Graph Attention Networks with Auxiliary Positional Embedding Models

Liheng Ma
School of Computer Science
McGill University, Montreal

November, 2020

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science in Computer Science

November, 2020

Abstract

Large-scale data collections represented by graphs are everywhere: social networks, chemical molecules, protein networks, the world wide web, and many more examples. The ubiquity of such graph-structured data calls for developing methods to understand and utilize information represented under this form. Graph Neural Networks (GNNs) are deep-learning based methods which provide the current state of the art performance in machine learning on graphs. In this work, we analyze the limitation of Graph Attention Networks (GAT), a common variant of GNNs, to fully explore the structural information within each locality on graphs, which hinders their ability to learn on non-homophilic graphs. Inspired by the positional encoding in the Transformers, a deep learning model of choice in many natural language processing tasks, we propose a framework, termed GAT-POS, to enhance GATs by incorporating node embeddings from an auxiliary embedding model, which learns the structural and positional information on graphs. In this framework, the embedding model is plugged into the standard GAT's architecture as an auxiliary model, and the supervised task of predicting node labels with the main model and the unsupervised task of predicting graph context with the auxiliary model are trained jointly with the same architecture. Experimental results show that the proposed GAT-POS achieves comparable performance as typical GNNs (GATs and GCNs) on homophilic datasets and outperforms typical GNNs and the recently introduced Geom-GCN, which is an embedding-enhanced GNN, on non-homophilic datasets.

Abrégé

Les collections de données à grande échelle représentées par des graphiques sont partout : réseaux sociaux, molécules chimiques, réseaux de protéines, le web, et bien d'autres exemples encore. L'omniprésence de ces données structurées par des graphiques nécessite le développement de méthodes pour comprendre et utiliser les informations représentées sous cette forme. Les réseaux neuronaux de graphes (GNN) sont des méthodes basées sur l'apprentissage profond qui fournissent les performances actuelles de l'état de l'art dans l'apprentissage machine sur des graphes. Dans ce travail, nous analysons les limites des Graph Attention Networks (GAT), une variante commune des GNN, pour explorer pleinement les informations structurelles dans chaque localité sur les graphes, ce qui limite leur capacité à apprendre sur des graphes non-homophiles. Inspirés par l'encodage positionnel des Transformers, un modèle d'apprentissage profond de choix dans de nombreuses tâches de traitement du langage naturel, nous proposons un cadre, appelé GAT-POS, pour améliorer les GAT en incorporant des représenta-tions structurelles de nœuds à partir d'un modèle de représentations auxiliaire, qui apprend les informations structurelles et positionnelles sur les graphes. Dans ce cadre, le modèle de représentations structurelles et positionelles est intégré à l'architecture standard de la GAT en tant que modèle auxiliaire, et la tâche supervisée de prédiction des étiquettes de nœuds avec le modèle principal et la tâche non supervisée de prédiction du contexte du graphe avec le modèle auxiliaire sont formées conjointement avec la même architecture. Les résultats expérimentaux montrent que le GAT-POS proposé atteint des performances comparables à celles des GNN typiques (GAT et GCN) sur des ensembles de données homophiles et surpasse les GNN typiques et le Geom-GCN récemment introduit, qui est un GNN amélioré par l'intégration, sur des ensembles de données non-homophiles.

Acknowledgements

First and foremost, I would like to greatly thank my supervisors, Dr. Adriana Romero Soriano and Dr. Reihaneh Rabbany, for giving me unwavering support and instruction to my Master studies and research. I am tremendously grateful for their countless and tireless help and guidance on my research.

My deep and sincere gratitude goes to my parents for their incredible love and support.

Many thanks also go to my friends for providing support and welcome distraction: Yilan, LC, JY, Yichen, Yang, Yingxue and Chen.

Sincerely,

Liheng Ma

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Graph Structured Data

Entities in the world do not exist independently of each other. Instead, there always exist interactions and relationships among them. There are several examples in the real world with such interactions and relationships among entities, such as atoms in molecules, neurons in brains, individuals in societies, etc. Such interactions and relationships between entities are usually referred to as *structures*. Accordingly, the data collected via the observation of entities with structure is termed *structured data*. A natural and ubiquitous way to describe a collection of structured data is to represent them with a network, or its mathematical term, a graph, in which the entities are represented as *nodes* while the pairwise interactions or relationships between entities are represented as *edges*.

With the rapid growth of the internet and modern technology, large-scale data collections represented by graphs are everywhere: social networks, chemical molecules, protein networks, purchasing records, the world wide web, and many more examples. The ubiquity of such graph-structured data calls for developing methods to understand and utilize information represented under this form.

1.2 Deep Learning on Graphs

With plentiful data, people intend to learn from those data and develop models and algorithms that improve with experience, and assist humans in making predictions and decisions with those intelligent systems. The study of such algorithms and systems is usually referred to as the field of *machine learning*.

Among machine learning methods, *deep learning* approaches have been enjoying tremendous popularity in the past decade. Models in deep learning tackle the learning problem by learning representations of the input data jointly with a predictive model for the specific downstream task. Those models usually are built by stacking multiple layers of differentiable non-linear transformations and trained in an end-to-end learning process. The class of models in deep learning is referred to as *deep neural networks*.

The vanilla neural networks (i.e., Multilayer perceptrons (MLP)) learn from a collection of entities independently. However, considering the natural existence of structures, researchers have attempted to develop neural networks that exploit the spatial and structural information with entities. Therefore, instead of considering each example as an individual entity, a collection of entities and the structure among them is considered as an example (e.g., an image with multiple pixels in image classification, a sentence with words in sentiment analysis). Alternatively, one can consider an example as an entity with interactions with other entities (e.g., a pixel in an image in semantic segmentation, a user or an item in the recommendation tasks).

Currently, several architectures of neural networks have been proposed to learn from structured data, which is embedded in some Euclidean spaces, such as grid structures and sequence structures[1]. For instance, Recurrent Neural Networks (RNNs) are proposed to work on data with sequential structures [Rumelhart et al., 1986]. RNNs are compatible with input sequences with different lengths by taking in entities in each sequence recur-

[1]Since grid structures and sequence structures have been termed lattice graphs [Acharya and Gill, 1981] and sequence graphs [Alekseyev and Pevzner, 2009] respectively, they can be viewed as special cases of graphs.

rently. As another example, Convolutional Neural Networks (CNNs) are well-defined and proven to be efficient on data with sequential structures [Collobert and Weston, 2008] and grid-like structures [LeCun et al., 1989], based on their shared-weights architecture and translation invariance characteristics.

However, a lot of structured data cannot be represented by structures embedded in Euclidean spaces. Thus, there has been an increasing interest in generalizing deep learning methods to non-Euclidean structured data represented by graphs recently. Graph Neural Networks (GNNs) [Frasconi et al., 1998, Sperduti and Starita, 1997, Gori et al., 2005, Scarselli et al., 2008, Bruna et al., 2014, Kipf and Welling, 2017, Velickovic et al., 2018, Hamilton et al., 2017, Monti et al., 2017, Defferrard et al., 2016, Atwood and Towsley, 2016, Duvenaud et al., 2015, Li et al., 2015] are arguably among the most widely used deep neural network architectures specifically designed to operate on graph-structured data and have been proven effective on many graph-structured domains. Nevertheless, GNNs still have several limitations in terms of stability and expressive power. The architecture design of GNNs is challenging considering the properties of graphs in relation to the varying number of the spatial coordinates (i.e., dependent on the number of the observed examples), permutation invariance of adjacent nodes, and different sized neighborhoods.

In this thesis, we analyze some shortcomings of GNNs, with a focus on convolutional-based GNNs, and propose a framework to enhance the Graph Attention Network (GAT), which is one of the top performing GNNs. Presently, most typical GNNs are categorized into spectral approaches and spatial approaches [2].

On the one hand, spectral approaches originated from the extension of convolutions to general graphs, termed spectral convolutions, by finding the corresponding Fourier basis given by the eigendecomposition of the graph Laplacian (i.e., a matrix representation of a graph) [Bruna et al., 2014]. However, this extension is based on the operators introduced for similarity graphs [Belkin and Niyogi, 2002, Chung and Graham, 1997]. The spectral

[2] A comprehensive introduction will be included in section 2.2.

convolution and its localized approximations [Defferrard et al., 2016, Kipf and Welling, 2017] are under the restriction of exploring the underlying group invariances of similarity graphs.

On the other hand, spatial approaches aim to define operators on the neighborhood of each node, as a local area, with shared-weights architectures and translation invariance alike CNN. Due to the various coordinates and permutation invariance of neighboring nodes, the operators are usually defined as pooling operations and their variants (e.g., mean-pooling, max-pooling [Hamilton et al., 2017] and sum-pooling [Xu et al., 2019]). Alternatively, the operators can be defined as adaptive filters with coefficients parameterized as content-based weighting functions for stronger expressive power [Monti et al., 2017, Velickovic et al., 2018].

Even though GNNs are working well on learning node representations on graphs with high homophily and haven been proven to capture graph structures in a manner similar to Weisfeiler-Lehman (WL) algorithm [Weisfeiler and Lehman, 1968], most GNNs are based on the assumption that edges indicate the similarity between node pairs and learn representations of nodes exploiting the group structure introduced by the graphs. Such characteristics of GNNs might lead to the oversmoothing problem [Li et al., 2018, Zhao and Akoglu, 2020] when learning semantic representations of nodes, as learned representations of nodes within the same connected component tend to be close. Thus, when the structure only indicates the existence of interactions or relationships (beyond similarity), such as several words within a sentence, typical GNNs might fail to learn it. One exception is worth mentioning that a number of GNNs considering edge attributes [Murphy et al., 2019, Gilmer et al., 2017] can go beyond such limitations. However, those GNNs only work on graph datasets providing edge attributes while there are numerous datasets in the real world without edge attributes, which are the focus of this thesis and the aforementioned GNNs.

Inspired by the positional encoding in the Transformer [Vaswani et al., 2017], a well-known language model based on neural attention mechanisms, we propose a framework to enhance GATs by incorporating node embeddings from an embedding model, which learns the structural and positional information of nodes on graphs. We call the proposed framework *Graph Attention Networks with Auxiliary Positional Embedding Models* (GAT-POS). With inspirations from the Transformer [Vaswani et al., 2017] and the comparison between attention mechanism and convolutions on images [Cordonnier et al., 2020, Zhao et al., 2020], we propose two versions of graph attentional layer incorporating the learned positional embeddings in different ways. We also introduce embedding variants which properly take advantage of the receptive fields of GNNs. Furthermore, in order to support end-to-end training and better train the embedding model, we adopt a joint training scheme inspired by [Weston et al., 2008] that connects the auxiliary positional embedding model with the main architecture, plugs the unsupervised learning objective into the corresponding layers of the auxiliary model, and trains the supervised tasks (i.e., predicting node labels) and unsupervised tasks (i.e., learning graph context) jointly using the same architecture simultaneously.

Our framework has been evaluated on multiple datasets, including citation networks [Yang et al., 2016, Kipf and Welling, 2017, Velickovic et al., 2018], as examples of graph-structured datasets with high homophily, as well as actor co-occurrence networks and Wikipedia networks, as the representatives of non-homophily graph-structured datasets. Furthermore, we have investigated the benefits and shortcomings of different versions of the proposed GAT-POS, and have discussed the potential directions to improve the proposed approach as future works to further address the discussed shortcomings.

Chapter 2

Background

2.1 Notation

This section provides a reference for commonly used notation across the thesis (Table 2.1). Additional notation for individual chapters will be introduced where necessary.

Table 2.1: Notation Table

Example	Explanation
x, y, z	A lowercase italic letter denotes a scalar or scalar-valued random variable.
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	a lowercase bold letter denotes a vector or vector-valued random variable.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	a uppercase bold letter denotes a matrix or matrix-valued random variable.
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	Calligraphic letters typically denote sets. An exception is $\mathcal{L}$ which is used to denote the scalar-valued objective functions.
$f(\cdot; \theta), \omega(\cdot; \theta)$	A function with parameter dependency denoted with a greek letter θ or ϕ. The parameter dependency can be dropped for simplicity.

x_i or $\mathbf{x}_{(i)}$	The i-th element of vector $\mathbf{x}$.
$\mathbf{X}_{(i,j)}$	The i, j-th element of matrix $\mathbf{X}$.
$\mathbf{I}_d$	The $d \times d$ identity matrix.
$\mathrm{diag}(\mathbf{x})$	A diagonal square matrix with entries on the diagonal as the elements of the vector $\mathbf{x}$.
$\mathbb{R}^d$	The d-dimensional real space.
$\sigma(\mathbf{x}), \sigma(\mathbf{X})$	The element-wise non-linear activation function on vector $\mathbf{x}$ or matrix $\mathbf{X}$.
$\mathrm{Softmax}_{\mathcal{X}}(\mathbf{x}_i)$	The softmax function defined as $\frac{\exp(\mathbf{x}_i)}{\sum_{\mathbf{x}_j \in \mathcal{X}} \exp(\mathbf{x}_j)}$
$[\mathbf{x} \Vert \mathbf{y}]$	The concatenation of two vectors $\mathbf{x} \in \mathbb{R}^{d_x}$ and $\mathbf{y} \in \mathbb{R}^{d_y}$ that the $[\mathbf{x} \Vert \mathbf{y}] \in \mathbb{R}^{d_x + d_y}$
$[\mathbf{x}, \mathbf{y}]^\intercal$	The stacking of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ that $[\mathbf{x}, \mathbf{y}]^\intercal \in \mathbb{R}^{2 \times d}$
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A graph $\mathcal{G}$ with a set of vertices (nodes[1]) $\mathcal{V}$ and a set of edges $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$; by default, we consider undirected and unweighted graphs.
$(v, u) \in \mathcal{E}$	The undirected edge between vertex v and u; $(v, u) \Leftrightarrow (u, v)$.
$(v \rightarrow u) \in \mathcal{E}$	The directed edge from vertex v to u; $(v \rightarrow u) \not\Leftrightarrow (u \rightarrow v)$
$\mathbf{A} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{V}\rvert}$	The adjacency matrix representing the $\mathcal{E}$ in $\mathcal{G}$; By default, $\mathbf{A}_{(v,u)} = 1$ iff $(v, u) \in \mathcal{E}$ or $(v \rightarrow u) \in \mathcal{E}$ and $\mathbf{A}_{(v,v)} = 0$
$d_v, \forall v \in \mathcal{V}$ $\mathbf{D} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{V}\rvert}$	d_v is the node degree of node v that $d_v := \sum_u \mathbf{A}_{(v,u)}$; The diagonal matrix $\mathbf{D}$ is the degree matrix of the graph such that $\mathbf{D}_{(v,v)} := d_v$
$\mathbf{x}_i, \mathbf{y}_i, \mathbf{h}_i \in \mathbb{R}^m$	The input, output and hidden vector representations of m dimension of vertex (example) i of a model or a module.

[1]Note that in the thesis we use the terms vertex and node interchangeably

$\mathbf{X}, \mathbf{Y}, \mathbf{H} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times m}$	The input, output and hidden matrix representations of all vertices (examples) of a model or a module, in which each rows is a vector representation of m-dimension.
$\mathcal{N}(v) : \mathcal{V} \rightarrow 2^{\mathcal{V}}$	A function return a set of nodes $\{u \mid (v, u) \in \mathcal{E}\}$ given $v \in \mathcal{V}$ as the argument.

2.2 Graph Neural Networks

Representation learning on graphs is one of the most rapidly growing domains in the world of machine learning. Among numerous models in the domain, Graph Neural Networks (GNNs) [Frasconi et al., 1998, Sperduti and Starita, 1997, Gori et al., 2005, Scarselli et al., 2008, Bruna et al., 2014, Kipf and Welling, 2017, Velickovic et al., 2018, Hamilton et al., 2017, Monti et al., 2017, Defferrard et al., 2016, Atwood and Towsley, 2016] are a class of models with novel network architectures specially designed for learning the rich structure in the graph-structured data. Considering the central importance of GNNs in this thesis, a comprehensive introduction will be given in the following.

2.2.1 Preliminaries

Problem Statement

Given a graph $G = (\mathcal{V}, \mathcal{E})$ with a set of n vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$, the goal is to learn representations for the vertices in $\mathcal{V}$ considering the structure of the graphs.

Inputs as Graphs Under an arbitrary fixed order of vertices, the vertices $\mathcal{V}$ can be represented by a corresponding attribute matrix [2] $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\intercal \in \mathbb{R}^{n \times d}$, where $\mathbf{x}_v \in \mathbb{R}^d$

[2] Note that, for clarification, we denote the input features of nodes as node attributes. In the thesis, the term node features may refer to the output of one filter applied to the previous layer corresponding to the concept of feature maps in CNNs on images.

is a d-dimension representation of node v, describing the attributes of the node. In some cases (e.g., in graph signal processing), the attribute matrix might downgrade to a vector if the input attribute of each node is a scalar.

Under the same order, the edges can be represented by an adjacency tensor $\mathbf{A} \in \mathbb{R}^{n \times n \times d'}$, in which, $\mathbf{A}_{(v,u,\cdot)}$ is the representation of node-pair of v and u. We denote the unordered node-pair as $(v, u) = (u, v)$ and the ordered node-pair from v to u as $(v \rightarrow u)$. Usually, when $(v, u) \notin \mathcal{E}$ (or similarly $(v \rightarrow u) \notin \mathcal{E}$), $\mathbf{A}_{(v,u,\cdot)}$ should be an all-zero vector.

In this thesis, we look at simple undirected graphs without edge attributes and focus on the case of node-level representation learning for node classification tasks. Node attributes are assumed available in our scenario unless otherwise stated. Accordingly, we consider the adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$, such that $\mathbf{A}_{(v,u)} = \mathbf{A}_{(u,v)} = 1$ iff[3] $(v, u) \in \mathcal{E}$. Correspondingly, the neighborhood is defined as $\mathcal{N}(v) = \{u | (v, u) \in \mathcal{E}, u \in \mathcal{V}\}$ for $v \in \mathcal{V}$.

Note that, unless otherwise stated, we assume that there is no self-loop by default in the graphs as well as our definition of $\mathcal{N}(\cdot)$. However, adding a self-loop for each node in the graph and including the center-node in each neighborhood is not uncommon [Kipf and Welling, 2017].

