this example, the new feature representation outputted by the two attention heads are \( h'_1 \) and \( h'_2 \). The final feature representation is obtained by averaging or concatenating the new feature representations from all attention heads.
Figure 12: An illustration of multi-headed attention.

4.3.3 Decoder

We use the inner product of two node embeddings to reconstruct each link. The prediction of the reconstructed link between node $i$ and node $j$ is computed as follow:

$$\hat{A}_{ij} = \sigma(z_i^T z_j)$$ (24)

where $z_i$ and $z_j$ are latent representations of node $i$ and node $j$ respectively. $\sigma$ is the sigmoid activation function. $\hat{A}_{ij}$ is the entry of $i^{th}$ row and $j^{th}$ column of the recovered matrix $\hat{A}$ by the decoder. $\hat{A}_{ij}$ represents the probability of whether there is a link between nodes $i$ and $j$.

4.4 Vectorized Implementation

The AGAE model is constructed by stacking two GAT layers as the encoder, followed by the dot-product decoder. The reason for using two layers in the encoder is to be consistent with the architecture used with GAE. However, other number of
encoder layers are also viable and could be explored in the future. The overall architecture of AGAE is illustrated in Figure 13. In practice, we can use efficient matrix multiplications to implement the AGAE model.

Suppose the attributed input graph $G$ is represented by an adjacency matrix $A$ and a node feature matrix $X$. In order to compute the masked attention coefficient matrix for an encoder layer $k$, we first need to compute the matrix $B^k$ that contains unnormalized attention scores from all other nodes to each node in a graph:

$$B^k = \sigma(W^k H^k a^k_s + (W^k H^k a^k_r)^T).$$  \hspace{1cm} (25)$$

$W^k$ and $H^k$ are the weight matrix and the node feature matrix of respectively. $a^k_s$ and $a^k_r$ are the attention weight vectors. The + symbol here is a broadcasting operation which allows element-wise addition between tensors with different shapes. In this case, $B^k_{i,j}$ equals to the $i$th element of the column vector $(W^k H^k a^k_s)$ plus the $j$th element of the row vector $(W^k H^k a^k_r)^T$. $\sigma$ is the LeakyReLU activation function.

The masked attention coefficient matrix of the $k$th encoder layer can then be computed as:

$$A^k_{attn} = \text{softmax}(B^k + M).$$  \hspace{1cm} (26)$$
Since we only consider first-order neighbors of a node, we need to mask out all other attention coefficients using the mask matrix $M$. One way to construct $M$ is to assign all elements that correspond to a link to 0 and all other elements to negative infinity as: $M = -\infty \ast (1 - A)$. The softmax function is then applied row-wise to the masked matrix.

Given the final attention coefficient matrix $A^k_{attn}$, the encoder can be shown as follows:

$$H^1 = \sigma(A^1_{attn}W^1X)$$ \hspace{1cm} (27)
$$Z = \sigma(A^2_{attn}W^2H^1)$$ \hspace{1cm} (28)

$X$ is the input feature matrix. $W^1$ and $W^2$ are weight matrices for encoder layers. The decoder takes the node embedding matrix $Z$ and outputs the reconstructed adjacency matrix $\hat{A}$ as:

$$\hat{A} = \sigma(ZZ^T),$$ \hspace{1cm} (29)

where $\sigma$ is the logistic sigmoid function.

### 4.5 Training

We train the model by minimizing the reconstruction error between the predicted link existence in $\hat{A}$ and the ground-truth link existence in $A$. We adopt the weighted cross-entropy loss function used to train GAE in [4]:

$$\text{Loss} = -\sum_i \sum_j (A_{ij}\log(\hat{A}_{ij})w + (1 - A_{ij})\log(1 - \hat{A}_{ij}))$$ \hspace{1cm} (30)

This loss function is almost the same as the cross-entropy loss, except the post-weight $w$ is introduced. The weighted cross-entropy loss is often used when the data
is imbalanced. In the case of a typical network, there are fewer 1s than 0s in the adjacency matrix. Therefore, we give more weight to the first term to penalize more on the false negative (i.e., Increase the recall.). The weight \( w \) is determined by \( w = \frac{\text{number of 0s}}{\text{number of 1s}} \) in the adjacency matrix.

All the trainable weights in the model (e.g., \( W^k, a^k, \) and \( a^k \)) are updated using gradient descent. We perform batch gradient descent using the full dataset for every training iteration. Stochasticity in the training process can be introduced via dropout. Dropout can be applied to each layer’s input, as well as the normalized attention coefficients \( A^k_{\text{attn}} \). If dropout is applied to the normalized attention coefficients, each node is exposed to a stochastically sampled neighborhood at each training iteration.