Representation Learning and Downstream Tasks Given the inputs, the goal of graph representation learning can be formulated into a more general concept of learning representations for target subgraphs[4] induced by the corresponding node-sets with manually set sizes. The sizes of subgraphs control the specific tasks of learning, with examples as follows:

- when the sizes are one, the task is denoted as learning representations for vertices (node level);
- when the sizes are two, the task is learning representations for links or node-pairs (edge level / node-pair level);

[3]The shortened of if and only if.

[4]The nodes and the edges associated with them.

- when the sizes are considered as infinity or the sizes of input graphs, the task is learning representations for graphs (graph level);

Currently, most GNNs focus on the node-level representation learning (i.e., learning node representations) and, for multiple downstream tasks such as graph classification and node classification, plug-in a readout function on the learned node representations as the output layer. The node representations can be trained jointly with the output layer for the downstream tasks in a supervised or semi-supervised manner. Alternatively, the node representations can be pretrained in an unsupervised manner, for predicting graph contexts, and then finetune a plug-in output layer for the downstream task with the learned node representations frozen or active.

In the thesis, we focus on the node-level representation learning and mainly introduce GNNs for node classification tasks. Overall, we mainly consider the models supporting an end-to-end training process, where the overall GNN-framework is trained with respect to all downstream tasks simultaneously.

Supervised Learning and Semi-supervised Learning Typically, node classification tasks can be trained and evaluated in either the supervised, the semi-supervised or the unsupervised setting. In the supervised setting, the training examples (i.e., nodes) and their corresponding labels are observed during the training stage while the test examples and validation examples are only observed during evaluation. In contrast, unsupervised learning allows the model to observe all examples, but no labels are available during training. Semi-supervised learning falls between supervised learning and unsupervised learning; in this setting, models are allowed to observe the training examples and their labels as well as some unlabeled examples during training.

Besides, there are some models learning representation for nodes in an unsupervised manner. After getting the representations, an extra prediction module taking those representations as inputs is trained in a supervised manner with observed examples as a separate model or fine-tuning layers [5]. This is common when evaluating the unsupervised learning model with downstream tasks [Hamilton et al., 2017], and supposed to be considered under the supervised or semi-supervised setting for the whole models.

In the thesis, we focus on the semi-supervised setting where the models are trained with labeled and unlabeled nodes.

Inductive and Transductive Setting By convention, in supervised learning, the examples for evaluation will not be observed in the training stage. By contrast, under the unsupervised setting, the model will observe all examples without labels and, for instance to infer the clustering pattern or detect the anomalies in the data. However, in the semi-supervised setting, models can observe labeled examples and unlabeled examples in the training stage. Thus, under the semi-supervised learning, there exist two types of settings, the inductive and the transductive settings. The main difference between them is that in the inductive setting, models need to predict the labels of examples unobserved during training, while under the transductive setting, models will predict the labels for unlabeled examples observed in the training stage.

Note that for graph representation learning, there are two types of cases under the inductive setting. One is that the unseen examples are within the new observed connected components in the graph, such as node classification tasks on *protein-protein interaction dataset* (PPI) discussed in [Velickovic et al., 2018]. Thus, they will not affect the graph structure of the previously observed examples. From the other perspective, this case can be viewed as training GNNs on one or several graphs and evaluating the models on some other graphs.

The other case is more challenging in that the unobserved examples have edges connected to the previously observed nodes in the evaluation stage, such as tasks on the *Reddit* dataset presented in [Hamilton et al., 2017]. In this case, the graph structure of previously observed nodes will be changed during evaluation. In other words, needless to

[5] In the case of fine-tuning layers, the original models learning representation can be either frozen or trained during supervised learning part.

say that GNNs have to generalize to unseen examples, even the predicted labels for those previously observed examples are not consistent across the training and the evaluation stages, due to the change of the graph structure associated with those examples.

Inspirations of Graph Neural Networks

Neighborhood as Spatial Locality Inspired by CNNs, a GNN tends to construct on filters exploiting spatial locality based on translation-invariance with the shared-weights architectures, which allow parallel computation and fewer parameters to train. Instead of a square region around a center pixel in the case of images as an example, each locality in a graph is controlled by the edges, termed neighborhood.

The neighborhood of node v, $\mathcal{N}(v) : \mathcal{V} \rightarrow 2^{\mathcal{V}}$, is defined as the set of nodes adjacent to v, $\{u|(v,u) \in \mathcal{E}\}$. In some cases, the definition may also refer to the subgraph induced by those adjacent nodes, with the edges between nodes preserved. If $\mathcal{N}(v)$ is defined to include v, it is called a closed neighborhood, and otherwise called an open neighborhood.

In fact, the neighborhood is extendable to k-hop ($k \geq 0$), in which the adjacent nodes is extended to the set of nodes reachable in k-hop through a path (i.e., the nodes u with the shortest path of k from or to node v).

We further define that if $k = 0$, the neighborhood denotes the center nodes itself and the GNN layer will be downgraded to a fully connected layer shared by all nodes. On the other hand, if $k \rightarrow \infty$, the neighborhood is the connected component which the target node belongs to. Unless otherwise stated, we refer the terms neighborhood to the first-hop neighborhood.

Considering the properties of different sized neighborhoods and permutation invariance, a filter of GNNs is expected to be compatible with different sized localities and satisfy the permutation invariance to the input order.

Neural Message Passing Another inspiration of GNNs comes from message-passing algorithms on loopy graphs (e.g., Loopy Belief Propagation [Yedidia et al., 2003]), which are widely used in probabilistic graphical models (PGMs). In message-passing algorithms, a state of a node v will be updated conditional on the message from its adjacent nodes (first-hop neighborhood) iteratively till convergence. Message-passing algorithms are computationally efficient, thanks to the localized propagation procedure [Weiss, 2000]. Weight-sharing architecture is also feasible by introducing parameterized potential based on neural networks [Kim et al., 2017].

Most GNNs, especially spatial GNNs, have a similar update rule that a node's representation is updated with the information aggregated from its neighborhood.

Thus, [Gilmer et al., 2017] proposed a common framework called Message Passing Neural Networks (MPNNs) which includes a large number of GNNs in the literature. Those GNNs can be formulated as follows,

$$\begin{aligned} \mathbf{m}_i^{t+1} &= \sum_{j \in \mathcal{N}(i)} M_t\left(\mathbf{h}_i^t, \mathbf{h}_j^t, \mathbf{e}_{ij}\right) \\ \mathbf{h}_i^{t+1} &= U_t\left(h_i^t, m_i^{t+1}\right) \end{aligned} \tag{2.1}$$

where $\mathbf{e}_{ij}$ denotes the attribute of the edge between node i and j, if available; $\mathbf{h}_i^t$ the representation of node i at timestep t or layer t; $M_t(\cdot), U_t(\cdot)$ can arbitrary functions.

Accordingly, those GNNs following the similar update rules to message passing algorithms are also termed as Neural Message Passing algorithms or Neural Message Passing GNNs (NMP-GNNs).

Weisfeiler-Lehman Algorithm and Isomorphism Test Weifeiler-Lehman algorithm (WL-algorithm) and Weisfeiler-Lehman Isomorphism Test (WL-Test) are also considered to be with a close relationship with GNNs. WL-test is designed to identify whether two graphs are considered isomorphic, that there exists a mapping between nodes of these two graphs which preserves nodes adjacencies.

WL-algorithm also holds an iterative updating rule till convergence. In each iteration, for a node v, its adjacent nodes' labels will be considered as a multiset and compressed by injective functions (i.e., hash function) to update the label of node v. Typically, each node will be initialized with the same label (i.e., 1) and the above iteration will be repeated for up to N (the number of nodes) iterations or convergence. After this procedure, if two graphs have different partitions of nodes by final compressed labels (i.e., represented as the number of nodes with each compressed label), these two graphs are not isomorphic. Note that WL-algorithm cannot guarantee that two graphs with the same partitions of nodes are isomorphic.

Many spatial GNNs can be viewed as doing WL-algorithm with parameterizing the compress functions as Neural Networks (NN) and termed as WL-GNNs. There are also some GNNs inspired by higher-order WL-test which extends operations on nodes to subgraphs, in order to enhance the ability to capture a structural representation of the graph [Morris et al., 2019, Maron et al., 2019].

Summary In many works in the literature, the WL-GNNs and NMP-GNNs are considered to be the same type of GNNs. However, their perspectives are different. For updating a node's representation, WL-GNNs usually emphasize how to encode its closed or open neighborhood represented as a multiset, while the NMP-GNNs focus on how a message is passed to the node from its neighbors. Thus, WL-GNNs have a close relationship to deep learning on sets, while NMP-GNNs are usually viewed as pairwise learning models similar to message-passing algorithms.

Note that, almost all GNNs can be viewed as learning higher-level features of a node by applying a linear or non-linear feature extractor on its (k-hop) neighborhood as generalized convolutional filters. However, even though a large amount of GNNs can be interpreted as WL-GNNs or neural message-passing algorithms, some GNNs are not able to be interpreted through these perspectives. For instance, GNNs aggregate information from higher-hop or even infinite-hop neighborhoods such as [Bruna et al., 2014, Defferrard et al., 2016] and GNNs with a non-symmetric filter on each neighborhood (GraphSAGE-

LSTM) [Hamilton et al., 2017] cannot be understood as WL-GNNs or neural message passing algorithms.

Convolution-based and Recurrent-based Graph Neural Networks

The abovementioned inspiring algorithms have two architectures. CNNs stacks multiple layers to build a non-linear filter overall, while WL-algorithm and message-passing algorithms will repeat some operations iteratively until convergence or reaching some criteria.

As a consequence, there are two genres of the overall architectures of GNNs, Convolutional GNNs, and Recurrent GNNs.

Convolutional Graph Neural Networks Inspired by CNNs, convolutional GNNs treat the operation of aggregating information from the neighborhood (a.k.a message passing) as a generalized convolutional filter on graphs. A GNN is built on stacking several layers of such filters with non-linear activation functions. Note that each layer has separate parameters to train, probably with different input and output dimensions.

Such like CNNs, stacking of multiple locally convolutional GNN filters with non-linear activation function will lead to non-linear filters that are increasingly global. In other words, the networks start from representations of small parts of the input and then assemble representations of larger areas from them.

Most GNNs [Bruna et al., 2014, Defferrard et al., 2016, Kipf and Welling, 2017, Monti et al., 2017, Hamilton et al., 2017, Velickovic et al., 2018] introduced in this thesis will be of this genre.

Recurrent Graph Neural Networks Inspired by methods with iterative updates, recurrent GNNs [Sperduti and Starita, 1997, Frasconi et al., 1998, Gori et al., 2005, Scarselli et al., 2008, Li et al., 2015, Yoon et al., 2019] adopt the architecture of recurrent neural networks (RNNs). An iteration of neural message passing is considered as an RNN cell and the node representations will be updated iteratively as the hidden states in RNNs. Note that, unlike convolution GNNs, recurrent GNNs repeat the same cells multiple times. The recurrent-based GNNs will not stop iterating until reaching some stop criteria, such as convergence and stop signals from another recurrent GNN as a controller. This process is usually computationally expensive and not scalable to large-scale graphs.

Overall, the ability of filters in GNNs to encode neighborhoods is not affected by the convolutional or recurrent architectures. Moreover, recurrent GNNs and convolutional GNNs are actually interchangeable by switching stacking filters and iterating filters.

In this thesis, we will have a focus on covering the literature related to convolutional GNNs, which are more common and more scalable, and have a comprehensive analysis of the advantages as well as the shortcomings of each filter in convolutional GNNs.

2.2.2 Spectral Graph Neural Networks

Spectral Convolutional Networks on Graphs

In learning representations with the spatial structures, CNNs are well-defined and extremely efficient on structures in Euclidean spaces, such as grids in image tasks and sequences in audio recognition tasks, based on the spatial convolutional theorem.

Thus, generalizing CNNs to non-euclidean structures, like graphs, is a significant step for designing the architecture of a deep model for graphs.

[Bruna et al., 2014] firstly introduces a spectral construction of the convolutional networks via the spectrum of graph Laplacian inspired by the Graph Fourier transform, which is widely used in spectral clustering and graph signal processing [Von Luxburg, 2007,Chung and Graham, 1997].

Spectral Convolutions on Grids and Graphs First, the spectral convolutions are constructed for grids. The structure of an image is represented as a weighted similarity graph $G = (\mathcal{V}, \mathcal{E})$ with n nodes (i.e., $\mathbf{A}_{vu} \geq 0, \forall v, u \in \mathcal{V}$). Note that representing an image as an unweighted graph can be an alternative option. By representing an image as a graph, one can reach the unnormalized graph Laplacian (combinatorial Laplacian) defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ [6], where $\mathbf{D}$ is the degree matrix with its diagonal $\mathbf{D}_{vv} = \sum_{u \in [1,n]} \mathbf{A}_{vu}$. The graph Laplacian is a matrix representation of an undirected graph which can be used to find useful properties of the graph.

With the graph Laplacian, we can transform the nodes' signals in the graph from the vertex domain to the spectral domain via graph Fourier transform, which is analogous to transforming signals from the time domain to the frequency domain with classical Fourier transform.

For better understanding, we consider encoding the nodes under the assumption that nodes which are "close" (i.e., $\mathbb{A}_{vu}$ is large) should have a "similar" label. This assumption is also known as the smoothness assumption. The goal is to find a mapping to encode the nodes in the graphs interpreting the smoothness introduced by the graph.

We assume that the mapping consists of several labeling functions. A labeling function is defined as $f : \mathcal{V} \to \mathbb{R}$ for encoding nodes in the graph . For simplicity, each labeling function is represented by a vector $\mathbf{m} = [m_1, \ldots, m_n], s.t. \, m_v = f(v), \forall v \in \mathcal{V}$, termed labeling vector. A natural definition of the smoothness introduced by the labeling function on the graph, $||\nabla f||^2_{\mathbf{A}}$ is defined as follows,

$$||\nabla f||^2_{\mathbf{A}} = \sum_v \sum_u \mathbf{A}_{vu} [m_v - m_u]^2 = \mathbf{m}^\intercal \mathbf{L} \mathbf{m} \tag{2.2}$$

In addition, we add an extra constraint to the labeling vectors such that $||\mathbf{m}|| = 1$ and labeling vectors are orthogonal to each other, in order to get rid of redundant encoding. Then, the smoothest labeling vector is a constant vector:

$$\mathbf{v}_0 = \underset{\mathbf{m} \in \mathbb{R}^n, ||\mathbf{m}||=1}{\arg\min} ||\nabla f||^2_{\mathbf{A}} = (1/\sqrt{n}) \odot \mathbf{1}_n, \tag{2.3}$$

[6]The symmetric normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ can be an option as well.

where $\mathbf{1}_n \in 1^n$ is an all-one vector of dimension n. And each successive labeling vector is defined as follows,

$$\mathbf{v}_i = \underset{\mathbf{m}\in\mathbb{R}^n, ||\mathbf{m}||=1, \mathbf{m}\perp\{\mathbf{v}_0,\ldots,\mathbf{v}_{i-1}\}}{\arg\min} ||\nabla f||^2_{\mathbf{A}}. \tag{2.4}$$

With the extra constraints, the above equations can be solved by the eigendecomposition of the graph Laplacian $\mathbf{L} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\intercal$, where $\mathbf{V}$ is a matrix consisting of n eigenvectors while $\mathbf{\Lambda}$ is a diagonal matrix whose diagonal is a vector of eigenvalues corresponding to $\mathbf{V}$ ($\mathbf{\Lambda}_{(i,i)} = \lambda_i$).

Then the smoothness introduced by each labeling function is $||\nabla f||^2_{\mathbf{A}} = \mathbf{m}^\intercal\mathbf{L}\mathbf{m} = \lambda\mathbf{m}^\intercal\mathbf{m} = \lambda$, where $\mathbf{m}$ is an eigenvector of $\mathbf{L}$ and λ is the corresponding eigenvalue. Thus, via reordering the $\mathbf{V}$ according to the eigenvalue, one can get an eigenvector-matrix $\mathbf{V} = [v_0, \ldots, v_{n-1}]^\intercal \in \mathbb{R}^{n\times n}$ corresponding to aforementioned labeling vectors.

It is worth noting that, with the aforementioned extra constraints, $\mathbf{I}_n = \mathbf{V}\mathbf{V}^\intercal$, where $\mathbf{I}_n \in \mathbf{R}^{n\times n}$ is an identity matrix. Thus, for a node signal $\mathbf{x} \in \mathbb{R}^n$ (each node has a scalar signal), $\mathbf{x} = \mathbf{V}\mathbf{V}^\intercal\mathbf{x}$ is held. In other words, with $\mathbf{V}$, one can transform a node signal from the vertex domain to the spectral domain and vice versa.

Therefore, we can construct spectral convolutions with a filter defined in the spectral domain, based on $\mathbf{V}$. It is worth mentioning that the spectral construction of convolutions can be directly generalized to any structure as long as it can be represented by a graph Laplacian. Thus, it is straightforward to apply the spectral convolutions to general graphs.

Inspired by the fact that the eigenvalues can modulate the smoothness, the authors define a filter $\mathbf{F} \in \mathbb{R}^{n\times n}$ as a diagonal matrix with parameters on its diagonal. Then the spectral convolutions transform input signals $\mathbf{x} \in \mathbb{R}^n$ to output signals $\mathbf{y} \in \mathbb{R}^n$ with a filter $\mathbf{F}$ as follows,

$$\mathbf{y} = \mathbf{V}\mathbf{F}\mathbf{V}^\intercal\mathbf{x}. \tag{2.5}$$

Spectral Convolutional Neural Networks With those on hand, the spectral convolutional networks can be constructed based on spectral convolutions for a graph with multi-dimensional node attributes. The filter of the spectral convolutional networks at a layer transforms an input $\mathbf{X} \in \mathbb{R}^{n \times F_{in}}$ to an ouput $\mathbf{Y} \in \mathbb{R}^{n \times F_{out}}$,

$$\mathbf{Y}_{\cdot,j} = \sigma\left(\mathbf{V} \sum_{i=1}^{F_{in}} \mathbf{F}_{ij} \mathbf{V}^{\intercal} \mathbf{X}_{\cdot,i}\right), \text{for } j = 1 \ldots F_{out}, \tag{2.6}$$

where $\mathbf{V}$ is a matrix of eigenvectors of the graph Laplacian[7], ordered by eigenvalues in descending; $\mathbf{F}_{ij}$ is a learnable diagonal matrix modulating the smoothness introduced by the corresponding eigenvectors; and σ is arbitrary nonlinear activation function.