4.6 Inference

After the model is trained, we can get the network embedding \( Z \) by forward passing the adjacency matrix \( A \) and the input feature matrix \( X \). To predict the existence of a link between two nodes \( i \) and \( j \), we perform dot-product on the embeddings of \( i \) and \( j \) and apply the sigmoid activation function:

\[
\hat{A}_{i,j} = \sigma(Z_i Z^T_j).
\] (31)

There exists a link between \( i \) and \( j \) if \( \hat{A}_{i,j} > 0.5 \).

4.7 Summary

This chapter describes our proposed method that applies the graph attention mechanism for the link prediction problem. The content of this chapter should answer the RQ1 in Section 1.3. The work that has been presented in this chapter will be used in the experiments discussed in the next chapter.
Chapter 5

Experiments

5.1 Introduction

The AGAE model described in Chapter 4 is evaluated and analyzed in this chapter. The AGAE is compared with the original GAE to see if the graph attention mechanism can learn better embedding for the link prediction task.

Section 5.2 describes how the training and evaluation datasets are generated. It includes the preparation of the real-world citation network data and the algorithm to generate synthetic data. Section 5.3 discusses the evaluation metrics used for the experiments. Section 5.4 describes evaluation and analysis of the AGAE model on citation network datasets. The analysis includes the visual inspection of the encoder outputs, the distribution of attention coefficients, and the effectiveness of the multi-headed attention. Section 5.5 describes evaluation and analysis of the AGAE model on synthetic networks. The performance of the model on networks with certain characteristics is investigated using the synthetic datasets.
5.2 Data Preparation

5.2.1 Citation Networks

Cora and Citeseer are two real-world network benchmark datasets often used in network inference tasks \[55,56\]. The datasets are prepared the way similar to the method described in \[57\]. The Cora dataset consists of a number of machine learning papers labelled by 7 different topics. These labels are: case based, genetic algorithms, neural networks, probabilistic methods, reinforcement learning, rule learning, and theory. We choose our corpus so that each chosen document is either cited or cites at least one of the other documents in the corpus. The undirected graph is formed by linking each pair of nodes if there is a cited relationship between them. The attributes for each node are bag-of-words features. Stop words and words appearing in less than 10 documents are not included in the vocabulary. The final processed Cora dataset has 2078 documents with a vocabulary of 1433 words and 5429 links. The CiteSeer is a public digital library of academic papers in the field of computer science and information science. The CiteSeer dataset is extracted from the CiteSeer database and processed in the same way as we process Cora. The documents are labelled with one of the class labels of agents, artificial intelligence, database, information retrieval, machine learning, and human computer interaction. The final corpus has 3312 documents with the vocabulary size of 3703 and 4732 links. The statistics of the two datasets is summarized in Table 1. Although the node labels are not needed by the AGAE model, it is used for analyzing the encoder outputs which is discussed in Section 5.4.3.

5.2.2 Synthetic Network

We use the algorithm described in Algorithm 1 \[57\] to construct the synthetic network.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Features</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>2708</td>
<td>5429</td>
<td>1433</td>
<td>7</td>
</tr>
<tr>
<td>Citseer</td>
<td>3327</td>
<td>4732</td>
<td>3703</td>
<td>6</td>
</tr>
</tbody>
</table>

**Table 1:** Summary of real-world datasets.

**Algorithm 1:** SyntheticNetworkGenerator(numNodes, numLabels, attrNoise, vocabSize, numWords, otherArgs)

1. Set $G = \emptyset$;
2. addNode($G$, numLabels, numNodes, otherArgs);
3. for $i = 1$ to numNodes do
   4. $v \leftarrow i^{th}$ node in G;
   5. getAttributes($v$, numLabels, attrNoise, vocabSize, numWords)
4. end
5. return $G$

This algorithm grows an undirected attributed graph from an empty set of nodes. The parameter numNodes determines number of nodes in the graph. The numLabels is the number of different node labels. The vocabSize determines the number of features each node has. The attrNoise is a float from 0 to 1 that determines the noise level of node features. The numWords is the maximum number of words from the vocabulary in a node. The meaning of otherArgs depends on the addNode function. It could be the parameter to control the link density or the parameter to control the topological randomness. The addNode function defines how the nodes are topologically connected in the graph. For instance, the Barabási–Albert preferential attachment algorithm can be used to create scale-free networks [46]. The Watts-Strogatz algorithm can be used to create small-world networks [58].

The getAttributes function creates attributes for each node. These attributes are
sampled from certain distributions according to the node label. Some common distribution often used for modelling bag-of-words documents are binomial distribution, multinomial distribution, and Latent Dirichlet Allocation (LDA) \cite{59}. According to \cite{57}, simply sampling words from the binomial distribution has enough amount of discriminative power to model documents. For the sake of simplicity, we use the binomial distribution to sample attributes for each node in our experiment. The binomial distribution $X \sim B(n, p)$ can be formulated as:

$$\Pr(X = k) = \frac{n!}{k!(n - k)!} p^k (1 - p)^{(n-k)},$$

where $n$ is the number of trials and $p$ is the probability of getting exactly $k$ success. In the scenario sampling attributes for each node, $n$ is the number of attributes of a node (e.g., vocabulary size) and $p$ is $(1 + \text{label of the node})/(1 + \text{number of labels})$. The algorithm of generating attributes for each nodes is described in Algorithm \[2\]. In addition to sampling from the binomial distribution, there is also a probability of $\text{attrNoise}$ that the word-ID is sampled uniformly at random. This is the mechanism of how the attribute noise is added.
Algorithm 2: \texttt{getAttributes}(v, numlabels, vocabSize, numWords, attrNoise)

1. for \(i = 1\) to \(numWords\) do
2. \hspace{1em} sample \(r\) uniformly at random from \([0, 1)\);
3. \hspace{1em} if \(r \leq \text{attrNoise}\) then
4. \hspace{2em} sample a word-ID uniformly at random from \([0, \text{vocabSize})\);
5. \hspace{2em} add the word-ID to \(v\);
6. \hspace{1em} else
7. \hspace{2em} sample a word-ID from the binomial distribution with \(p = (1 + v.label)/(1 + \text{numLabels})\) and \(n = \text{vocabSize}\);
8. \hspace{2em} add the word-ID to \(v\);
9. end

5.3 Evaluation Metrics

The evaluation metrics used in this research are the area under receiver operating characteristic curve (ROC-AUC), the average precision score (AP), and simply the classification accuracy. The ROC-AUC and AP are metrics often used in GAE-based link prediction researches [4, 54, 60]. We use these two metrics so that our results can be compared with other GAE-based models. Also, ROC-AUC and AP are calculated on the predicted scores while accuracy is calculated on the predicted classes. A certain threshold is needed before computing the accuracy. For the binary classification tasks, 0.5 is the obvious choice but it can be sub-optimal. On the other hand, ROC-AUC and AP can summarize over all possible thresholds. Therefore, in addition to accuracy, it is always preferable to include score-based metrics for evaluating the model.

ROC-AUC score is the area under the ROC curve. ROC curve shows the trade off between true positive rate (TPR) and false positive rate (FPR). It plots TPR and FPR
Figure 14: Example ROC Curve.

pairs for every threshold. Figure 14 shows an example plot of ROC curve. A good classifier with a reasonable threshold will have high TPR and low FPR. Therefore, a perfect classifier will have ROC-AUC equals to 1 and a purely random classifier will have ROC-AUC equals to 0.5. The ROC-AUC score of the ROC curve in Figure 14 is 0.928. The ROC-AUC cares equally about positive and negative classes.

The AP score is basically the precision averaged across all values of recall between 0 and 1. It can also be viewed as the area under the precision-recall curve. An example precision-recall curve is shown in Figure 15. The precision-recall curve shows how precision and recall changes as you change the threshold. A good classifier’s precision will stay high as recall increases. The AP score provides a single number between 0 and 1 to summarize the precision-recall curve. The classifier with better performance has higher AP score. The AP score of the precision-recall curve in Figure 15 is 0.923. The AP cares more about positive than negative class. It is more sensitive to the
improvements for the positive class.

In our experiments, the test datasets are balanced so that they have equal number of positive and negative labels. The classification accuracy is a good metric for balanced datasets. It is straightforward and easy to interpret. The classification accuracy cares equally about positive and negative classes.

5.4 Experiments on Citation Networks

5.4.1 Initial Experiments on Cora and Citeseer

We evaluate the AGAE model on two standard citation network benchmark datasets which are Cora and Citeseer. The AGAE model works on both attributed graphs and non-attributed graphs. For the non-attributed graphs, the one-hot encoded feature matrix can be used to distinguish each node. In this research, we only consider
the application of AGAE on attributed graphs. We closely follow the experimental setup in [4]. The model is trained on an incomplete version of these datasets where parts of the citation links are removed while all node features are kept. Validation and test sets are formed from those removed edges as well as the same number of randomly sampled non-edges. The validation set contains 5% of citation links while the test set contains 10% of citation links. In each run of the experiment, the dataset is randomly split, meaning that the training set, validation set, and testing set are selected uniformly at random. We compare the performance of AGAE with GAE. For GAE, we use the implementation and hyperparameters from [4].

The model is implemented and trained using Tensorflow [61]. The weights are initialized using Glorot Norm as described in [11]. Due to the limited computation resources, we adopt the hyperparameter values in [4] and only tune the dropout rates. The model is trained using Adam SGD optimizer [62] with a learning rate of 0.01. The output of the first encoder layer is 32-dimensional and the output embedding is 16 dimensional. We use a grid search technique to tune the feed-forward dropout rate $p_{ffd} \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and the attention dropout rate $p_{ad} \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. The optimal $p_{ffd}$ is 0.4 and optimal $p_{ad}$ is 0.1.

Results for link prediction task on Cora and Citeseer are shown in Table 2 and 3 respectively. Each result is the mean and standard error for 10 runs on random dataset splits. It can be seen from the results that AGAE achieves comparable performance to GAE on both Cora and CiteSeer datasets.