The spectral convolutional networks support subsampling as well. Since only the first d eigenvectors in descending of the graph Laplacian are useful in practice, one can utilize $\mathbf{V_d}$ instead, which consists of the first d columns of $\mathbf{V}$. The cutoff frequency of d depends on the intrinsic regularity of the graph, as well as the target sample size.

However, spectral CNNs have three shortcomings. The first is that the filters of spectral convolutions are not localized with respect to the spatial structure, requiring the whole graph in computation. Consequently, the computation of the eigendecomposition with a complexity of $O(n^3)$ as well as the matrix multiplications with a complexity of $O(n^2)$ prevents spectral CNNs from being scalable. Moreover, without the localized constraint introduced by subsampling, the spectral convolutions tend to filter delocalized templates. The second shortcoming is the matrix of eigenvectors transforming signals to the spectral domain is sensitive to the perturbation of the graph structure due to the global computation, limiting the capacity of generalization to unseen nodes under the inductive setting. The third shortcoming is that the construction of spectral convolution is based on the assumption that the edges indicate similarity and the graph has an underlying group invariance. Therefore, the filters might not work if the graph does not have a group structure or the group structure does not commute with the Laplacian. Furthermore, when the spectral CNNs are applied to graphs, in which an edge only indicates the

[7]For a general graph, the symmetric normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is preferred, considering the computational issues on irregular graphs [Chung and Graham, 1997].

existence of interaction rather than the similarity, the theoretical foundation is not solid anymore.

Fast Localized Spectral Filters

In order to tackle the first shortcoming of spectral CNNs [Bruna et al., 2014], [Defferrard et al., 2016] modify the spectral filter to a localized filter in a ball of radius K (K-hops from the central vertex) by a K-order approximation based on Chebyshev polynomial. The symmetric normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is considered here.

In [Bruna et al., 2014], a filter with parameters $\theta \in \mathbb{R}^n$ is directely defined on the spectral domain as $g_\theta = \text{diag}(\theta)$ where diag : $\mathbb{R}^n \to \mathbb{R}^{n\times n}$ is a function to construct a diagonal matrix whose diagonal is the argument. The spectral convolutions transform a node signal $\mathbf{x} \in \mathbb{R}^n$ to $\mathbf{y} \in \mathbb{R}^n$ as follows:

$$\mathbf{y} = (\mathbf{V} g_\theta \mathbf{V}^\top)\mathbf{x} \tag{2.7}$$

where $(\mathbf{V} g_\theta \mathbf{V}^\top) \in \mathbb{R}^{n\times n}$ can be understood as a transformation matrix (between nodes) controlling how signal from a node is transmitted to another node. We can understand such a transformation matrix is a function of the graph Laplacian and a filter defined on the spectral domain.

In [Defferrard et al., 2016], the filter is understood as a function of the eigenvalues of $\mathbf{L}$, denoted as $g_\theta(\mathbf{\Lambda})$. In order to reach strictly localized filters, filter is defined as,

$$g_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k, \tag{2.8}$$

where $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_{K-1}] \in \mathbb{R}^K$ is a vector of polynomial coefficients. Note that $\mathbf{L}^k = (\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\intercal)^k = \mathbf{V}\boldsymbol{\Lambda}^k\mathbf{V}^\intercal$. Thus, Eq 2.7 becomes as follows,

$$\begin{aligned} \mathbf{y} &= \mathbf{V}g_\theta(\boldsymbol{\Lambda})\mathbf{V}^\intercal\mathbf{x} \\ &= \mathbf{V}(\sum_{k=0}^{K-1}\theta_k\boldsymbol{\Lambda}^k)\mathbf{V}^\intercal\mathbf{x} \\ &= (\sum_{k=0}^{K-1}\theta_k\mathbf{L}^k)\mathbf{x} \end{aligned} \tag{2.9}$$

Therefore, if $(\sum_{k=0}^{K-1}\mathbf{L}^k)_{(v,u)} = 0$, then the transformation matrix $(\mathbf{V}g_\theta(\boldsymbol{\Lambda})\mathbf{V}^\intercal) \in \mathbb{R}^{n\times n}$ will masked the signal from node u to node v. Thus, this localized filter is equal to masking the signal transformation from node u to v if $d_G(v,u) > K$, where d_G denotes the shortest path distance between two nodes on the graph G. Consequently, the filter is localized to K-hop neighborhoods strictly.

Moreover, the $g_{\theta'}(\boldsymbol{\Lambda})$ requires $O(n^2)$ operations for the multiplication with the Fourier basis $\mathbf{V}$,

However, computing either $\mathbf{L}^K$ or the eigendecomposition of $\mathbf{L}$ is computationally expensive, requiring at least $O(n^2)$ operations.

To reduce the computation cost, the $g_\theta(\boldsymbol{\Lambda})$ is parameterized to a polynomial function allowing to be computed recursively from $\boldsymbol{\Lambda}$. The Chebyshev polynomial is a good option that the term of order k is computed recursively as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1, T_1 = x$. Therefore, the filter is parameterized as the truncated expansion as follows,

$$g_\theta(\boldsymbol{\Lambda}) = \sum_{k=0}^{K-1}\theta_k T_k(\tilde{\boldsymbol{\Lambda}}) \tag{2.10}$$

where the parameter $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_K - 1]^\intercal \in \mathbb{R}^K$ is the vector of Chebyshev coefficients and $T_k(\tilde{\boldsymbol{\Lambda}}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k. The $\tilde{\boldsymbol{\Lambda}} = 2\boldsymbol{\Lambda}/\lambda_{\max} - \mathbf{I}_n$, which is a diagonal matrix of scaled eigenvalues in $[-1, 1]$. The λ_{max} denotes the maximal eigenvalues in $\mathbf{\Lambda}$. Accordingly, the transformation matrix can be rewritten as follows,

$$\begin{aligned} g_\theta(L) &= \mathbf{V} g_\theta(\mathbf{\Lambda})\mathbf{V}^\intercal \\ &= \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}}) \end{aligned} \tag{2.11}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}_n$ is the scaled Laplacian corresponding to the scaled eigenvalues.

The above mentioned filter can generalized to transform the input signals with F_{in} channels to the output signal with F_{out} channels as follows,

$$\mathbf{Y}_{\cdot,j} = \sum_{i=1}^{F_{in}} g_{\theta_{ij}}(\mathbf{L})\mathbf{X}_{\cdot,i} \tag{2.12}$$

where input $\mathbf{X} \in \mathbb{R}^{n \times F_{in}}$ and output $\mathbf{Y} \in \mathbb{R}^{n \times F_{out}}$; the $F_{in} \times F_{out}$ vectors of Chebyshev coefficients $\theta'_{ij} \in \mathbb{R}^K$ are the trainable parameters of the localized spectral convolutional networks.

Besides Chebyshev polynomial, approximating spectral convolutions with the Lanczos algorithm [Susnjara et al., 2015] can be an alternative. Based on it, [Liao et al., 2019] proposed another localized spectral graph convolutional networks called *LanczosNet*.

Graph Convolutional Networks

Following [Defferrard et al., 2016], [Kipf and Welling, 2017] proposed Graph Convolutional Networks (GCNs) with the approximated first-order localized spectral convolutions.

Based on the fast localized spectral filters in [Defferrard et al., 2016], for node signal $\mathbf{x} \in \mathbb{R}^n$, GCNs limit the order k as 1 and further approximate with the assumption $\lambda_{max} \approx 2$, as follows,

$$g'_\theta * \mathbf{x} \approx \theta'_0\mathbf{x} + \theta'_1(\mathbf{L} - \mathbf{I}_n)\mathbf{x} = \theta'_0\mathbf{x} - \theta'_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x}, \tag{2.13}$$

with two free parameters θ'_0 and θ'_1.

In order to address the overfitting problem and minimize the number of operations, GCNs introduce a *renormalization* trick to reduced the number of free parameters as follows,

$$\begin{aligned} g_\theta * \mathbf{x} &\approx \theta'_0\mathbf{x} + \theta'_1(\mathbf{L} - \mathbf{I}_n)\mathbf{x} \\ &\underset{\theta=\theta'_0=-\theta'_1}{\approx} \theta(\mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x} \\ &\approx \theta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x} \end{aligned} \tag{2.14}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ and $\tilde{\mathbf{D}}_{(i,i)} = \sum_j \tilde{\mathbf{A}}_{ij}$, which is equal to adding self-loop to the graph. θ is the learnable parameter for each channel of signals.

Therefore, for multiple feature channels, the filters of GCNs transform the input signals $\mathbf{X} \in \mathbb{R}^{F_{in}}$ with F_{in} channels to the output signals $\mathbf{Y} \in \mathbb{R}^{F_{out}}$ with F_{out} channels,

$$\mathbf{Y} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\boldsymbol{\Theta} \tag{2.15}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^{F_{in} \times F_{out}}$ is a matrix of learnable parameters.

Even though each filter will only convolve the first-hop neighborhood of a node, k successive applications of such filters can effectively aggregate the information from the k-hop neighborhood in the neural network model as classical CNNs on grids.

Because each filter of GCNs only considers the first-hop neighbors, the approximated Fourier basis V in [Bruna et al., 2014, Defferrard et al., 2016] is downgraded to coefficients only based on node degrees for neighborhood aggregation. Thus, GCNs enjoy efficient computation and excellent scalability.

Moreover, because the filters in GCNs only consider the first-hop neighborhood, each filter of GCNs can be formulated into a formula of a neural message passing algorithm. Considering updating the representation of node v from $\mathbf{x}_v = \mathbf{X}_{(v,\cdot)}$ to $\mathbf{y}_v = \mathbf{Y}_{(v,\cdot)}$, the filter of GCNs can be reformulated into,

$$\mathbf{y}_v = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \tilde{\mathbf{D}}^{-1/2}_{(v,v)} \cdot \tilde{\mathbf{D}}^{-1/2}_{(u,u)} \cdot \mathbf{x}_u. \tag{2.16}$$

Thus, GCNs are viewed as a bridge between spectral GNNs and spatial GNNs, and inspire several spatial GNNs.

2.2.3 Spatial Graph Neural Networks

Spectral GNNs are well defined and efficient on graphs, however, , they cannot support inductive node representation learning with unseen graphs well because generalizing to unseen nodes requires "aligning" newly observed graph structure. However, the Fourier basis or the approximate localized variants (i.e., V in [Bruna et al., 2014] and the approximated coefficients in [Defferrard et al., 2016, Kipf and Welling, 2017]) in filters of spectral GNNs are fixed and inferred from the observed graph during training, which are fixed but should be changed during evaluation due to newly observed nodes in the evaluation stage. Therefore, Spectral GNNs cannot generalize to unseen nodes in the inductive setting well.

In fact, if the unseen nodes are in a separate connected component, the spectral GNNs might still work well by computing the Fourier basis for the new connected component during inference, if the newly observed connected component, considered as a separate graph, has a similar group-structure pattern with the previously observed graph. However, computing the Fourier basis or those approximated coefficients for newly observed connected components requires extra operations which are computationally costly. This type of inductive node-level representation learning has close relationship with the graph-level tasks such as graph classification.

In the case that the new observed nodes during evaluation are connected to the previously observed graph, considered as with the newly observed subgraph, the spectral GNNs usually fail on generalizing to unseen nodes because the Fourier basis or its approximation might differ largely from the training stage to the evaluation stage.

Therefore, several Graph Neural Networks are designed from the spatial perspective to get rid of the graph Laplacian, for the inductive node representation learning [Hamilton et al., 2017, Velickovic et al., 2018, Monti et al., 2017, Atwood and Towsley, 2016]. Unlike the construction of the spectral convolutions, spatial GNNs introduce feature extractors on the local spatial neighborhood of nodes with learned or fixed coefficients independent of the graph Laplacian.

Owe to the support of inductive setting, spatial GNNs have been applied to many tasks, including geometry learning [Monti et al., 2017], inductive node classification on large-scale graphs with sampling schemes [Hamilton et al., 2017], and graph classification [Xu et al., 2019], etc.

Considering a large amount of related literature and the space limit in the thesis, we will only introduce typical spatial GNNs closely related to our proposed models in the following.

Mixture Model CNNs

Mixture Model CNNs (MoNet) [Monti et al., 2017] is proposed to generalize CNNs on non-Euclidean structured data, not only for graphs but also with a focus on manifolds. Instead of the perspective of graph Laplacian, MoNet introduces a concept of *Pseudo-coordinates* from the perspective of geometry.

To be compatible with deep learning on manifolds, the notations and the perspectives of MoNet are different from previously introduced GNNs. However, these notations and perspectives can be utilized in the case of learning on the graph.

First, let $\mathcal{X}$ be a d-dimensional differentiable manifold, possibly with boundary $\partial\mathcal{X}$. $\mathcal{X}$ corresponds to an undirected weighted graph G from the perspective of graph learning. A smooth real functions (scalar fields) on the manifold is denoted by $f : \mathcal{X} \to \mathbb{R}$, which is corresponding to the node signal in a graph. A point on a manifold or a vertex of a graph is denoted as $x \in \mathcal{X}$ and points $y \in \mathcal{N}(X)$ are considered in the neighborhood of x. For a x with each such y, MoNets associate a d-dimensional vector $\mathbf{u}_{vu}$ as *pseudo-coordinates*. In these coordinates, a weighting function (kernel) is defined as $\mathbf{w}_{\mathbf{\Theta}} = (\omega_1(\mathbf{u}), \ldots, \omega_J(\mathbf{u}))$, parameterized by trainable parameters $\mathbf{\Theta}$. Then, a function named *patch operator* for neighbor information aggregation can be written in the following form,

$$D_j(x)f = \sum_{y \in \mathcal{N}(x)} \omega_j(\mathbf{u}(x,y))f(y), j = 1, \ldots, J, \tag{2.17}$$

where J represents the dimensionality of the extracted patch.

With the patch operator, a spatial generalization of the convolution on non-Euclidean domains can be defined by a template-matching procedure with a filter g as follows,

$$(f \star g)(x) = \sum_{j=1}^{J} g_j D_j(x) f \tag{2.18}$$

where g_j are trainable parameters.

The weighting function is designed as parametric kernels with learnable parameters. The frame work is agnostic to the particular choice of the weighting function. The weighting function utilized in [Monti et al., 2017], is

$$\omega_j(\mathbf{u}) = \exp(-\frac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_j)^\intercal \boldsymbol{\Sigma}_j^{-1}(\mathbf{u} - \boldsymbol{\mu}_j)) \tag{2.19}$$

where $\boldsymbol{\Sigma}_j \in \mathbb{R}^{d \times d}$ and $\boldsymbol{\mu}_j \in \mathbb{R}^{d \times 1}$ are covariance matrix and mean vector of a Gaussian kernel as trainable parameters.

Similar to previously mentioned GNNs, such filters can be generalized to node features with multiple channels.

GraphSAGE

GraphSAGE [Hamilton et al., 2017] is proposed to learn low-dimensional representations of nodes in large graphs in the inductive setting. The learned node representations can be utilized in different down-stream tasks, including node classification and link prediction. GraphSAGE supports not only to learn node representations with down-stream tasks in a supervised manner, but also to pretrain node representations unsupervisedly by predicting graph contexts and finetune a plug-in prediction layer for downstream tasks afterward. The unsupervised learning scheme is introduced in section 2.5.2.

The architecture of each layer in GraphSAGE consists of neighbor aggregation and node representation update. The former is a feature extractor on the closed or opened neighborhood, which includes and excludes the center node, respectively. The latter is to concatenate the representation of the center node and the neighborhood representation from neighbor aggregation and feed the concatenated representation to a fully connected layer with a nonlinear activation function, which can be understood as a concatenation-based skip connection in [Huang et al., 2017].

For node v, a layer of GraphSAGE transform the input representation $\mathbf{x}_v$ of node v to its output node representation $\mathbf{y}_v$ as follows,

$$\begin{aligned} \mathbf{x}_{\mathcal{N}(v)} &= \mathrm{AGGREGATE}\left(\{\mathbf{x}_u, \forall u \in \mathcal{N}(v)\}\right) \\ \mathbf{y}_v &= \sigma\left(\mathbf{W} \cdot [\mathbf{x}_v \| \mathbf{x}_{\mathcal{N}(v)}]\right) \end{aligned} \tag{2.20}$$

where $\mathbf{x}_{\mathcal{N}(v)}$ denotes the representation of the neighborhood of node v; $\mathrm{Aggregate}$ denotes the neighborhood aggregation function.

Three aggregation functions are introduced in [Hamilton et al., 2017]: Mean-aggregator, LSTM-aggregator and Pooling-Aggregator, showed as the following,

$$\begin{aligned} \mathrm{AGGREGATE}^{\mathrm{mean}} &= \frac{1}{|\mathcal{N}(v) \cup \{v\}} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \mathbf{x}_v, \\ \mathrm{AGGREGATE}^{\mathrm{LSTM}} &= \mathrm{LSTM}\left(\pi([\mathbf{x}_u, \ldots, \mathbf{x}_{u'}]^\intercal)\right), \forall u, u' \in \mathcal{N}(v), \\ \mathrm{AGGREGATE}^{\mathrm{pool}} &= \max(\{\sigma(\mathbf{W}_{\mathrm{pool}} \mathbf{x}_u + \mathbf{b}), \forall u \in \mathcal{N}(v)\}), \end{aligned} \tag{2.21}$$

where $\mathrm{LSTM} : \mathbb{R}^{\cdot \times d} \to \mathbb{R}^{d'}$ takes in a sequence of vectors represented as a matrix and output the last hidden states; $\pi : \mathbb{R}^{m \times d} \to \mathbb{R}^{m \times d}$ denotes a random permutation operation of a matrix along the first dimension.

In GraphSAGE, the local neighborhood of a node is viewed as a multiset. Therefore, the aggregation function, or so-called neighbor aggregator, is expected to be symmetric (permutation invariant) in ideal cases.