5.4.2 Analysis of the Attention Coefficient Distribution

The encoder of AGAE is GAT and the encoder of GAE is GCN. The key difference between GAT and GCN is how the importance scores between a node and its neighbors are determined. GAT implicitly captures these importance scores as the
<table>
<thead>
<tr>
<th>Method</th>
<th>ROC AUC</th>
<th>AP</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAE</td>
<td>0.914 ± 0.008</td>
<td>0.922 ± 0.008</td>
<td>0.761 ± 0.014</td>
</tr>
<tr>
<td>AGAE</td>
<td>0.907 ± 0.006</td>
<td>0.914 ± 0.006</td>
<td>0.759 ± 0.010</td>
</tr>
</tbody>
</table>

Table 2: Compare the performance of GAE and AGAE on Cora.

<table>
<thead>
<tr>
<th>Method</th>
<th>ROC AUC</th>
<th>AP</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAE</td>
<td>0.907 ± 0.009</td>
<td>0.920 ± 0.008</td>
<td>0.748 ± 0.014</td>
</tr>
<tr>
<td>AGAE</td>
<td>0.884 ± 0.010</td>
<td>0.899 ± 0.009</td>
<td>0.740 ± 0.012</td>
</tr>
</tbody>
</table>

Table 3: Compare the performance of GAE and AGAE on Citeseer.

attention coefficients via attention mechanism during training. On the other hand, GCN explicitly assign the importance score $a_{ij}$ between node $v_i$ and $v_j$ by:

$$
a_{ij} = \frac{1}{\sqrt{\deg(v_i)\deg(v_j)}},
$$

Equation 33 is called symmetric normalization since the non-parametric importance score is determined by both degrees of the nodes connected by the edge. By normalizing the edges by their node degrees, it prevents nodes with large degrees to have large values in their feature representations while nodes with small degrees have small values. In deep learning, unscaled features can lead to exploding gradients and make the network hard to train. We denote the importance score computed by the attention mechanism as attention coefficient and the importance score computed by Equation 33 as non-parametric coefficient.

In the previous section, we observe that AGAE and GAE have comparable performance on citation networks. Therefore, we want to investigate if the attention mechanism just learns a similar importance score equal to what is explicitly assigned
Figure 16: First encoder layer attention coefficients vs the non-parametric coefficients.

in GCN. We compare attention coefficients of the first and second encoder layers with the non-parametric coefficients. We plot the distributions of the attention coefficients and the non-parametric coefficient as shown in Figure 16 and 17. The results are obtained by training AGAE and GAE on the Cora dataset.

It can be seen from Figure 16 and 17 that both the first and the second encoder layer attention coefficients follow a similar distribution as the non-parametric coefficients. The non-parametric coefficients do not change over the encoder layers since it is determined solely by the input graph structure. Most of the weights reside in the range between 0.0 to 0.6 with the interval [0.1, 0.2) having the highest frequency. There are some weights distributed in the interval [0.9, 1] since there are 104 isolated nodes in the training graph (e.g., Each of these nodes only has self-connection to itself). The self-connection weight of an isolated node equals 1.0 according to 33. There is no non-parametric coefficients distributed in the range 0.6 to 0.9 while there are a few attention coefficients distributed in the range 0.6 to 0.8. Interestingly, it looks like
the distribution of the attention coefficients is broader than the non-parametric coefficients. We will leave the investigation on this as future work. Overall, we can qualitatively see from the figure that both attention coefficients and the non-parametric coefficients have similar distributions. In other words, the attention mechanism can learn similar edge weight distributions to those provided by symmetric normalization.

**5.4.3 Visual Analysis of Encoder Outputs**

The encoder creates lower dimensional embeddings of the inputs, which provide further information needed for the decoder to predict edge links. In order to know if the learned latent embeddings can provide meaningful information for the decoder, we qualitatively investigate the encoder outputs by providing a visualization of the embeddings in a two-dimensional space.

We use the t-distributed Stochastic Neighbor Embedding (t-SNE) visualization algorithm for this purpose. The t-SNE takes high dimensional data and reduces
them into a low dimensional space that retains a lot of the original information. In
this case, we project 16-dimensional encoder outputs into the 2-dimensional space.
The perplexity is an important tunable parameter of the t-SNE algorithm. It is
related to a guess about the number of close neighbors each point has. According
to [63], the performance of t-SNE is fairly robust to changes in the perplexity and
typical perplexity values are between 5 and 50. We choose the perplexity value equal
to 30 for this experiment. We use the t-SNE implementation from the scikit-learn [64]
library and keep all other parameters as default.

In both Figure 18 and Figure 19, the different node colours represent 7 different
labels of the nodes. Here, we assume that nodes with the same label are similar to
each other. Figure 18a shows the projection of the feature input. Figure 18b shows
the projection of the AGAE encoder output. By comparing the two charts, we can
see that node embeddings are more clustered than the input. This implies that useful
graph structural information is added to the node feature embeddings by the AGAE
encoder. Figure 19 compares the AGAE encoder output (a) with the GAE encoder
output (b). We can see from the figure that AGAE encoder can generate embeddings
as good as the GAE encoder. This is consistent with the quantitative results in Table

![Figure 18: Compare input feature with the attentive encoder output.](image)
Figure 19: Compare attentive encoder output with the non-attentive encoder output.

5.4.4 Multi-headed Attention

The GAT paper [3] states that multi-headed attention has been found to be beneficial for the learning process of self-attention. However, the paper neither explains why it is beneficial nor includes any experiment results to show that multi-headed attention is better than single-headed attention in graph learning tasks. Therefore, we compare multi-headed AGAE with single-headed AGAE to see if AGAE can benefit from multi-headed attention.

According to [3], k-headed attention means aggregation of k independent graph attention layers to form a single layer.

In our multi-headed implementation, we use concatenation for the first encoder layer and averaging for the second encoder layer in accordance with GAT [3]. The output dimensionality of each multi-headed encoder layer is same as the corresponding single-headed encoder layer. The first encoder layer consists of 4 attention heads outputting 8 features each. All the heads are concatenated for a total of 32 output
<table>
<thead>
<tr>
<th>Metric</th>
<th>Single-headed attention</th>
<th>Multi-headed attention</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>0.913 ± 0.007</td>
<td><strong>0.920 ± 0.006</strong></td>
</tr>
<tr>
<td>ROC-AUC</td>
<td>0.906 ± 0.006</td>
<td><strong>0.911 ± 0.007</strong></td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.754 ± 0.014</td>
<td><strong>0.765 ± 0.009</strong></td>
</tr>
</tbody>
</table>

Table 4: Compare multi-headed attention with single-headed attention.

features. The second encoder layer consists of 4 attention heads outputting 16 features each. All the heads are averaged for a total of 16 output features. We compare the performance of the multi-headed AGAE with the single-headed AGAE. The evaluation is performed on the Cora dataset. All other experiment settings are same as the experiment settings described in Section 5.4.1. Due to the time constraint, we did not tune the hyperparameter number of attention head. The choice of 4 attention heads is according to the implementation of GAT in [3].

The results are shown in Table 4. We can see from the table that the multi-headed AGAE works slightly better than single-headed AGAE on the Cora citation network. Therefore, we confirm that the link prediction task can also benefit from the multi-headed attention. However, due to multi-headed attention requires more computation resources, we use single-headed attention for all other experiments in this thesis.

5.5 Experiments on Synthetic Networks

In this section, we evaluate GAE and AGAE on synthetic networks generated by Algorithm 1. The benefit of using synthetic networks for evaluation is that we have control over the network properties so that we can better analyze whether the models work well on networks with certain properties. The models are evaluated on two well-known networks which are the scale-free network and the small-world network. These
two types of networks are often observed in reality. To the best of our knowledge, this is the first work done to evaluate graph convolutional network models against certain network characteristics.

5.5.1 Scale-free Networks

Many real-world networks have hubs which are a small number of nodes with a very large degree [65]. This kind of networks can be modeled by the scale-free network [46]. The degree distribution of a scale-free network follows the power law. The hypothesis of the scale-free network is that nodes with a larger degree are more likely to be connected by new nodes being added to the network (e.g., The rich-get-richer phenomena). The World Wide Web and citation networks are two typical networks that have the scale-free property. In this experiment, we use the preferential attachment process to populate the scale-free network for evaluating AGAE and GAE models. This process starts with a small seed network and the probability of connecting a new node to an existing node is proportional to the degree of the existing node. In our implementation, we also consider the nodes’ labels so that a new node is always connected to the existing node with the same label [57]. This makes sense since similar nodes are more likely to connect to each other. Our implementation of the preferential attachment process is described in Algorithm 3. The parameter $m$ in this algorithm determines how many new links are created when a new node is added. The final algorithm used for generating the attributed graph is by replacing the addNode function in Algorithm 1 by the Algorithm 3.
Figure 20: Degree distribution of a network generated by our implementation of preferential attachment.

Algorithm 3: preferentialAttachment(G, numLabels, numNodes, m)

1 for i=1 to numNodes do
2    add a new node v to G;
3    randomly choose a label c from [1...numLabels);
4    Set v.label ← c;
5    select m existing node from G with the labels equal to c and probability of selection proportional to their degree;
6    introduce edges between v and the m existing nodes;
7 end

The degree distribution of an example scale-free network generated using preferential attachment is shown in Figure 20. The x-axis is the number of degree in log scale and the y-axis is the count of nodes. The network is generated with the parameters of 500 nodes, 10 labels, and m = 10. It can be seen from the figure that the distribution of the node degrees approximately follows the power law.
<table>
<thead>
<tr>
<th>parameters</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>numNodes</td>
<td>3000</td>
</tr>
<tr>
<td>numLabels</td>
<td>10</td>
</tr>
<tr>
<td>attrNoise</td>
<td>0.2</td>
</tr>
<tr>
<td>vocabSize</td>
<td>50</td>
</tr>
<tr>
<td>numWords</td>
<td>25</td>
</tr>
<tr>
<td>m</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5: Parameter settings for the synthetic data generator.