Among the three aggregators introduced, the Mean-aggregator can be viewed as the inductive version of GCNs, which is permutation invariant and does not depend on the graph Laplacian. However, it is not trainable and essentially based on mean-pooling operations within local neighborhoods. By contrast, the LSTM-aggregator has a better expressive capability because it is trainable by considering each local neighborhood as a sequence of nodes. But it is not inherently permutation invariant and relies on the random permutation of neighbors during the training stage. The Pooling-aggregator is symmetric and trainable, inspired by [Qi et al., 2017]. The max-pooling in pooling-aggregator can be replaced by mean-pooling, and there is no significant difference in performance. However, it is not spatially trainable because all trainable parameters are shared by all nodes in the neighborhood. In other words, there are no specific trainable parameters dependent on different nodes in the neighborhood.

Besides the aggregators, the representation fusion part can be viewed as a generalized skip-connection similar to the one introduced in [Huang et al., 2017], which allows preserving lower-level semantic information learned in previous layers. This leads to significant gains in performance, especially for those aggregators not considering the center node.

Moreover, GraphSAGE also introduces a neighborhood sampling scheme that a fixed-size set of neighbors are randomly and uniformly sampled from the full neighborhood for neighbor aggregation. The fixed size of sampled neighborhoods allow the consumed memory and expected runtime of GraphSAGE in a single batch to be predictable. In addition, the random sampling scheme still allows GraphSAGE to access the information of the full neighborhood of each node with multiple training epochs. The neighborhood sampling scheme is essential for representation learning on large-scale graphs owe to the predictable and controllable memory consumption and runtime in a single batch.

There are also several works discussing the neighborhood sampling scheme. However, since it is less related to our work, we do not discuss works related to it in detail here.

Graph Attention Networks

Recalling previously mentioned spatial GNNs, including GCNs, we can simply categorize them into two genres, spatially fixed (i.e., GCNs [Kipf and Welling, 2017], GraphSAGE-Mean, GraphSAGE-Pooling [Hamilton et al., 2017], GIN [Xu et al., 2019]) and spatially learnable (i.e., MoNet [Monti et al., 2017], GraphSAGE-LSTM [Hamilton et al., 2017]). GNNs in the former genre aggregate neighbors with fixed coefficients, while those in the latter genre aggregate neighbors with coefficients as trainable parameters or parameterized as a neural network. Each filter of spatially fixed GNNs can be decomposed into a pooling layer or a fixed kernel filter shared by all channels on neighborhoods and a fully connected layer shared by all nodes. Such filters are able to capture the natural group structures of graphs by denoising the node representation via smoothing the local neighborhoods. However, using a fixed spatial filter shared by all layers might filter the features with a single template multiple times and lead to extreme results. For example, stacking too many GCN-layers might lead to the oversmoothing problem that all nodes representations converge to a similar value. Considering the theoretical foundation of spectral GNNs and mean-pooling-based spatial GNNs, they frequently suffer from the oversmoothing problem since they are essential blurring kernels.

Therefore, it is meaningful to construct a spatial learnable filter in order to adaptively learn node representations in the graph. However, due to the property of permutation invariance, a common way to construct a spatial learnable filter is to build an adaptive filter [Zamora Esquivel et al., 2019] with content-based weighting functions.

Inspired by the powerful language model, Transformer, self-attention [Vaswani et al., 2017] is introduced into Graph Neural Networks [Velickovic et al., 2018]. Owe to the characteristic of the attention mechanism, the spatial filters of Graph Attention Networks

(GAT) are symmetric as well as trainable with respect to feature channel space and spatial space. The comprehensive introduction of the attention mechanism and Transformer will be in Section 2.3.

From the perspective of the attention mechanism, a GAT layer is essentially a self-attention layer with masking preventing nodes from attending to unconnected nodes. It can be viewed as a variant of the self-attention layer in the decoder of Transformer [Vaswani et al., 2017], in which the masking prevents positions from attending to subsequent positions. From the perspective of GNNs, a GAT layer allows a node to aggregate information from a local neighborhood with different importances. For each node, the aggregation importances, represented as the attention scores, are computed based on the representation of the center node as the query and the representations of nodes in its neighborhood as keys.

The framework of GAT is agnostic to the particular choice of attention mechanism. The particular attention mechanism architecture utilized in [Velickovic et al., 2018], with input node representations $\mathbf{x}_v, \forall v \in \mathcal{V}$, is as follows,

$$\begin{aligned} e_{vu} &= \mathrm{LeakyReLU}\left(\mathbf{a}[\mathbf{W}\mathbf{x}_v \| \mathbf{W}\mathbf{x}_u]\right) \\ \alpha_{vu} &= \begin{cases} \frac{\exp(e_{vu})}{\sum_{u' \in \mathcal{N}(v) \cup \{v\}} \exp(e_{vu'})} & \text{, if } u \in \mathcal{N}(v) \cup \{v\} \\ 0 & \text{, otherwise} \end{cases} \end{aligned} \tag{2.22}$$

where $e_{vu} \in \mathbb{R}$ and $\alpha_{vu} \in \mathbb{R}$ are called prenormalized and normalized attention coefficients; $\mathbf{a}$ and $\mathbf{W}$ are trainable vector and matrix; $\mathrm{LeakyReLU}$ denotes the LeakyReLU nonlinearity function (with negative input slope as 0.2).

Note that in [Velickovic et al., 2018], the closed neighborhood of each node is considered. However, considering the open neighborhood can be an alternative if skip-connection architecture in GraphSAGE [Hamilton et al., 2017] is introduced. Nevertheless, because the attention mechanism outputs a categorical distribution by normalizing across all keys, the attention coefficients are different due to the change of the set of keys (i.e., the key set of each neighborhood includes or excludes the center node).

Similar to Transformer [Vaswani et al., 2017], GATs also employ multi-head attention mechanism with K-independent heads,

$$\mathbf{y}_v = \begin{cases} [\|_{k=1}^{K} \sigma(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}^k \mathbf{x}_u)] & \text{, if on the hidden layers} \\ \sigma(\frac{1}{K} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}^k \mathbf{x}_u) & \text{, if at the output layer (final layer)} \end{cases} \tag{2.23}$$

in which, $\mathbf{a}_k$ and $\mathbf{W}_k$ denotes the trainable parameters in kth attention head in eq 2.22.

Graph Isomorphism Networks

Graph Isomorphism Network (GIN) is a framework for graph-level representation learning [Xu et al., 2019]. It is proved to be one of the GNNs with the most expressive power to capture different graph structures. By considering the neighbors of a node as a multiset, the aggregation of a GNN with the most expressive power needs to be injective with input as multisets. Each layer of GINs transform the input representations $\mathbf{x}_v, \forall v \in \mathcal{V}$ to the output representations $\mathbf{y}_v$,

$$\mathbf{y}_v = \mathrm{MLP}\left((1 + \epsilon) \cdot \mathbf{x}_v + \sum_{u \in \mathcal{N}(v)} \mathbf{x}_u \right) \tag{2.24}$$

where MLP denotes multi-layer perceptrons; ϵ can be a learnable parameter or a fixed scalar value.

GINs is one of the most expressive powerful GNNs for graph isomorphism test because summation is an injective operation while some operations utilized by other GNNs, such as averaging, max-pooling, are not injective. However, GINs do not show significantly better and even show worse performance on node-level tasks (i.e., node classification), probably because capturing the distribution (proportion) of elements in a neighborhood is more important than capturing the exact multiset.

2.2.4 Neural Message Passing

As aforementioned, many GNNs can be formulated into a framework of Neural Message Passing [Gilmer et al., 2017]. These type of GNNs have some typical characteristic in common,

- The locality is a first-hop neighborhood;
- The representation of a neighborhood is a linear transformation of the nodes in the neighborhood.

The general formulation for message passing, or termed as neighbor information aggregation, of neural message passing GNNs-(NMP-GNN) is as follows,

$$\mathbf{y}_v = \sigma\Big(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \cdot \mathbf{W}\mathbf{x}_u\Big) \tag{2.25}$$

where $\mathbf{x}_v \in \mathbb{R}^d, \mathbf{y}_v \in \mathbb{R}^{d'}$ is the input and output representation vectors of node v respectively; $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is the weight matrix; σ denotes arbitrary non-linear activation function; $\alpha_{vu} \in \mathbb{R}$ is the aggregation coefficient from u to v.

Linear Transformation of Neighborhood The aggregation coefficients α_{vu} in NMP-GNN control how a node treats the information from information. They can be interpreted from multiple perspectives, such as the smoothness factor between nodes, or the aggregation importances of the neighbors given the center node. However, overall, similar to CNNs, it is a linear transformation of values of nodes in a neighborhood with coefficients $\{\alpha vu\}$.

The design of the aggregation coefficients can be in multiple ways, as follows,

- as a constant as a hyperparameter; i.e., $\alpha_{vu} = 1$, equivalent to sum-pooling in GIN [Xu et al., 2019];
- inferred from the structural property of center node i; i.e., $\alpha_{vu} = \frac{1}{|\mathcal{N}(v) \cup \{v\}|}$, equivalent to mean-pooling in GraphSAGE-Mean [Hamilton et al., 2017];

- inferred from the structural information of the center node and neighbor nodes; i.e., $\alpha_{vu} = \frac{1}{\sqrt{(|\mathcal{N}(v)|+1)(|\mathcal{N}(u)|+1)}}$; equivalent to GCNs [Kipf and Welling, 2017];
- α_{vu} parameterized to a neural network on representation of nodes; i.e., MPNNs [Gilmer et al., 2017] and GATs [Velickovic et al., 2018].

Generalized Depthwise Separable Convolution From the general equation of NMP-GNN, one can notice that the coefficients for spatial (i.e., for aggregating information from adjacent nodes) and the weights for projecting feature channels to new a channel space, are separable.

This form is, in fact, close to the concept of Depthwise Separable Convolutions for images in Xception [Chollet, 2017]. In Xception, the convolution operation is separated into two parts, *depthwise convolution* which is a spatial convolution performed independently over each channel of input, and *pointwise convolution* which is essentially a 1×1 convolution projecting its input channels onto a new channel space.

Similarly, we can also separate NMP-GNNs into the depthwise operations with aggregation coefficient α_{vu}, and the pointwise operations with feature projecting weights $\mathbf{W}$. Instead of a spatial filter for input channels, in the abovementioned NMP-GNNs, the spatial filter in depthwise operations is shared by all input channels.

Moreover, we can further extend $\{\alpha_{vu} \in \mathbb{R}\}$ shared by all channels to multiple $\{\alpha^k_{vu} \in \mathbb{R}, k = 1, \ldots, K\}$ (i.e., K spatial filters in total) that input channels are split into K groups and the input channels in kth group are assigned with α^k_{vu}, inspired by the group convolution in InceptionNets [Szegedy et al., 2015, Szegedy et al., 2016, Szegedy et al., 2017]. This extension matches the architecture of multi-tower MPNN [Gilmer et al., 2017] and multi-head GAT [Velickovic et al., 2018]. If K equal to the dimension of input channels, it is the same as depthwise separable convolutions except for $\alpha^k_{vu}, \forall i, j, k$ in NMP-GNN are not trainable parameters indexed by related positions. In this case, the max-pooling aggregation in GraphSAGE-Pooling [Hamilton et al., 2017] can be reformulated into NMP-GNNs.

Therefore, we can also view the generalized convolution on graphs in GNNs in the form of the depthwise separable convolutions. In general, depthwise separable convolutions are with similar expressive power with conventional convolutions with fewer parameters to train in a specific order (i.e., a pointwise convolution followed by a depthwise convolution) [Chollet, 2017]. This provides the ground for designing depthwise and pointwise operations separately in GNNs.

In the following paragraphs, we will discuss the spatial filter in the depthwise operations of NMP-GNNs.

Note that one can also insert hidden layers after the input layer, before the output layer, or switching the order of depthwise and pointwise operation in each layer of NMP-GNNs.

Depthwise Pooling or Depthwise Convolution In those GNNs with fixed aggregation coefficients, the spatial filters of depthwise operations are fixed during training and inference. Therefore, we can view such depthwise operations as generalized pooling operations or fixed convolutional kernels spatially.

Even though such depthwise pooling operations can capture the natural clustering information by smoothing, stacking too many such depthwise pooling (i.e., more than 3 layers) might cause severe information loss on spatially due to over sub-sampling or smoothing the neighborhoods. Consequently, instead of smoothing the representations of nodes in the same cluster to be similar, the representations of nodes in the graph become similar overall. This is called the oversmoothing problem [Zhao and Akoglu, 2020] and is considered a common issue of many GNNs.

Adding residual-connections, by addition [He et al., 2016], gating [Pham et al., 2018], concatenation [Huang et al., 2017] mechanism, to NMP-GNNs can preserve node identity and mitigate the oversmoothing [Kipf and Welling, 2017, Hamilton et al., 2017, Li et al., 2015, Velickovic et al., 2018, Xu et al., 2018, Gilmer et al., 2017].

Furthermore, parameterizing $\{\alpha_{vu}\}$ to a neural network can make the filter more expressive than simple pooling operation by introducing learnable parameters.

Such filters can be interpreted as adding a generalized gating mechanism when each node aggregate information from each neighbor [Gilmer et al., 2017] or assigning categorical probabilities across each neighborhood in aggregation [Velickovic et al., 2018]. More sophisticated cases, such as applying any type of feature extractor to the neighborhood as a whole, are feasible, but these cases may not be included in NMP-GNNs (i.e., GraphSAGE-LSTM [Hamilton et al., 2017]).

Factors Controlling the Aggregation In NMP-GNNs, the local neighborhoods provide the natural clustering information from graph structure. However, the definition of aggregation coefficients (a.k.a, the coefficients of the spatial linear transformation) may also introduce other inductive biases.

Assume that a filter updates the representation of node v by aggregating neighbors' information. The mean-pooling filters in GraphSAGE-Mean [Hamilton et al., 2017] assume that all neighbors contribute to the center nodes equally, in which $\{\alpha_{vu}\}, \forall u \in \mathcal{N}(v)$ only depend on the size of $\mathcal{N}(v)$. Instead of normalizing the neighborhood from the perspective of the center node, the filters in GCNs [Kipf and Welling, 2017] consider a symmetric normalization $\alpha_{ij} = \frac{1}{\sqrt{(|\mathcal{N}(v)|+1)(|\mathcal{N}(u)|+1)}}, \forall u \in \mathcal{N}(v)$. Besides, the sum-pooling spatial filters, in GINs [Xu et al., 2019] and GG-NN [Li et al., 2015], attempt to encode the size of each neighborhood as well. We view such filters are controlled by the local structural property of nodes (i.e., node degrees).

By contrast, filters in GATs [Velickovic et al., 2018] and MPNN [Gilmer et al., 2017] tend parameterize $\{\alpha_{vu}\}$ to neural networks with inputs as the representations of nodes. We view such filters are controlled adaptively by the semantic representations of nodes.

For spectral GNNs considering more than first-hop neighborhood [Bruna et al., 2014, Defferrard et al., 2016], even though they are not strictly NMP-GNNs, they also have an operation to aggregate information from nodes in the desired k-hop neighborhoods. The

Model	Local Neighborhood	Aggregation function	Representation Update
GCN	$\mathcal{N}'(v) = \mathcal{N}(v) \cup \{v\}$	$\mathbf{h}_v = \sum_{u \in \mathcal{N}'(v)} \alpha_{vu} \cdot \mathbf{W}\mathbf{x}_u$ $\alpha_{vu} = \tilde{d}_v^{-1/2} \cdot \tilde{d}_u^{-1/2}$	$\mathbf{y}_v = \sigma(\mathbf{h}_v + \mathbf{b})$
GraphSAGE (Mean)	$\mathcal{N}'(v) = \mathcal{N}(v) \cup \{v\}$	$\mathbf{h}_v = \sum_{u \in \mathcal{N}'(v)} \alpha_{vu} \cdot \mathbf{W}\mathbf{x}_u$ $\alpha_{vu} = 1/\lvert\mathcal{N}'(v)\rvert$	$\mathbf{y}_v = \sigma(\mathbf{W}\mathbf{h}_v + \mathbf{W}_{res}\mathbf{x}_v + \mathbf{b})$
GraphSAGE (Pooling)	$\mathcal{N}'(v) = \mathcal{N}(v)$	$\mathbf{h}_v = \max_{u \in \mathcal{N}'(v)} (\sigma(\mathbf{W}_{pool}\mathbf{x}_u + \mathbf{b}_p))$	$\mathbf{y}_v = \sigma(\mathbf{W}\mathbf{h}_v + \mathbf{W}_{res}\mathbf{x}_v + \mathbf{b})$
GAT	$\mathcal{N}'(v) = \mathcal{N}(v) \cup \{v\}$	$\mathbf{h}_v = \sum_{u \in \mathcal{N}'(v)} \alpha_{vu} \cdot \mathbf{W}\mathbf{x}_u$ $\alpha_{vu} = \mathrm{LeakyReLU}(\mathbf{a}_l^\intercal \mathbf{W}\mathbf{x}_v + \mathbf{a}_r^\intercal \mathbf{W}\mathbf{x}_u)$	$\mathbf{y}_v = \sigma(\mathbf{h}_v + \mathbf{b})$
GIN	$\mathcal{N}'(v) = \mathcal{N}(v) \cup \{v\}$	$\mathbf{h}_v = \sum_{u \in \mathcal{N}'(v)} \alpha_{vu} \cdot \mathbf{x}_u$ $\alpha_{vu} = \begin{cases} \epsilon & \text{if } u = v \\ 1 & \text{otherwise} \end{cases}$	$\mathbf{y}_v = \mathrm{MLP}(\mathbf{h}_v)$

where $\mathcal{N}(v)$ denotes the open neighborhood of node v and $\mathcal{N}'(v)$ denotes the neighborhood definition utilized by the model; $\mathbf{W}_\cdot, \mathbf{b}_\cdot, \mathbf{a}_\cdot$ are trainable parameters; ϵ are fixed or trainable parameter; σ denotes non-linear activation function; MLP denotes multilayer-perceptrons; $\tilde{d}_v$ denotes the adjusted node degrees after adding self-loop.

Table 2.2: Summary of NMP-GNNs;

mapping from the spatial domain to the spectral domain can be viewed as constructing graph partitions and aggregating information from each partition.

Summary Table A list of common NMP-GNNs is summarized in the Table 2.2 without considering the multihead or high-dimensional aggregation coefficients cases.