<table>
<thead>
<tr>
<th>Model</th>
<th>ROC-AUC</th>
<th>AP</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAE</td>
<td>0.880 ± 0.056</td>
<td>0.869 ± 0.053</td>
<td>0.766 ± 0.061</td>
</tr>
<tr>
<td>AGAE</td>
<td><strong>0.941 ± 0.004</strong></td>
<td><strong>0.898 ± 0.008</strong></td>
<td><strong>0.869 ± 0.024</strong></td>
</tr>
</tbody>
</table>

Table 6: Evaluation of GAE and AGAE on synthetic networks.

The synthetic network used for this experiment has 3000 nodes and 50 attributes. Table 5 shows the parameter settings of our synthetic data generator. In each experiment, we use the algorithms and the parameters described above to generate a new network for evaluating AGAE and GAE. Each result shown in Table 6 is the average of 10 independent experiments. The hyper-parameters used in this experiment is the same as in the citation network datasets except all the dropout rates are set to be 0 and the training epoch number is set to be 100.

It can be seen from the results that AGAE works significantly better than GAE on all three metrics. Therefore, it can be concluded that AGAE works better than GAE on the scale-free networks.

Performance with Varied Link Density

Network density is an important property of a network. Network density is the portion of the potential connections in a network that are actually connected. The network
density in an undirected network can be computed as:

\[
\text{density} = \frac{2 \times e}{n \times (n - 1)},
\]

where \( e \) is the number of edges and \( n \) is the number of nodes in a network. The citation networks we used in the previous experiments are very sparse. For instance, the Cora network has 2078 nodes and 5429 links. The link density of Cora is only 0.15% approximately.

In the preferential attachment algorithm described in Algorithm 3, the network is initialized by \( m \) unconnected nodes and each new node is connected with \( m \) existing nodes. Therefore, the number of links in the network can be determined by the parameter \( m \) as:

\[
e = (n - m) \times m,
\]

where \( m < n \).

In this experiment, networks are produced with \( m = 2^k \), where \( k = 1, 2, 3, 4, 5, 6, 7, 8, 9 \). All other parameters are fixed to the values described in Table 5. AGAE and GAE are then evaluated on these synthetic networks. The results are summarized in Figure 21. In each sub-figure, we plot the metric scores against the \( \log(m) \).

It can be seen from the results that the performance of AGAE is better than GAE at most of the \( m \) values. Overall, the performance of both models declines as the \( m \) value goes up. One possible reason for this is that when \( m \) is large, the network is too dense to be a scale-free network. Figure 22 plots the degree distribution of a network generated by Algorithm 3 with 500 nodes and \( m = 100 \). Comparing to the Figure 20, we can see that the degree distribution no longer follows the power law. Therefore, the AGAE works better on networks with the scale-free characteristic.
Figure 21: Evaluation of GAE and AGAE on scale-free networks with different link density.
5.5.2 Small World Network

The small-world network is another kind of network that is often observed in reality. Social networks, the neural network of the worm C-elegans, and the power grid of the western United States are all shown to be small world networks [58]. Generally in a small-world network, most nodes are not neighbours of one another, but the distance between two nodes is often small. In order to generate the small-world network, we start with a connected caveman network [66]. The connected caveman network is formed by creating \( n \) cliques of size \( k \), then a single edge in each clique is rewired to a node in an adjacent clique. For each edge \((u, v)\) in the connected caveman network, it has a probability \( p \) to be replaced with \((u, w)\) where \( w \) is chosen at uniformly random from existing nodes. The algorithm to generate small-world graphs is described in Algorithm 4.
Algorithm 4: smallWorld($G$, $numLabels$, $numNodes$, $p$)

1. add $numLabels$ cliques to $G$, each clique has $numNodes/numLabels$ nodes;
2. randomly rewire a single link in each clique to a node in its adjacent clique;
3. for all other edges, each edge $(u, v)$ has a probability $p$ to be rewired by $(u, w)$. $w$ is randomly chosen from other nodes in $G$;

In order to generate attributed small-world network, we replace the addNode function in Algorithm 1 by Algorithm 4. The topological structure of an example small-world network is illustrated in Figure 23. It has 10 communities and each community has 20 nodes. The rewiring probability $p = 0.01$.

The synthetic small-world network generated for evaluation has 10 communities, 300 nodes in each community. The rewiring probability $p = 0.01$. So the final network

**Figure 23:** Example small-world network with 10 communities and 20 nodes in each community.
<table>
<thead>
<tr>
<th>parameters</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>numNodes</td>
<td>3000</td>
</tr>
<tr>
<td>numLabels</td>
<td>10</td>
</tr>
<tr>
<td>attrNoise</td>
<td>0.2</td>
</tr>
<tr>
<td>vocabSize</td>
<td>50</td>
</tr>
<tr>
<td>numWords</td>
<td>25</td>
</tr>
<tr>
<td>p</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 7: Parameter settings for generating the small-world networks.

<table>
<thead>
<tr>
<th>Model</th>
<th>ROC-AUC</th>
<th>AP</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAE</td>
<td>0.857 ± 0.064</td>
<td>0.861 ± 0.061</td>
<td>0.746 ± 0.056</td>
</tr>
<tr>
<td>AGAE</td>
<td><strong>0.995 ± 0.000</strong></td>
<td><strong>0.996 ± 0.000</strong></td>
<td><strong>0.945 ± 0.024</strong></td>
</tr>
</tbody>
</table>

Table 8: Evaluation of GAE and AGAE on small-world networks.

The small-world network lies between the completely regular network and the completely random network [58]. A regular network is a network where every node has

has 3000 nodes and 448500 edges in total. Table 7 summarizes the parameters used for generating the small-world networks for this experiment.

The experimentation results on AGAE and GAE are shown in Table 8. We show mean and standard deviation of 10 independent experiments. In each independent experiment, a new network is generated and split for training and testing. The models are trained for 100 epochs without dropout. All other settings are the same as those used in the citation-network experiments.

It can be seen from the Table 8 that AGAE works significantly better than GAE on the small-world networks. Therefore, it can be concluded that AGAE works better than GAE on small-world networks.

Performance with the Varied Topological Randomness

The small-world network lies between the completely regular network and the completely random network [58]. A regular network is a network where every node has
the same degree. Given the number of nodes \( n \) and the number of edges \( m \), a completely random network is a network chosen uniformly at random from the set of all networks which have \( n \) nodes and \( m \) edges.

The parameter \( p \) in the Algorithm 4 controls the topological randomness of the small-world network. If \( p = 0 \), the network becomes a connected caveman network. If \( p = 1 \), the network becomes a completely random network. By varying the parameter \( p \) from \( p = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \), we evaluate the performance of the two models against different degree of topological randomness. We plot the testing accuracy, ROC-AUC, and AP of the two models against the randomness degree \( p \) as shown in Figure 24.

We can see from the figure that the performance of both models declines as \( p \) increases from 0 to 1. Overall, AGAE works better than GAE at almost all of the \( p \) values. Both models perform poorly as \( p \) approaching 1. Therefore, it can be concluded that GAE and AGAE do not work well on topologically random networks.

**Performance with the Varied Attribute Noise Level**

In the attribute sampling procedure described in Algorithm 2, the degree of the attribute noise level can be controlled by the attrNoise. In reality, node attributes have naturally occurring noise distributions. In the following experiment, we examine the effect of attribute noise on GAE and AGAE. We use the same algorithm to generate small-world networks with rewiring probability \( p = 0.01 \). All other network parameter settings are same as the parameters in Table 7 except we vary the attrNoise. The evaluation dataset consists of networks with \( \text{attrNoise} = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \). The evaluation results of AGAE and GAE are shown in Figure 25.

We can see from the figure that AGAE works significantly better than GAE at all noise levels. As the noise level increases, the performance of AGAE does not
Figure 24: Evaluation of GAE and AGAE against different topological randomness levels.
Figure 25: Evaluation of GAE and AGAE against attribute noise levels.
significantly decline while GAE works almost randomly when the noise level is high. Therefore, it can be concluded that AGAE is more resistant to attribute noise than GAE.

5.6 Limitations

Although AGAE achieves comparable performance to GAE on citation networks and outperforms GAE on synthetic networks, it does have some limitations.

Due to the fact that the attention mechanism introduces more trainable weights to the model, the training and inference time of AGAE will be greater than GAE. When the dataset is large, these differences could be significant. Therefore, the average training time per epoch and the average inference time could be measured so that we can have an idea of the scalability of AGAE.

In this research, we did limited hyper-parameter tuning. We borrowed many hyper-parameter settings from GAE for AGAE. More hyper-parameter tuning effort has the potential to further improve the performance of AGAE. For instance, it is worth to explore the optimal number of encoder layers for AGAE.

Moreover, in this research, the real-world citation datasets are prepared the same way as how it is prepared for GAE in [4]. The extracted features for each node are bag-of-words features. However, the bag-of-words features might not be optimal for AGAE. Other topic modeling techniques such as LDA can be tested to see if they can further improve AGAE performance.