2.3 Attention Mechanism

Attention mechanisms are widely used in many domains such as natural language processing [Bahdanau et al., 2015,Shaw et al., 2018,Wang et al., 2019,Vaswani et al., 2017,Dai et al., 2019], computer vision [Xu et al., 2015,Bello et al., 2019,Cordonnier et al., 2020,Zhao et al., 2020], etc. Inspired by the fact that people might pay more attention to some words or some regions when reading or viewing, attention mechanism aims to allow to aggregate the information from a set of entities with different focuses, given a condition.

2.3.1 Soft Attention and Hard Attention

In general, there are two variants of attention, ″hard″ attention and ″soft″ attention mechanism. ″Hard″ attention mechanisms [Itti et al., 1998, Larochelle and Hinton, 2010, Alexe et al., 2012, Ba et al., 2014, Xu et al., 2015] allow stochastic hard selection on focused entities and are trainable by maximizing an approximate variational lower bound or equivalently by *Reinforce* [Williams, 1992]. On the other hand, ″soft″ attention mechanisms [Bahdanau et al., 2015, Luong et al., 2015, Vaswani et al., 2017] are deterministic and output the probabilities to pay attention to candidates. Consequently, ″soft″ attention mechanisms are trainable by standard back-propagation methods and easily embedded into deep models.

Considering our proposed method is based on ″soft″ attention, we would the attention mechanism is a ″soft″ attention in the following sections, unless otherwise stated.

2.3.2 The Computation of Attention

In order to aggregate information from a set of m target entities with different importances, given a certain condition, attention mechanisms are factorized into two functions, the attention coefficients computation and the aggregation function with attention coefficients.

An entity i in the target set has two representations, the key representation $\mathbf{k}_i$ (shortened as key) and the value representation $\mathbf{v}_i$. (shortened as value). The key representation is utilized in the attention coefficients computation as a condition given by the target i, while the value representation is aggregated as the target information of i in the aggregation function.

Usually, the attention computation function $\omega(\cdot)$, expects two arguments for computing the attention score for an entity i in the target set. One is the key $\mathbf{k}_i \in \mathbb{R}^{d_k}$ of entity i, which contains the information of the target's condition. The other one is a given condition of the aggregation, represented by the query vector $\mathbf{q} \in \mathbb{R}^{d_q}$. Typically, when aggregating information from the target set for an entity, the query is considered a condition representation of it. The attention computation function will output an unnormalized attention score $s_i \in \mathbb{R}$ for each query and key pair. Note that, the attention mechanism requires the convert the unnormalized attention scores to a categorical distribution for each query over its set of keys. In other words, the attention score for each target will be normalized by softmax over the target set.

With the attention scores from attention computation, the aggregation function will aggregate the value representation of each target, with the corresponding attention scores. The set of values corresponding to the set of targets is denoted by $\mathcal{V} = \{\mathbf{v}_i\}, i = 1, \ldots, m$.

Therefore, an attention mechanism can be written as the following,

$$\begin{aligned} \mathbf{s}_i &= \omega(\mathbf{q}, \mathbf{k}_i, \mathcal{K}) \\ \mathbf{c} &= \sum_{i=1}^{m} \mathbf{s}_i \cdot \mathbf{v}_i \end{aligned} \tag{2.26}$$

where $\mathbf{c}$ is the output representation of attention mechanism. $\mathcal{K}$ denotes the set of keys considered.

This architecture of attention modules is called query-key architecture and is widely utilized in many attention-based models- [Bahdanau et al., 2015, Vaswani et al., 2017, Velickovic et al., 2018].

Attention Coefficient Computation Function $\omega(\cdot)$

Typically, $\omega(\cdot)$ will output probabilities of a multinomial distribution for keys given a query. Sometimes, the query can be replaced by a learnable seed vector, embedded into the module as learnable parameters. A number of typical variants are summarized in Table 2.3.

Name	Attention Coefficient Function $\omega(\mathbf{q}, \mathbf{k}_i, \mathcal{K})$	Citation
Content-base	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathrm{cosine}(\mathbf{q}, \mathbf{k}_i)\right)$	[Graves et al., 2014]
Additive/Concatenate	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathbf{v}^\intercal \tanh(\mathbf{W} \cdot [\mathbf{q} \Vert \mathbf{k}_i])\right)$	[Bahdanau et al., 2015, Luong et al., 2015]
Dot-product	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathbf{q}^\intercal \mathbf{k}_i\right)$	[Luong et al., 2015]
General	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathbf{q}^\intercal \mathbf{W} \mathbf{k}_i\right)$	[Luong et al., 2015]
Scaled Dot-product	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\frac{\mathbf{q}^\intercal \mathbf{k}_i}{\sqrt{d_k}}\right)$	[Vaswani et al., 2017]
Location-Base	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathbf{W} \mathbf{k}_i\right)$	[Luong et al., 2015]
Pooling by Attention	$\mathbf{s}_i = \mathrm{Softmax}_{\mathcal{K}}\left(\mathbf{v}^\intercal \mathbf{k}_i\right)$	[Lee et al., 2019b]

where $\mathbf{v}, \mathbf{W}$ are learnable vector and matrix respectively; cosine denotes the cosine similarity $\mathrm{cosine}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\intercal \mathbf{y}}{\Vert\mathbf{x}\Vert_2 \cdot \Vert\mathbf{y}\Vert_2}$, where $\Vert\mathbf{x}\Vert_2 = \sqrt{\sum_{i=1}^{d_x} \mathbf{x}_{(i)}^2}$ is the l^2-norm of $\mathbf{x}$; $\tanh(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$ is the tangent element-wise activation function.

Table 2.3: Summary of Attention Modules with Query-Key Architecture.

Matrix Computation of Attention Coefficients

The attention mechanism described above can be easily generalized to work on several queries if they share the same set of keys. For computational efficiency, the computation of attention coefficients can be rewritten in matrix form and compute attention coefficients for queries in parallel.

Thus, the set of n queries $\mathcal{Q}$ and the set of m keys $\mathcal{K}$ are written in matrix form $\mathbf{Q} \in \mathbb{R}^{n \times d_q}$ and $\mathbf{K} \in \mathbb{R}^{m \times d_k}$ respectively. Notably, even though written in matrix-form, attention mechanisms are working on set-structured inputs and need to satisfy permutation invariance [Vaswani et al., 2017, Lee et al., 2019b].

If the queries do not share the same set of keys completely, masking can be introduced to the key matrix $\mathbf{K}$ of the union of key-sets, to specify a customized key set for each query and prevent implausible connections [Vaswani et al., 2017, Velickovic et al., 2018].

For example, the scaled dot-product attention can be rewritten to,

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{M}) = \text{Masked-Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}, \mathbf{M}\right)\mathbf{V} \tag{2.27}$$

where $\text{Maksed-Softmax}(\mathbf{X}, \mathbf{M})_{(i,j)} := \frac{(\exp(\mathbf{X}) \odot \mathbf{M})_{(i,j)}}{\sum_{j'=1}^{m} (\exp(\mathbf{X}) \odot M))_{(i,j')}}$ and $\mathbf{M} \in \{0, 1\}^{n \times m}$ is a masking matrix that implausible connections are maksed as 0 and otherwise as 1.

Multihead Attention

Instead of performing a single attention computation in one attention layer, Transformer [Vaswani et al., 2017] employs h parallel attention heads, computing attention coefficients, and attentional aggregation within each head separately. Each attention head can be viewed as a separate attention mechanism. However, all attention heads share the same set of queries, keys and values. In order to capture different information with different attention heads, queries, keys, and values will be linearly transformed by three trainable weight matrices respectively, ahead of being fed into an attention head. Each attention head will have the corresponding trainable weight matrices.

The output representations of each head would be concatenated as the layer output. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions, while with a single attention head, averaging inhibits this.

Here is an example of multihead attention,

$$\begin{aligned} \text{MultiHeadAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= [\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^O \\ \text{where head}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \end{aligned} \tag{2.28}$$

where $\mathbf{W}_i^Q \in \mathbb{R}^{d_q \times d'_q}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_k \times d'_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_v \times d'_v}$ and $\mathbf{W}_i^O \in \mathbb{R}^{hd'_v \times d_v}$. Usually, $d'_k = d_k/h$, $d'_q = d_q/h$, $d'_v = d_v/h$ to keep the total computational cost similar to that of single-head attention with full dimensionality. The multi-head attention techniques can be introduced to most variants of attention mechanisms to enhance the capacity.

Similar ideas have been employed in Inception networks [Szegedy et al., 2015,Szegedy et al., 2016,Szegedy et al., 2017] and Xception networks (a.k.a Depthwise separable CNNs).

Self-Attention

Recalling the attention-mechanism with query-key architecture, if the input queries and keys are corresponding to the same set of entities, the attention mechanism is called self-attention or intra-attention.

Self-attention has been utilized successfully to compute a representation of the set of inputs in various tasks such as reading comprehension, abstractive summarization, textual entailment, and learning task-independent sentence representations [Cheng et al., 2016, Parikh et al., 2016, Paulus et al., 2018, Lin et al., 2017].

Besides, self-attention can work as a feature extractor on a set of input entities, by directly attending to them, as introduced in Transformer [Vaswani et al., 2017] and its variants, instead of attending to the hidden states of an encoder. Such self-attention mechanisms have been proven to be efficient and powerful in the NLP domain, and have recently been generalized to other domains as well.

2.3.3 Alignment Model and Feature Extractor

The "Soft" attention mechanism is originally introduced as an alignment model bridging encoder and decoder in tasks such as machine translation and image caption generation [Bahdanau et al., 2015, Xu et al., 2015]. With the attention mechanism, each hidden state of the decoder can aggregate information from all hidden states of the encoder with focuses. This frees the encoder-decoder based model from having to squash all the information from a source to a fixed-length vector. In this case, the input representations of the attention mechanism are assumed to contain structural information encoded by the encoders.

Transformer [Vaswani et al., 2017] introduces self-attention mechanisms that directly attends to the inputs instead of the hidden states of RNNs or CNNs. In this case, the self-attention is utilized as a feature extractor to compute representations of its input in the encoder and decoder of the Transformer. Compared to CNNs and RNNs, self-attention has some advantages. For example, in sequence tasks, compared to CNNs, self-attention can easily generalize to input sequences of unseen length in the training stage and capture long-range dependencies. Compared to RNNs, self-attention allows for significantly more parallelization. The proposed method in [Vaswani et al., 2017], Transformer, is a model architecture eschewing recurrence and relying entirely on attention mechanisms to capture global dependencies between input and output. There are a large number of variants based on the Transformer for tasks in NLPs and other domains [Shaw et al., 2018,Dai et al., 2019,Huang et al., 2018a,Cordonnier et al., 2020,Bello et al., 2019,Lee et al., 2019b,Zhao et al., 2020,Wang et al., 2019].

Notably, the self-attention mechanism cannot inherently capture positional and structural information of input entities [Vaswani et al., 2017, Lee et al., 2019b]. Therefore, to better work on structured data, such as sequences and grids, the positional or structural information is injected to inputs for the self-attention mechanism. One of the most common techniques is adding ″positional encodings″ or ″positional embeddings″ to the input representations before the first self-attention layer. The details about positional encoding are discussed in section 2.3.4.

2.3.4 Positional Encoding/Embedding in Attention

Since the attention mechanism is inherently permutation invariant, it cannot capture the positional information or higher-order structural information among a set of entities. Therefore, for attention mechanisms as feature extractors, we need to inject positional information or structural information manually.

Note that, for attention mechanisms as alignment models, it is unnecessary to do so, if CNNs or RNNs in the encoder and the decoder have already embedded positional/structural information into representations.

We will introduce some methods to enhance the attention mechanisms with the information of structure, with a focus on those attention mechanisms as feature extractors.

One common way to inject positional information is adding positional encodings or learnable positional embeddings for input entities to their representations before the self-attention layer, which is originally for sequential input [Vaswani et al., 2017]. Nearly identical performance are achieved by the variants of Transformer [Vaswani et al., 2017] coupled with positional encodings and learnable positional embeddings. [Shaw et al., 2018] further modifies the learnable positional embeddings from for each absolute position to relative positions. [Wang et al., 2019] generalizes the encodings to tree-structure to model the complicated structural information in trees and rename the positional encoding as structural encoding.

For simplicity, in the following, we will refer all of them to positional embeddings, regardless of whether they are learnable or not, for sequences or other structures.

We will discuss different types of positional embeddings and different ways to inject positional embeddings to attention mechanisms in detail in the following sections.

The Definition of Positional Embeddings

Positional Embeddings are defined as representations that carry the information about the position, order, structure of entities among the input.

The first positional embedding utilized by the attention mechanism is the one introduced in Transformer [Vaswani et al., 2017]. This position embedding aims to provide the order information of each word in a sentence.

The definition of the position embedding is,

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}\right), \\ PE_{(\mathrm{pos},2i+1)} &= \cos\left(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}\right), \end{aligned} \tag{2.29}$$

where *pos* denotes the order of the word in the sentence; *sin* and *cos* are sine and cosine functions respectively.

There are two reasons to utilize the sinusoidal function. First, it allows the model to extrapolate the sequence lengths longer than the ones encountered during training. Second, it allows the model to easily infer the relative positions from absolute positions.

Learned and Fixed Positional Embeddings

The positional embeddings utilized in [Vaswani et al., 2017] are pre-computed and fixed during training. The learned version of positional embeddings [Gehring et al., 2017] are also evaluated in [Vaswani et al., 2017] and produce nearly identical results.

However, with other modifications and on other tasks, learned and fixed positional embeddings may perform differently.

Absolute and Relative Positional Embeddings

In [Vaswani et al., 2017], each word in a sentence is assigned a positional embedding, carrying the absolute positional information within a sentence. Thus, this type of positional embeddings is called absolute positional embeddings.

One of the reasons to utilize sinusoidal function for positional embeddings is that it allows the model to infer relative positional information from absolute positional embeddings. Therefore, on tasks relying on relative differences more than absolute positions, directly learning positional embeddings for pairwise relations (i.e., the distance between two positions) to capture relative positional information might be more efficient [Shaw et al., 2018, Huang et al., 2018b].

[Shaw et al., 2018] proposes an efficient way of incorporating learned relative position embeddings with self-attention, which shows significant improvements in machine translation tasks. With the hypothesis that the precise relative position information is not useful beyond a certain distance, clipping the maximum valid distance is utilized for better generalization to sequence length not seen during training.

For modeling symbolic music, [Huang et al., 2018b] further proposes a memory-efficient implementation of relative-position based attention mechanisms, which is able to work on very long sequences. Furthermore, according to the experimental results, Transformer with relative positional embeddings can better generalize to sequences longer than training examples than the one with absolute positional embeddings.

The fixed relative positional embeddings are introduced in [Dai et al., 2019], in order to enable the model to consider the previous segment when learning entities in the current segment, which benefits the long-range dependencies modeling. Compared to the learned relative positional embeddings, the advantage of fixed positional embeddings is the inductive bias built into the sinusoidal positional encoding, which is beneficial to generalization to longer sentence length. Furthermore, a trainable weight matrix for positional embeddings is introduced in each head of the attention mechanism.

From Sequence to Other Structures

The above-mentioned positional embeddings are mostly for input with sequential structure. However, positional embeddings are generalizable to more complicated structures.

[Wang et al., 2019] proposes structural position representations to model the latent structure of the input sentence, such as the tree-structure of grammar dependency trees.

Moreover, [Bello et al., 2019, Cordonnier et al., 2020, Zhao et al., 2020] extend absolute/relative sinusoid positional encoding in [Vaswani et al., 2017] to two dimensional grid structures for tasks on images. Theoretically, with specific positional embeddings and sufficient attention heads, self-attention can be a feature extractor on images as expressive as any convolutional layer [Cordonnier et al., 2020, Zhao et al., 2020].

With those variants of positional embeddings for attention mechanism, one might raise the following questions:

- With suitable positional embeddings, can (self-)attention mechanism be a general feature extractor for data with different structures?

- How to define positional embeddings for more complicated structures than sequences, trees, and grids?

2.3.5 Other Variants of Attention Mechanism

Besides those above-mentioned attention mechanisms, there are some variants computing the marginal probabilities or the masking scores of keys as the attention coefficients instead of the categorical probabilities.

Structured Attention Network [Kim et al., 2017] extends the standard attention to a conditional random field (CRF). Instead of the probabilities of a multinomial distribution, it computes marginal probabilities for keys as the attention coefficients. Moreover, with inference methods for graphical models, structured attention networks are able to capture structural dependencies between keys without additional methods to inject positional and structural information. Structured attention networks can work on sequences, trees [Kim et al., 2017], as well as on grids [Zhu et al., 2017]. The structured attention networks can be viewed as an alternative of the positional encoding to enhance the standard attention mechanism with positional and structural information. However, in practice, the structured attention mechanism suffers from high computational cost.

Self-attention graph-pooling [Lee et al., 2019a] using GNNs to compute ”attention coefficients” for nodes in a graph. However, the ”attention scores” here can be any scores from non-linearity activation functions such as Tangent and Sigmoid function, which is closer to the masking scores from gating mechanisms in Long short-term memory networks [Hochreiter and Schmidhuber, 1997], Gated recurrent units [Cho et al., 2014] and Gate Linear Unit [Dauphin et al., 2017].

Co-attention is another variant of attention mechanism [Lu et al., 2016, Yu et al., 2017, Yu et al., 2019]. Instead of computing probabilities for a multinomial distribution for each query, the co-attention considers all connections between queries, and keys are under one multinomial distribution.

2.4 Auxiliary Embedding Models

2.4.1 Supervised Learning with Auxiliary Embeddings

In [Weston et al., 2008], an architecture combining a deep model and an embedding model is proposed in order to fully explore unlabeled data during the training stage of transductive semi-supervised learning. In general, in this architecture, the main deep model aims to tackle the supervised task (i.e., predicting the class labels), while the embedding model learns an unsupervised task (i.e., predicting the connections between nodes) to explore unlabeled data as an *auxiliary* task or a regularizer. It is worth mentioning that the embedding model is plugged into the main architecture as an auxiliary model. Thus, the supervised and the unsupervised tasks will be trained simultaneously using the same overall architecture.

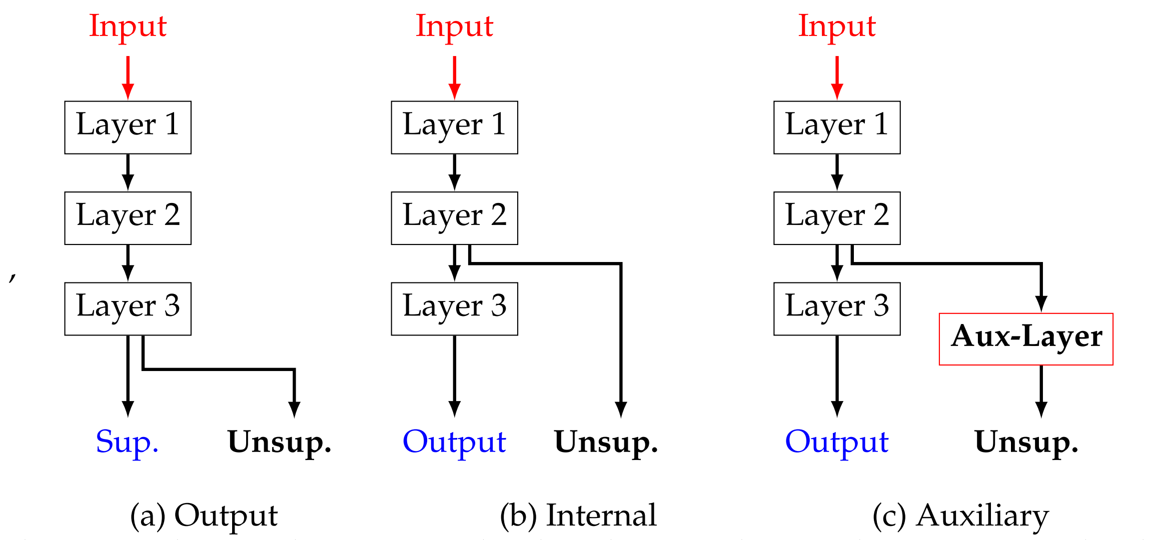
The overall objective function will become,

$$\mathcal{L} = \sum_{i=1}^{L} \mathcal{L}_s(f_s(\mathbf{x}_i, \mathbf{y}_i)) + \lambda \sum_{i,j=1}^{L+U} \mathcal{L}_u(f_u(\mathbf{x}_i, \mathbf{x}_j)), \tag{2.30}$$

where L and U denote the indices of labeled and unlabeled examples respectively; subscript s and u denote supervised and unsupervised parts of the loss function or model, respectively; the supervised and unsupervised models may share part of their architecture.

If the *auxiliary* task can guide the embedding model to learn information beneficial for the supervised task, this framework is supposed to outperform the standard one in the semi-supervised learning under the transductive setting.

Besides plugging the auxiliary models into the main model, there are three ways to connect the unsupervised tasks to the main model to assist the framework to explore unlabeled examples (demonstrated in Figure 2.1)



where Sup. denotes the supervised task and Unsup denotes the unsupervised task; Aux-Layer denotes a layer of the auxiliary embedding model.

Figure 2.1: Three Modes of Embedding Model in Deep Architectures

2.4.2 Semi-Supervised Learning with Graph Embeddings

[Yang et al., 2016] present a semi-supervised learning framework called *Planetoid*, with a similar idea to [Weston et al., 2008]. In Planetoid, the main model aims to predict the class label of entities while an auxiliary embedding focuses on predicting graph context to learn the structural information of entities in the graph.

In the transductive version, the auxiliary embedding model learns node embeddings by the unsupervised tasks of predicting graph context. The auxiliary embedding model is plug into the main model by concatenating the output from hidden layers in the main model and the node embeddings from auxiliary model. The supervised task and unsupervised task are trained with the overall architecture simultaneously.

Besides the transductive version, Planetoid also introduces an inductive formulation, in order to support inductive learning, in which the node embeddings learned is parameterized as a function of input features. The architectures of the transductive and inductive version of Planetoid are demonstrated in Figure 2.2.

As validated by the experimental results in [Yang et al., 2016], learning information of graph structure with the unsupervised task can assist the main model to better explore the structural information in the graph-structured data, especially when the main model cannot explicitly model the structure of the graph.

Note that the framework is agnostic to the choice of the auxiliary embedding models and the unsupervised task. In particular, the auxiliary embedding model utilized in [Yang et al., 2016] is based on the multilayer perceptrons (MLPs) and the unsupervised objective function utilized is following the one in Eq 2.42 in [Tang et al., 2015], which is discussed in section 2.5.2.

These series of works provide a potential approach that utilizes an embedding model to capture graph context (i.e., positional and structural information) as an auxiliary model to provide structural information to the main model.

2.5 Embedding Learning on Graphs/Networks

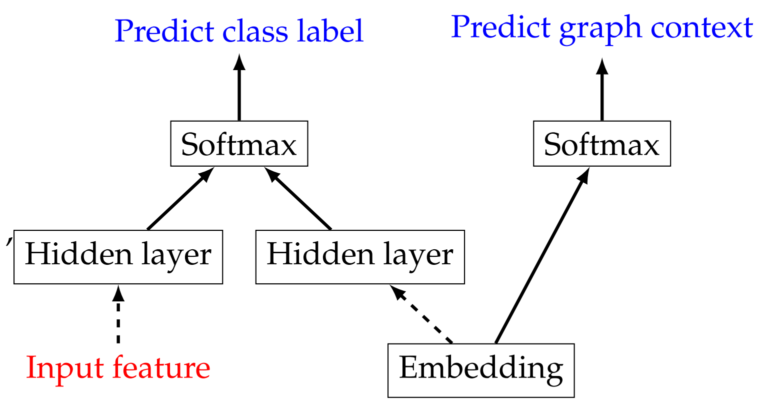
Since there is a large number of corpora related to learning embeddings or vector representations for nodes in graphs or networks, we will only introduce those works highly related to our approach, considering the limited space in the thesis.

Those methods are usually called network embedding methods or (node-level-output) graph embedding methods [Cai et al., 2018], which usually focus on capturing the information from structures and ignore the input feature representation.

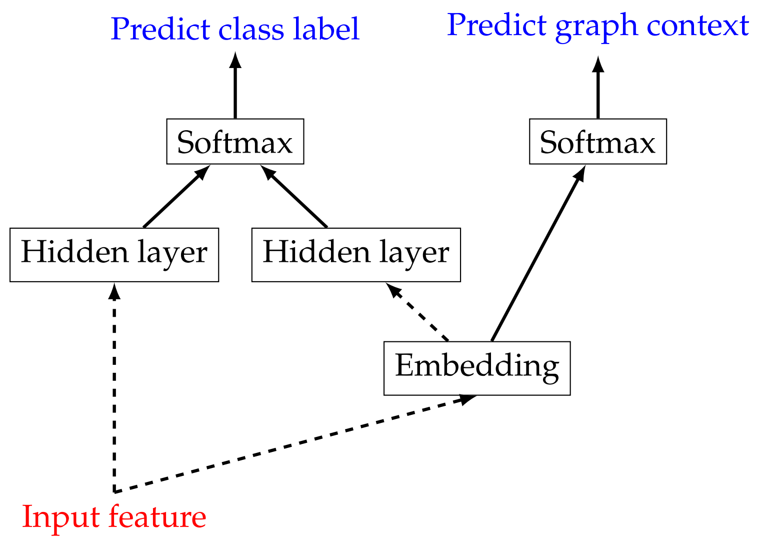
2.5.1 Word Embedding Learning on Text

Before introducing network embedding methods, we will introduce a Word2vec [Mikolov et al., 2013a, Mikolov et al., 2013b], which is a group of related models to produce word embeddings from large corpora of text, because some techniques introduced in it are widely utilized in many embedding methods.

Learning word embeddings from large corpora of text is widely embraced in the natural language processing domain (NLP), with the goal to model the relationships between



(a) Transductive Formulation



(b) Inductive Formulation

Figure 2.2: Architecture of Planetoid: Transductive v.s. Inductive;
where dashed arrow represents a feed-forward network with an arbitrary number of layers and solid arrows denote direct connections.

words within the dataset. Word2vec can capture precise syntactic and semantic word representations and is endowed with a relatively lower computational cost when learning large scale datasets. Some techniques introduced in Word2vec, such as Skip-gram and the Continuous Bag of Words (CBOW) architectures, and the negative sampling strategy are widely utilized in embedding learning methods in many domains.

CBOW and Skip-gram

CBOW and Skip-gram are widely used architectures to learn word embeddings in the NLP domain and, as such, have inspired many embedding models in other domains.

In general, CBOW follows the idea of the bag-of-words in that the words within a certain range of a target word, denoted as the context, are used to predict the target word with a log-linear classifier. Note that the order of words in the context does not influence the projection of the context, and the weight matrix between the input and the projection layer is shared across all word positions.

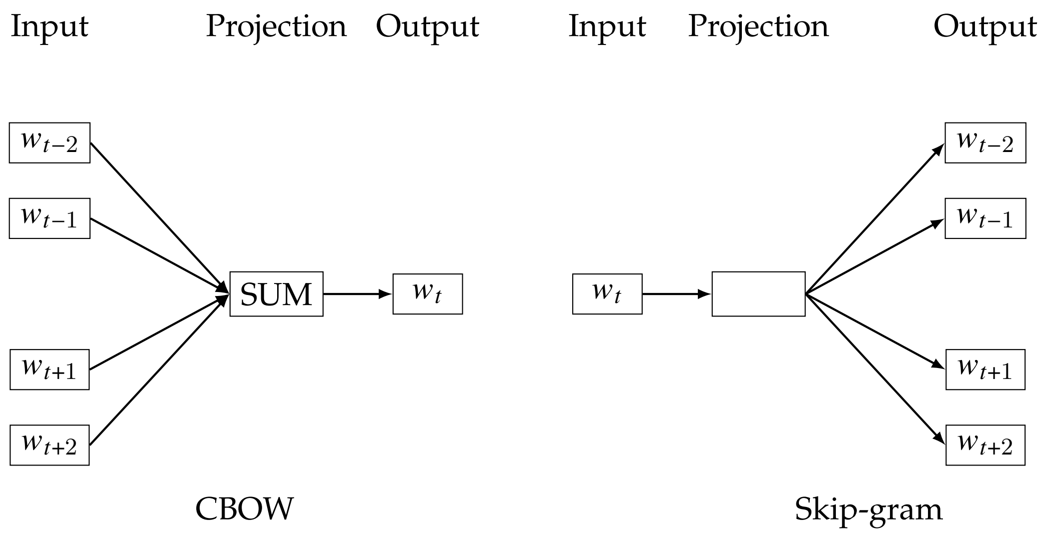
Skip-gram has a similar idea to CBOW. The difference is that, given a word as the input, Skip-gram aims to predict the words surrounding the current word, as context. Increasing the range of context can improve the quality of the learned word embeddings. However, it also increases the computational cost. Therefore, in order to take distant words into consideration, a sampling scheme is introduced. However, considering the more distant words are less likely related to the current word, the sampling scheme gives a lower probability to sample the distant words.

The architectures of CBOW and Skip-gram are shown in Figure 2.3.

Negative Sampling

For the log-linear classifier in Skip-gram architecture, the full softmax is computationally expensive when the number of candidate words is large. Alternatively, there are a few works aiming to approximate the full softmax. Two typical alternatives are hierarchical

,



where w_t denotes the t-th word. The arrow denotes linear transform and there is a log layer on the output.

Figure 2.3: The Architectures of CBOW and Skip-gram.

softmax [Morin and Bengio, 2005] and Noise Constrastive Estimation (NCE) [Gutmann and Hyvärinen, 2012, Mnih and Teh, 2012]. Specifically, NCE converts the multinomial classification problem (softmax) to a binary classification problem (logistic regression) with the belief that a good model should differentiate data from noise via logistic regression. The idea of learning to distinguish data from noise is similar to the hinge loss (a.k.a margin ranking loss), which trains the model to rank the data over noise [Collobert and Weston, 2008]. When it comes to learning vector representations, negative sampling further simplifies NCE, by getting rid of some constraints as well as the asymptotic consistency guarantees of NCE.

The negative sampling will convert the objective function to as follows,

$$\log \sigma(\mathbf{v}'^{\intercal}_{w_o} \mathbf{v}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w}[\log \sigma(-\mathbf{v}'^{\intercal}_{w_i} \mathbf{v}_{w_I})] \tag{2.31}$$

where σ denotes sigmoid function; $\mathbf{v}'_{w_o}$ denotes the vector representation of a target word w_o; $\mathbf{v}_{w_I}$ denotes the vector representation of the input word w_I; $\mathbf{v}'_{w_i}$ denotes the vector representation of a negative sample w_i which is sampled from the noise distribution $P_n(w)$; and k denotes the number of negative samples drawn for each data example.

2.5.2 Node Embedding Learning on Graphs/Networks

Inspired by Word2vec, there are several network embedding techniques learning node embeddings with a similar architecture.

DeepWalk

DeepWalk [Perozzi et al., 2014] is one of the first approaches, currently widely used as a benchmark in model comparisons. The key part of DeepWalk is to use a stream of short random walks to extract sequences from a graph. Afterward, the node embeddings are learned via the Skip-gram model introduced in Word2vec [Mikolov et al., 2013b].

Node2vec

The idea of Node2vec [Grover and Leskovec, 2016] is similar to Deepwalk in that they turn each node in a graph into words in a sentence. The difference is that Node2vec is equipped with a search bias variable α to balance two neighborhood sampling strategies: the breadth-first-search strategy (BFS) and the depth-first-search strategy (DFS), for the random walks. The reason is that, as two extreme sampling schemes, BFS is ideal for learning local neighborhoods, while DFS is better to capture global variables. Thus, using a search bias to control the priorities between them can lead to better sampling from a graph.

More specifically, assuming a random walk just traversed edge $(t, v), t, v \in \mathcal{V}$ and resides at node v, the unnormalized transition probability from node v to node x is

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot \mathbf{A}_{(v,x)}$$
$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \tag{2.32}$$

where $\mathbf{A}$ is the (weighted) adjacency matrix; p and q are two hyper-parameters controlling the priorities of BFS and DFS; d_{tx} denotes the shortest path between node t and node x.

Structural Deep Network Embedding

Besides those methods relying on random walks, some network embedding methods consider first-order and second-order proximities defined on graphs instead of converting graphs to sequences through random walks.

Structural Deep Network Embedding (SDNE) [Wang et al., 2016] utilizes the autoencoder-based framework, considering the following metrics:

- **First-order proximity**: two nodes should be considered similar if they share an edge;
- **Second-order proximity**: two nodes are considered similar if they share many neighboring nodes.

In SDNE, the input vector of node i is defined as the ith row of the adjacency matrix, $\mathbf{x}_i := \mathbf{A}_{(i,\cdot)} \in \mathbb{R}^n$. For each node in the graph, the corresponding input vector will be fed to an autoencoder, which reconstructs an output vector $\hat{\mathbf{x}}_i \in \mathbb{R}^n$ given an input vector $\mathbf{x}_i$ for node i.

Let $\mathbf{h}_i^{(k)}$ denote the representation of node i from the k-th hidden layer of the autoencoder. The $\mathbf{h}_i^{(k)}$ is expected to be a low-dimensional representation of node i encoded by the autoencoder.

For the first-order proximity, the loss function is defined by predicting the connections between each node-pair based on the low-dimensional representations of nodes, as follows:

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} \mathbf{A}_{(i,j)} \|\mathbf{h}_i^{(k)} - \mathbf{h}_j^{(k)}\|_2^2 \tag{2.33}$$

where $\|\cdot\|_2^2$ denotes the square of the l_2 norm.

The second-order proximity is preserved by the reconstruction of the autoencoder, which is analogous to predicting the neighbors of each node given the corresponding input vector via the autoencoder in an unsupervised manner,

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b}_i\|_2^2 \tag{2.34}$$

where $\odot$ denotes the Hadamard product; $\mathbf{b}_i = [b_{i1}, \ldots, b_{in}] \in \mathbb{R}^n$ is a vector to impose more penalty to the reconstruction error of the non-zero elements in $(\hat{\mathbf{x}}_i - \mathbf{x}_i)$, where $b_{ij} = 1$ if $\mathbf{A}_{(i,j)} = 1$, otherwise $b_{ij} = \beta > 1$, in which β is a hyperparameter.

By combining loss functions for both proximities and regularization, the overall objective function is:

$$\mathcal{L} = \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \lambda\mathcal{L}_{reg} \tag{2.35}$$

where $\mathcal{L}_{reg}$ is l_2-norm regularizer applied to weight matrices in the autoencoder; α and λ are hyperparameters controling the priorities of three loss functions.

LINE: Large-scale Information Network Embedding

Similar to SDNE, LINE [Tang et al., 2015] also considers first-order and second-order proximities. The main difference is that LINE view proximities from a probabilistic perspective.