5.7 Summary

This chapter discusses the experiments that are performed on the AGAE model. The AGAE model is evaluated on both real-world and synthetic networks. The research
question RQ2 can be answered by all the experiment results in this chapter. In general, AGAE achieves comparable performance to GAE on real-world datasets and works better than GAE on synthetic datasets. The research question RQ3 can be answered by the results in Section 5.5. AGAE achieves better performance on both scale-free and small world networks. AGAE is more resistant to the attribute noise than GAE.
Chapter 6

Conclusion and Future Work

This thesis introduces a method called AGAE that introduces the graph attention mechanism into the GAE. The graph attention mechanism has shown its ability to learn good graph embeddings for node classification tasks. In this research, we evaluate its performance on the link prediction tasks. To achieve this, AGAE simply replaces the encoder layers in GAE by the graph attention layers and keep the decoder unchanged. The formal description of AGAE and its implementation is introduced in Chapter 4.

Since AGAE is based on the GAE framework, we compare the performance of AGAE with GAE to see if it benefits from the graph attention mechanism. The evaluations are performed on real-world citation networks and synthetic networks. We extensively analyzed the experiment results in both types of network. The analysis on the citation networks mainly regards the AGAE model itself, including the distributions of learnt attention coefficients, the qualitative inspection of the encoder outputs, and the effectiveness of the multi-headed attention. The analysis on the synthetic networks involves the performance of AGAE model on networks with different characteristics. These characteristics include network topology, link density, topological randomness, and attribute noise level.

Section 6.1 summarizes the results of the experiments carried out for this thesis.
Section 6.2 provides potential research directions that could be done on top of this research. The python implementation of AGAE model and experiments described in this thesis can be found at: https://github.com/leereborn/gae.

6.1 Summary of Results

AGAE achieves comparable performance to GAE on both Cora and CiteSeer citation networks. By further comparing the distribution of attention coefficients from AGAE with non-parametric coefficients from GAE, it is shown that their distributions are similar. The qualitative inspection of the encoder outputs shows the AGAE learns meaningful embeddings as good as GAE. Our results show that the AGAE model can benefit from the multi-headed attention mechanism.

On synthetic networks, AGAE and GAE are compared by varying network characteristics. The observations are summarized in the list below:

- AGAE works significantly better than GAE on both scale-free and small-world networks, especially on the small-world networks where AGAE achieves extremely good performance.

- Performance of AGAE declines as the scale-free network density increases which may indicate that AGAE works better on networks with the scale-free characteristic.

- AGAE works better than GAE on small-world networks of almost all the topological randomness levels. However, the performance of AGAE declines steadily as the topological randomness level goes up. Therefore, the topological randomness has negative impact on the performance of AGAE and AGAE barely works on the completely random networks.
• Surprisingly, the performance of AGAE does not dramatically decrease while the performance of GAE declines significantly as we increase the attribute noise level. This observation suggests that the graph attention mechanism is more resistant to the attribute noise.

In general, the graph attention mechanism can learn significantly better network embedding on networks with certain characteristics.

6.2 Future Work

6.2.1 Further Analysis of Attention Coefficient

Attention coefficients play a key role in an attentive graph model. In this research, we compare the attention coefficient with the non-parametric coefficient by plotting their distribution. Distribution fitting techniques can be used to fit the coefficients so that the approximate type of the distribution can be recognized. The distance of the two distributions can also be computed to quantitatively measure two distributions. In this thesis, the attention coefficients are only analyzed on the Cora network. However, the coefficients could also be analyzed on the synthetic networks to observe how the distributions change as graph characteristics are varied.

Since an attention coefficient represents the importance of one node to another, analyzing these learned attention coefficients may also lead to benefits in interpretability of the network. It can be observed from Figure 17 that the distribution of attention coefficients is actually broader than non-parametric coefficients. Investigations on why the non-parametric coefficients do not fall in certain ranges of large value can be done in the future. By sampling nodes which are more important than others as well as edges with large attention coefficients, it can be analyzed if these nodes and edges are key to certain network properties.
6.2.2 Decoder Architecture

AGAE adopts the simple inner product of two embeddings as the decoder. However, the bilinear scoring function (BSF) is often used in other literature for link prediction tasks [67–69]. The most general form of the BSF can be written as:

\[ s = e_i^T M e_j \]  

(36)

where \( e_i \) and \( e_j \) are embeddings of two nodes in the graph. \( M \) is a learnable weight matrix. The inner product decoder is actually the simplified special case of BSF which have \( M \) fixed to an identity matrix. [67] claims that if the weight matrix \( M \) is restricted to be diagonal, it achieves good performance on several link prediction tasks. In this case, the decoder can be viewed as the weighted element-wise dot product of two embeddings. By replacing the inner-product decoder by BSF, the performance of AGAE could potentially be improved. Therefore, evaluations can be performed on AGAE with diagonal BSF decoder.

6.2.3 Applications in Recommendation systems

In a recommendation system, interactions between users and items can be modeled as a bipartite graph. The recommendation can then be reduced to a link prediction problem with users and items represented by nodes and user-item interactions represented by edges. [68] proposes to apply GAE for recommendation systems and achieves good results on several recommendation system datasets. Based on the work done by [68], AGAE can be naturally applied for recommendation systems and have the potential to further improve the GAE recommendation model proposed in [68].
List of References