For the first-order proximity, the joint probability (to be connected) between node u and v is defined as follows,

$$p_1(u,v) = \frac{1}{1+\exp(-\mathbf{h}_i^\intercal \cdot \mathbf{h}_j^\intercal)}, \tag{2.36}$$

where $\mathbf{h}_u \in \mathbb{R}^d$ is the low-dimensional vector representation of node u. The objective function for the first-order proximity should be,

$$\mathcal{L}_{1st} = d\left(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot)\right) \tag{2.37}$$

where $\hat{p}_1(u,v) := \frac{\mathbf{A}_{(u,v)}}{Z}$ is the empirical probability for the first-order proximity, in which $Z = \sum_{(u,v)\in\mathcal{E}} \mathbf{A}_{(u,v)}$; $d(\cdot,\cdot)$ is a distance function. By utilizing KL-divergence as the distance function, the objective function becomes,

$$\mathcal{L}_{1st} = \sum_{(u,v)\in\mathcal{E}} \mathbf{A}_{(u,v)} \log p_1(u,v) \tag{2.38}$$

Similarly, for the second-order proximity, the authors define the conditional probability of "context" v generated by node u as follows:

$$p_2(v|u) = \frac{\exp(\mathbf{h}_v^\intercal \cdot \mathbf{h}_u)}{\sum_k^{|\mathcal{V}|} \exp(\mathbf{h}_k^\intercal \cdot \mathbf{h}_u)} \tag{2.39}$$

The corresponding empirical conditional probability is $\hat{p}(v|u) = \frac{\mathbf{A}_{(u,v)}}{d_u}$. Thus, the objective function for second-order proximity is as follows,

$$\mathcal{L}_{2nd} = \sum_{u\in\mathcal{V}} \lambda_u d\left(\hat{p}_2(\cdot|u), p_2(\cdot|u)\right) \tag{2.40}$$

where λ_u is a factor to represent the prestige of vertex u, as an importance factor, which can be the degree or estimated via algorithms such as PageRank [Page et al., 1999]. By replacing the $d(\cdot,\cdot)$ with KL-divergence, the corresponding objective function is defined as,

$$\mathcal{L}_{2nd} = -\sum_{(u,v)\in\mathcal{E}} \mathbf{A}_{(u,v)} \log p_2(v|u).. \tag{2.41}$$

To reduce the computational cost, the final objective function combining first-order and second-order proximities is approximated with negative sampling [Mikolov et al., 2013b], as follows,

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{E}} \left(\log \sigma(\mathbf{h}_u^\intercal \cdot \mathbf{h}_v) + \sum_{i=1}^{T} \mathbb{E}_{v_n \sim P_n(\mathcal{V})} [\log \sigma(-\mathbf{h}_{v_n}^\intercal \cdot \mathbf{h}_u)] \right) \tag{2.42}$$

where $P_n(\mathcal{V})$ is the noisy distribution for edge (u, v) in negative sampling; T is the number of negative edges for each edge. For a target node u, the node v is usually called a positive example, and the nodes v_n from negative sampling are termed as negative examples.

Node Embedding Learning with GNNs

The aforementioned embedding models learn node embeddings in an unsupervised manner with delicately designed objective functions. However, those models do not consider node attributes when learning node embeddings and are not able to generalize the learned node embeddings to the unseen nodes during training, which is known as inductive learning.

If given node attributes, the aforementioned embedding models may be able to support inductive learning by parameterizing the learned positional embeddings as a function of the input node attributes. However, those models usually generalize poorly because the graph structure is not considered explicitly during inference.

Thus, in order to learn node embeddings in an unsupervised manner and better supporting inductive setting, [Hamilton et al., 2017] proposed to learn node representations with GraphSAGE with an unsupervised loss function for predicting graph context (i.e., Eq 2.42). Theoretically, GraphSAGE can provide node embeddings, which generalize to unseen nodes better than those from the non-GNN-based models, because the graph structure associated with unseen nodes can be explored during inference. Specifically, the definition of positive examples for a graph in Eq 2.42 can be extended from the adjacent nodes of the target node *u* to a node that co-occurs near *u* on fixed-length random walks [Hamilton et al., 2017].

Nevertheless, both embedding models rely on parameterizing the learned node embeddings as a function of node features. Thus, if the node attributes contain little information, neither those embedding models nor GraphSAGE can learn high-quality node representations in the inductive setting [Li et al., 2020b]. An extreme example is that when all nodes have the same node attributes, GraphSAGE cannot produce discriminative node embeddings. Besides GraphSAGE, other GNNs can be utilized to learn node embeddings in an unsupervised manner supporting the inductive setting, in cases where the GNN of choice can support inductive setting.

2.6 Graph Neural Networks with Embedding/Encoding Enhancement

Recalling the analysis on the shortcomings of GNNs in section 2.2.4, some works aim to enhance GNNs with embedding models capturing graph contexts.

2.6.1 Geometric Graph Convolutional Networks

[Pei et al., 2020] argues that even though NMP-GNNs have been successfully applied to representation learning on graphs, two fundamental weaknesses of their aggregators limit their ability to represent graph-structured data: the incapacity to explore the structural information of nodes within each neighborhood and the inability to capture long-range dependency. The former weakness is because most NMP-GNNs view each neighborhood as a multiset, ignoring the structural dependencies between nodes within each local area. The latter is because the filters of NMP-GNNs are localized, focusing on filtering local phenomena, which limits their ability to capture long-range dependencies.

Therefore, [Pei et al., 2020] proposes a geometric aggregation scheme for GCNs, termed Geometric Graph Convolutional Networks (Geom-GCN), to tackle the two weaknesses abovementioned.

First, Geom-GCN introduces an embedding module to map the nodes in a graph to a latent continuous space in order to capture the structural information of the graph. The authors introduce three potential embedding methods, Isomap [Tenenbaum et al., 2000], Poincare embedding [Nickel and Kiela, 2017] and struc2vec [Ribeiro et al., 2017]. Note that the embedding methods are trained on the input graph separately, and the end-to-end training is not supported. In particular, the learned node embeddings are 2 dimensional.

With the learned node embeddings $\mathbf{z}_v, \forall v \in \mathcal{V}$, termed latent space node embeddings, Geom-GCNs introduce an extended neighborhood scheme. Besides the conventional definition of the neighborhood on graph , termed neighborhood-in-graph $\mathcal{N}_g(v)$, Geom-GCNs define the neighborhood-in-latent-space as $\mathcal{N}_s(v) := \{u | u \in \mathcal{V}, d(\mathbf{z}_v, \mathbf{z}_u) < \rho\}$ where ρ is a pre-given hyperparameter and the $d(\cdot, \cdot)$ is the distance function dependent on the particular metrics in the latent space. Then, the overall neighborhood is defined as $\mathcal{N}'(v) = (\mathcal{N}(v), \mathcal{N}_s(v))$, termed the structural neighborhood. For a center node v, Geom-GCNs can aggregate not only nodes in the first-hop neighborhood (neighborhood-in-graph) but also the nodes close to node v in the latent space. Thus, Geom-GCNs can aggregate nodes, not in the localized neighborhood but have strong relationships with the center node regarding the global graph structure.

Furthermore, Geom-GCNs further propose a bi-level aggregation scheme. With each aggregation, nodes in neighborhoods are classified to be with different relations to the center nodes, by a relational operator $\tau : (\mathbf{z}_v, \mathbf{z}_u) \rightarrow r \in R$ considering the node embeddings in latent space, where R is the set of the geometric relationships. The nodes with each relation will be assigned with a specific aggregation function similar to rela-

$\tau(\mathbf{z}_v, \mathbf{z}_u)$	$\mathbf{z}_v[0] > \mathbf{z}_u[0]$	$\mathbf{z}_v[0] \le \mathbf{z}_u[0]$
$\mathbf{z}_v[1] \le \mathbf{z}_u[1]$	upper left	upper right
$\mathbf{z}_v[1] > \mathbf{z}_u[1]$	lower left	lower right

Table 2.4: The Relational Operator of Geom-GCN

tional GCN [Schlichtkrull et al., 2018]. There are 8 relations considered in Geom-GCNs, 4 for neighborhood-in-graph and 4 for neighborhood-in-latent-space. In particular, the relational operator utilized in Geom-GCN is defined in Table 2.4, with relationship set R ={upper left, upper right, lower left, lower right}. Accordingly, Geom-GCNs can explore the local structural information within neighborhoods by the relational operator τ as well as capture long-range dependencies with the extended neighborhood scheme.

2.6.2 Distance Encoding for Graph Neural Networks

Similar to Geom-GCN, [Li et al., 2020b] introduces an encoding scheme to enhance the GNNs.

Many well-known GNNs under aggregation-update framework actually mimic the 1-Weisfeiler-Lehman (WL) Algorithm, including GCN [Kipf and Welling, 2017], GraphSAGE [Hamilton et al., 2017], GAT [Velickovic et al., 2018], MPNN [Gilmer et al., 2017], GIN [Xu et al., 2019], etc. Therefore, those GNNs, termed WL-GNNs, also inherit the limitation of the 1-WL test in representing structural information of a set of nodes.

Therefore, Distance Encoding (DE) [Li et al., 2020b] is proposed to enhance GNNs in learning the structural representation of a target node set $\mathcal{S}$. Intuitively, DE is designed to encode certain distance metrics from $\mathcal{S}$ to a node u. DEs have two definitions. One is for nodes with respect to a target set, while the other is defined between two nodes. Starting with the former, DE for a target set $\mathcal{S}$ is defined as an encoding function $\zeta(\cdot|\mathcal{S}) : \mathcal{V} \rightarrow \mathbb{R}^k$, mapping nodes in the graph to k-dimensional vectors. For simiplicty, $\zeta(\cdot|\mathcal{S})$ can be defined as a set aggregation (e.g., sum-pooling) for DEs between nodes u, v, denoted as

$\zeta(u|v)$, where $v \in \mathcal{S}$ as follows:

$$\zeta(u|\mathcal{S}) = \mathrm{AGG}(\{\zeta(u|v)|v \in \mathcal{S}\}) \tag{2.43}$$

With the before-mentioned definition, what information can be captured by DE for a target set is dependent on the choice of DE between nodes.

The authors define two types of DE between nodes, as examples, which capture *shortest-path-distance* (SPD) and *generalized PageRank scores* (GPR) [Li et al., 2019] respectively:

$$\begin{aligned}
&\zeta_{spd}(u|v) = f_3(l_{uv}), l_{uv} = ((\mathbf{W})_{uv}, (\mathbf{W}^2)_{uv}, \ldots, (\mathbf{W}^k)_{uv}, \ldots) \\
&\quad \text{where } \mathbf{W} = \mathbf{A}\mathbf{D}^{-1} \text{ is the random walk matrix} \\
&\qquad\qquad f_3 \text{ can be heristics or parameterized as a neural network} \\
&\zeta_{gpr}(u|v) = \sum_{k\geq 1} \gamma_k (\mathbf{W}^k)_{uv} = (\sum_{k\geq 1} \gamma_k \mathbf{W}^k)_{uv}, \quad \gamma_k \in \mathbb{R}, \forall k \in \mathbb{N}. \\
&\quad \text{where } \{\gamma_k\}_{k\geq 0} \text{ is a weight sequence for PageRank}
\end{aligned} \tag{2.44}$$

After getting DE, the authors also introduce two variants of Distance Encoding Enhanced GNNs. The first one utilizes DE as extra node attributes, termed DE-Enhanced-GNN (DEGNN). The other one, based on the first one, proposed to leverage DE between two nodes to control the aggregation procedure of DEGNN, termed as DE-Aggregation-GNN (DEAGNN).

In DEGNN and DEAGNN, the generalized definition of the adjacency matrix is defined as $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times\cdot}$, in which, $\mathbf{A}_{v,v,\cdot}$ denotes the node attributes of node v while $\mathbf{A}_{v,u,\cdot}$ denotes the attributes of the node-pair (v, u) ($\mathbf{A}_{v,u,\cdot}$ is a zero vector if $(v, u) \notin \mathcal{E}$). DEGNN has the aggregation procedure:

$$\mathrm{AGG}\left(\left\{f_2\left(\mathbf{h}_u^{(l)}, \mathbf{A}_{vu}\right)\right\}_{u\in\mathcal{N}(v)}\right), \tag{2.45}$$

while DEAGNN has the aggregation procedure as:

$$\mathrm{AGG}\left(\left\{\left(f_2\left(\mathbf{h}_u^{(l)}, \mathbf{A}_{vu}\right), \zeta(u \mid v)\right)\right\}_{u\in V}\right), \tag{2.46}$$

where $\mathbf{h}_v^{(0)} := [\mathbf{A}_{vv} \| \zeta(v|\mathcal{S})]$.

Note that DEAGNN can be viewed as a generalization of the Transformer with positional encoding [Vaswani et al., 2017] to the graph domain.

Since DE purely depends on the graph structure, without the dependency on node identifiers and node attributes, DE has generalization ability to unseen nodes, without the constraint from node attributes.

Chapter 3

Methods

3.1 Introduction

Recalling the abovementioned GNNs, they still have limitations due to their inherent characteristics, considering our focused problem statement introduced in Subsection 2.2.1.

The spectral CNNs on graph [Bruna et al., 2014] are based on the operator given by the graph Laplacian. However, the operator is originally proposed for spectral clustering with graphs constructed by similarity, termed similarity graphs, and based on the smoothness introduced by the graph Laplacian [Von Luxburg, 2007, Belkin and Niyogi, 2002, Chung and Graham, 1997]. Thus, it can discover the underlying group invariance in the graph if it can be found via the graph Laplacian. Correspondingly, if the graph does not have a group structure or the group invariance cannot be captured by the graph Laplacian, it might fail to explore the structural information in the graph. There are many cases like this in the real world, in which edges only indicate the existence of some interactions instead of similarity, such as words in a sentence represented as a sequence. The variants that approximate spectral CNNs with Chebyshev Polynomials [Defferrard et al., 2016, Kipf and Welling, 2017] inherit the same limitation since they follow the spectral construction of convolutions. Regardless of the constructed similarity graphs, there exist real-world graph-structured datasets in which an edge indeed indicates the similarity between the two nodes connected. This property is usually referred to as homophily [McPherson et al., 2001] and is rather common. This explains why spectral GNNs have been proven effective on many graph-structured datasets, which are homophilic. Accordingly, we denote the graphs, in which edges only indicate the existence of interactions instead of similarity, as non-homophilic graphs.

Unlike the spectral GNNs, the spatial GNNs define the localized operator on a neighborhood in two ways, either based on a pooling operator and its variants, or defined as an adaptive filter with a content-based weighting function. The spatial GNNs based on the pooling operators, such as mean-pooling, max-pooling [Hamilton et al., 2017] and sum-pooling [Xu et al., 2019], coarsen the learned node representations within each local area. Unlike the pooling layer for images or graph pooling layers [Lee et al., 2019a], such operations do not perform any downsampling, but instead, act similar to a fixed blurring convolutional kernel for images. Therefore, such spatial GNNs can work well on homophilic graphs, by exploiting the underlying group invariances by blurring each local area.

By contrast, the spatial GNNs with adaptive filters are more expressive and, in a way, go beyond blurring local areas because they aggregate information from neighbors with different trainable coefficients. The coefficients of each adaptive filter are usually produced by a pairwise content-based weighting function in current works, such as mixture models [Monti et al., 2017] and attention mechanism [Velickovic et al., 2018]. However, such content-based weighting functions applied to the local neighborhood, can at most explore the co-occurrence of nodes in each local area, while the edges exist between nodes. In other words, the local structural information of each local area is not fully explored because the content-based weighting functions cannot inherently explore the structure in each neighborhood. Consequently, without the guide from structural information, content-based adaptive filters might result in smooting the localities [Chen et al., 2020] and still fail in non-homophilic cases. Despite that, the spatial GNNs with adaptive filters are supposed to have better expressive power and potential to work on non-homophilic graphs, if endowed with some enhancements.

Therefore, we aim to propose a framework to enhance the content-based adaptive filters of spatial GNNs with extra structural information to better learn on graphs beyond the homophily. Specifically, we propose a framework to enhance GATs by incorporating node embeddings from an auxiliary embedding model, which learns the structural and positional information of nodes on graphs. The framework is termed *Graph Attention Networks with Auxiliary Positional Embedding Models* (GAT-POS).

In the Transformer [Vaswani et al., 2017] and its variants [Shaw et al., 2018, Dai et al., 2019], in order to enable the attention mechanism to exploit the sequence structure of the words in a sentence, the positional encoding is incorporated in the attention mechanism. In the works from [Cordonnier et al., 2020, Zhao et al., 2020], the authors also extend the aforementioned positional encoding to two dimensions when generalizing attention mechanism as localized operators for images.

Inspired by them, we introduce a kind of embeddings, which provide node-level positional and structural information in the graph, and modify the graph attentional layer in GATs to incorporate them. However, considering general graphs' properties, there is no straightforward way to generalize the positional encoding for sequences and grids to general graphs. Thus, inspired by the learned positional representations in [Shaw et al., 2018], we propose to learn node embeddings that to capture structural and positional information of the nodes in the graph, termed positional embeddings on graphs. For simplicity, we term the information about the graph context, including structural, positional and locational information, as structural information of graphs. With the positional embeddings containing the information of spatial structure, each filter in GATs is expected to adaptively extract features from local areas considering the semantic information and structural information. Specifically, the positional embedding model is trained to learn graph context with an unsupervised loss task, while the main model, a modified GAT, is trained with a supervised task to predict node labels. Furthermore, inspired by the works from [Weston et al., 2008, Yang et al., 2016], we plug the embedding model into our modified GAT architecture as an auxiliary model, and train the supervised and the unsupervised tasks simultaneously. Therefore, our framework enjoys the benefits of end-to-end training. Furthermore, the learned positional embeddings are not only predictive of the graph context but also potentially beneficial to predict node categories.

It is worth mentioning that the information captured by positional embeddings is dependent on the choice of the embedding model as well as the objective of the unsupervised task. Our framework is agnostic to a specific type of node embeddings, if they can provide beneficial information in relation to the structure of the graph.

3.2 Motivation

3.2.1 Semantic Information and Structural Information

Revisiting the representation learning on graphs, there are two sources of information in graph-structured data, the information from node attributes and the information from graph structures. However, for different tasks, there may require different emphases.

In graph isomorphism tasks, the specific semantic information for each node seems less important, while the structural rule of each node deserves more attention. Accordingly, models in such tasks tend to preserve as much structural information as possible while the learning of the semantic information of each node can be compromised [Xu et al., 2019].

In contrast, for node classification tasks, the semantic representations of nodes are more important than the explicit structure of the graph. Thus, in such tasks, models intend to learn the semantic representations of nodes with the graph structure [Kipf and Welling, 2017, Velickovic et al., 2018, Hamilton et al., 2017]. In these cases, the meaningful semantic information should be preserved as much as possible while the information of structure is supposed to be an auxiliary.

Therefore, a good model for graph isomorphism tasks might not perform well on node classification tasks and vice versa.

In general, for representation learning on graphs, an ideal model should have the ability to fully explore both the semantic and structural information and learn node representations incorporating both of them adaptively.

3.2.2 Positional Embeddings for Graph Attention Networks

Filters beyond Blurring Convolutional Kernels

Recalling the discussion about current GNNs in Section 3.1, most GNNs are designed to work on homophilic graphs. According to the previous analysis, the filters of spectral GNNs, as well as pooling-based spatial GNNs, are reminiscent to blurring convolutional kernels.

A model stacking several (i.e., 2 or 3) GNN layers with such filters can work well on homophilic by discovering the underlying group structures. However, with an increasing number of such layers, the learned node representations would converge to similar values. This issue is usually referred to as the oversmoothing problem [Li et al., 2018, Zhao and Akoglu, 2020].

Theoretically, spatial GNNs with content-based adaptive filters have the potential to go beyond blurring kernels. However, in practice, many such GNNs also suffer from the oversmoothing problem, which implies that the learned content-based adaptive filters still end up as blurring convolutional kernels [Chen et al., 2020]. This is reasonable because the weighting functions of such filters cannot fully explore the structural information within each local area. Without incorporating any structural information, the learned adaptive filters generated by a content-based weighting function likely end up as a blurring kernel.

Therefore, in order to construct a GNN model applicable to non-homophilic graphs, it is necessary to define a filter which is able to exploit structural information as well as semantic information of the nodes.

GATs Lack the Positional Embeddings on Graph

A sentence with several words with sequence structure is a natural and special case of non-homophilic graphs since the edges between words indicate the interactions rather than the similarity. The Transformer [Vaswani et al., 2017] has been proven effective and powerful in language modeling, learning word representations balancing semantic and positional information from numerous sentences. This implies that a content-based attention mechanism incorporating positional information has the potential to learn complicated semantic representations with structure.

However, the GATs, as a representative of spatial GNNs with content-based adaptive filters, do not incorporate any type of positional and structural information. In the following, we will analyze the difference between graph attention mechanism and standard attention mechanism and discuss how to enhance the expressive power of GATs to learn non-homophilic graphs.

On the one hand, with GNNs architecture, GATs are able to explore the natural grouping information of the graph by modeling the co-occurrences of nodes in the neighborhood. Each filter of GATs computes the aggregation coefficients as the attention scores for a neighborhood with center nodes as query and all nodes in the closed neighborhood as keys and values. However, similar to the standard attention mechanism, each filter of GATs cannot inherently capture the structural information within the neighborhood. In other words, each local area of a graph (i.e., neighborhood) is considered as a multiset, which is actually a depth-1 tree (a.k.a a star graph).

On the other hand, the arguments of the attention mechanism at the first layer of standard GATs are the node attributes or node representation without information about graph structure and node position. Thus, the aggregation coefficients disregard any potentially useful structure information. Consequently, neither the first layer of GATs nor the successive layers would output node representations with any useful structural information within each local area in the graph.

In order to enable the attention mechanism to consider the structure in the target keys, a widely used solution in the NLP domain is to add positional encodings or embeddings [Vaswani et al., 2017] to entities as discussed in section 2.3.4.

Therefore, we aim to provide a type of node embedding, which captures the information contained in the graph structure and helps GATs overcome the abovementioned shortcomings. Since we expect the node embeddings can provide the information about the structural closeness at graph level between two nodes, following the terms positional encoding [Vaswani et al., 2017] and positional representations [Shaw et al., 2018], we term them as positional embeddings for graphs.

Positional Embeddings on Graph

It is worth noting that, unlike sequences, trees, or grids [Vaswani et al., 2017, Wang et al., 2019, Dai et al., 2019, Bello et al., 2019], defining positional information on general graphs is challenging.

Thus, we propose to learn node-level graph embeddings (a.k.a node representations of graphs) that are predictive of graph context, termed positional embeddings. The learned positional embeddings are fed to GATs to enable the attention mechanism in filters to learn meaningful node representation considered the structural information of the graph.

Inspired by learned positional representations [Shaw et al., 2018], we further set the positional embeddings trainable during the supervised training of GATs. Since GATs can incorporate positional embeddings in multiple ways, we choose to follow the framework of [Weston et al., 2008] and learn the positional embeddings through an auxiliary model. The positional embeddings extracted from this auxiliary model are then plugged into the main GAT model. Note that the semantic node representations from the main GAT and the positional embeddings from the auxiliary model are trained jointly to learn positional embeddings that are not only predictive of the graph context, but also useful for the supervised task of the main model.

3.3 Model Architecture

Our proposed model consists of two parts, the auxiliary embedding model, which learns the positional node embeddings, and the main model, which is a modified GAT incorporating the positional embeddings from the auxiliary embedding model.

3.3.1 Auxiliary Embedding Model

As discussed in the previous section, we proposed to build an auxiliary embedding model to learn positional representation predictive of graph context. The goal is to enable the filters of GATs to capture the structural information from the graph structure, which cannot be inherently leveraged with the original formulation of GAT. By incorporating positional embeddings predictive of graph context, the filters of GATs can not only explore the structural information within each local area but also be aware of other information of graphs beyond the locality.

The particular auxiliary embedding model setup utilized by us closely follows the node-level graph embedding methods based on proximity discussed in section 2.5.2 (i.e., LINE [Tang et al., 2015], Planetoid [Yang et al., 2016] and GraphSAGE [Hamilton et al., 2017]). The positional embeddings learned by such embedding models can capture the proximity between node pairs from the spatial structure of the graph, which is essential structural information that cannot be inherently leveraged with the original formulation of GATs. Particularly, we base our proposed auxiliary embedding models on two widely used node-level graph embedding approaches: Planetoid [Yang et al., 2016] and GraphSAGE [Hamilton et al., 2017].

Note that our framework is agnostic to the particular choice of auxiliary embedding model. One alternative option, as an example, is the node representations capturing the structural rule of each node [Srinivasan and Ribeiro, 2020].

Furthermore, instead of training two the main model and the embedding models separately, we plug the embedding model into GATs as an auxiliary architecture based on the work from [Weston et al., 2008]. The supervised tasks for the main model (i.e., learning node labels) and the unsupervised tasks for the auxiliary model (i.e., predicting graph context) are trained using the same architecture simultaneously. With such architecture, the auxiliary embedding model can learn positional embeddings not only predictive of graph context but also beneficial for the supervised tasks. Moreover, end-to-end training is also supported because of the simultaneous training on the overall framework.

Planetoid

The auxiliary embedding model of Planetoid is a multiple layer perceptron. We considered both the inductive and transductive version of Planetoid. The transductive version, referred to as Planetoid-transductive, learns the representation embedding for each node in the observed graph by learning to predict the graph context. This version relies on the index of nodes observed, which means it cannot be applied to the inductive setting. To be compatible with the inductive setting, the inductive version of Planetoid, referred to as Planetoid-inductive, parameterizes the node embeddings as a function of node attributes. Therefore, the learned representations can be generalized to new observed nodes and graphs, through the shared node attributes. However, there are two shortcomings of Planetoid-inductive. First, this method highly relies on how discriminative the node attributes are and whether the graph structure commutes with node attributes. For example, Planetoid-inductive cannot output two different positional embeddings for two nodes with different proximities to the third node if they have the same node attributes.

In the thesis, we will utilize Planetoid-inductive as one possible auxiliary model considering its desired inductive property. However, in the transductive learning setting, the Planetoid-transductive may outperform without the dependence on node attributes.

GraphSAGE-Mean

The other auxiliary embedding model we utilize is a GNN-based embedding model, GraphSAGE, which, unlike Planetoid, considers the graph-structure during inference. Consequently, GraphSAGE is supposed to have better generalization in the inductive setting because the learned node representation of a node v is parameterized to a function of the attributes and the structure associated with the nodes in the k-hop neighborhood of the node v.

In this thesis, we focus on one of the versions of GraphSAGE, which utilizes a mean-aggregator, referred to as GraphSAGE-Mean. Other versions of GraphSAGE are also applicable, but we do not include them in the comparison due to time and computational constraints. In the following sections, the term GraphSAGE may refer to the GraphSAGE-Mean without further clarification.

3.3.2 Graph Attentional Layer with Positional Embeddings

Each standard graph attentional layer in GATs only incorporates the node representations from the last layers (i.e., content-based attention mechanism), which focuses on capturing the semantic information by smoothing the neighborhoods. Thus, there is no extra positional or structural information other than the natural clustering information of the neighborhood.

Inspired by the positional encoding in Transformer [Vaswani et al., 2017] and its variants, we modify the graph attentional layer to be compatible with extra positional embedding.

Attention Coefficients Computation

In the attention scores computation, we need to incorporate the positional embeddings as well as the node representations. To do so, we choose to concatenate the node representations and positional embeddings before being fed to the attention computation. Utilizing concatenation instead of summation used in [Vaswani et al., 2017] can lead to more flexibility to fuse representations of different sizes [Huang et al., 2017].

Thus,

$$\mathbf{x}'_v = [\mathbf{x}_v || \mathbf{p}_v] \tag{3.1}$$

where $\mathbf{p}_v$ is the positional representation of node v produced by the auxiliary embedding model.

For instance, the computation in eq 2.22 will become,

$$\begin{aligned} e_{vu} &= \mathrm{LeakyReLU}\left(\mathbf{a}^\intercal [\mathbf{W}' \cdot \mathbf{x}'_v || \mathbf{W}' \cdot \mathbf{x}'_u]\right) \\ &= \mathrm{LeakyReLU}\left(\mathbf{a}^\intercal [\mathbf{W} \cdot \mathbf{x}_v + \mathbf{W}_p \mathbf{p}_v || \mathbf{W} \cdot \mathbf{x}_u + \mathbf{W}_p \mathbf{p}_u]\right) \end{aligned} \tag{3.2}$$

where $\mathbf{W}'$ is the trainable matrix for the representation fusing the node representations and the positional embeddings, $\mathbf{W}_p$ is a trainable matrix for the positional embeddings.

Aggregation with Positional Representations

When considering the attentional aggregation in GATs, we have two options.

One is to follow the Transformer and inject the positional embeddings to the node representations before the first graph attentional layer. Note that in this case, no extra positional embeddings are considered in the following layers. For example, considering only one attention head, the first layer of graph attentional layer is computed as follows,

$$\begin{aligned}
e_{vu} &= \mathrm{LeakyReLU}\left(\mathbf{a}^{\intercal}\left[\mathbf{W}\mathbf{x}_v + \mathbf{W}_p\mathbf{p}_v \| \mathbf{W}\mathbf{x}_u + \mathbf{W}_p\mathbf{p}_u\right]\right) \\
\alpha_{vu} &= \begin{cases} \frac{\exp(e_{vu})}{\sum_{u\in\mathcal{N}(v)\cup\{v\}}\exp(e_{vu})} & \text{, if } u \in \mathcal{N}(v) \\ 0 & \text{, otherwise} \end{cases} \\
\mathbf{y}_v &= \sigma\Big(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \alpha_{vu}\cdot(\mathbf{W}\mathbf{x}_u + \mathbf{W}_p\mathbf{p}_u)\Big)
\end{aligned} \tag{3.3}$$

The successive layers follow the same computation steps as the standard graph attentional layer (eq 2.22).

The other option is to consider positional embeddings in the attention coefficients computation, but not take them into account in the neighbor aggregation step. For instance, also considering a single attention head, one graph attentional layer is defined as follows,

$$\begin{aligned}
e_{vu} &= \mathrm{LeakyReLU}\left(\mathbf{a}^{\intercal}\left[\mathbf{W}\mathbf{x}_c + \mathbf{W}_p\mathbf{p}_c \| \mathbf{W}\mathbf{x}_u + \mathbf{W}_p\mathbf{p}_u\right]\right) \\
\alpha_{vu} &= \begin{cases} \frac{\exp(e_{vu})}{\sum_{u\in\mathcal{N}(v)\cup\{v\}}\exp(e_{vu})} & \text{, if } u \in \mathcal{N}(v) \\ 0 & \text{, otherwise} \end{cases} \\
\mathbf{y}_v &= \sigma\big(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \alpha_{vu}\cdot\mathbf{W}\mathbf{x}_u\big)
\end{aligned} \tag{3.4}$$

This type of modification is inspired by the works of [Zhao et al., 2020, Cordonnier et al., 2020] which extend attention mechanism to images. In this case, such a layer filters merely on the node representations, mainly capturing the semantic information, with coefficients based on the node representations as well as positional embeddings.

Note that $\alpha_{vu} \in \mathbb{R}$ is a scalar shared by all feature channels and the extension to multi-head attention can be employed.

3.3.3 Graph Attention Network with Auxiliary Positional Embedding

Considering the different options of auxiliary positional embedding model and graph attentional layer with positional embedding, we explore four versions of our proposed framework, which we call Graph Attention Networks with Auxiliary Positional Embeddings (GAT-POS).

The main differences between versions lie on the following points,

1. the architecture of the auxiliary embedding model;
2. the architecture of graph attentional layer which incorporates positional embedding;
3. the interaction between the main model and the auxiliary model in the forward inference and gradient backward steps.

Notation

In the following sections, for simplicity, let $\mathbf{h}_v^l$ denote the representation of node v at lth layer of the main model consisting of modified graph attentional layers. $\mathbf{h}_v^0$ is initialized as the node attributes $\mathbf{x}_v$, and the final prediction is defined as $\mathbf{y}_v := \mathbf{h}_v^L$. The main model can consist of graph attentional layers, fully connected layer, as well as normalization layers (i.e., batch normalization, layer normalization and l_2 normalization layer). Since we aim to compare how positional embeddings can enhance GATs, we retain the original architecture of the standard GATs in our main-model, which is stacking several graph attentional layers.

We also let $\mathbf{p}_v^t$ denote the representations of node v at the tth layer of the auxiliary positional embeddings model (shortened as aux-model). In the Planetoid-transductive aux-model, $\mathbf{p}_v^0$ is initialized as the vth column of the identity matrix $\mathbf{I}_n \in \mathbb{R}^{n \times n}$, where n is the number of nodes in the input graph. This is equal to learning a representation based on a learned d-dimensional embedding for each node, where d is the dimension of the first hidden layer. Note that Planetoid-transductive is only applicable to the transductive setting because the number of nodes, n, needs to be fixed during training and evaluating. In the aux-models based on Planetoid-inductive and the GraphSAGE, $\mathbf{p}_v^0$ is initialized as

$\mathbf{x}_v$, which is equal to parameterizing the based node embedding in transductive version to a function of node attributes.

Thus we can reformulate the aux-model as,

$$\begin{aligned} &\text{Planetoid:} \\ &\mathbf{p}_v^t = \sigma(\mathbf{W}_{emb}^t \mathbf{p}_v^{t-1}) \\ &\text{GraphSAGE:} \\ &\mathbf{p}_v^t = \sigma(\mathbf{W}_{emb}^t [\mathbf{p}_v^{t-1} \| \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{p}_u^{t-1}]) \end{aligned} \tag{3.5}$$

where σ is a non-linearity (i.e., ReLU) if $t < T$, and usually set as identity function in the final layer.

We assume that the last layer of the model (i.e., the output layer) is L, the last layer of the aux-model is T, and the number of node classes is C.

GAT-POS-Transformer

This version follows the architecture of Transformer [Vaswani et al., 2017], where for each node, the positional embedding is injected into the node representation before being fed to the first graph attentional layer. Accordingly, this version is termed GAT-POS-Transformer.

Main Model with Graph Attentional Layers We consider the main model as L stacks of graph attentional layers. Then, the lth graph attentional layer is defined as follows,

$$\begin{aligned} \tilde{\mathbf{h}}_v^{l-1} &= \begin{cases} [\mathbf{h}_v^{l-1} \| \mathbf{p}_v^T] & \textbf{, if } l = 1 \\ \mathbf{h}_v^{l-1} & \text{, otherwise} \end{cases} \\ \alpha_{vu}^k &= \underset{u \in \mathcal{N}_v \cup \{v\}}{\text{Softmax}}(\text{leakyrelu}(\mathbf{a}_k^{l\,\top} [\mathbf{W}_k^l \tilde{\mathbf{h}}_v \| \mathbf{W}_k^l \tilde{\mathbf{h}}_u]) \\ \mathbf{h}_v^l &= \begin{cases} \|_{k=1}^K \sigma(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \tilde{\mathbf{h}}_u^l) & \text{, if } l < L, \\ \underset{C}{\text{Softmax}}\left(\frac{1}{K} \sum_{k=1}^K \sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \tilde{\mathbf{h}}_u^l)\right) & \text{, if } l = L. \end{cases} \end{aligned} \tag{3.6}$$

where $\mathbf{W}_k^l$ and $\mathbf{a}_k^l$ are the weight matrix and vector respectively of the kth attention head at the lth layer.

Auxiliary Embedding Models We consider a T-layer auxiliary positional embedding model based on Planetoid-transductive, Planetoid-inductive and GraphSAGE. Note that the positional embeddings of node v from the lth layer of auxiliary model is denoted as $\mathbf{p}_v^l$.

Supervised Loss The supervised loss is applied to the predicted labels of nodes from the output of the main model and the corresponding labels to guide the model to learn the category of nodes. The specific loss function utilized is the cross-entropy error over all labeled examples,

$$\mathcal{L}_{\mathcal{Y}}(\{\hat{\mathbf{y}}_v\}_{v \in \mathcal{Y}_L}, \{\mathbf{y}_v\}_{v \in \mathcal{Y}_L}) = -\sum_{v \in \mathcal{Y}_L} \mathbf{y}_v^\top \log \hat{\mathbf{y}}_v \tag{3.7}$$

where $\mathcal{Y}_L$ is the set of nodes with labels (i.e., the nodes in training, validation in the corresponding stages); $\mathbf{y}_v^\top$ denotes the label of example v while $\hat{\mathbf{y}}_v$ denotes the predicted label of example v from the output layer of the main model.

Without further description, all proposed versions will have the same supervised loss.

Unsupervised Loss The unsupervised loss is meant to be applied to the learned positional embeddings of all observed nodes during training from the auxiliary model and guide the learned embeddings to contain positional information of the nodes in the graph. The particular setup we utilized is based on the one utilized in LINE [Tang et al., 2015], Planetoid [Yang et al., 2016], GraphSAGE [Hamilton et al., 2017] without random walks augmentation. This objective function has been utilized to guide the node embeddings to capture structural information of the graph in many graph/network embedding models, as discussed in Section 2.5.2.

$$\mathcal{L}_{\mathcal{G}}(\{\mathbf{p}_v^T\}_{v\in\mathcal{V}}, G)) = \sum_{v\in\mathcal{V}} \sum_{u\in\mathcal{N}(v)} \Big(-\log\sigma(\mathbf{p}_v^{T\top}\mathbf{p}_u^T) - Q\cdot\mathbb{E}_{u'\sim P_n(v)}\log(\sigma(-\mathbf{p}_v^{T\top}\mathbf{p}_{u'}^T))\Big) \tag{3.8}$$

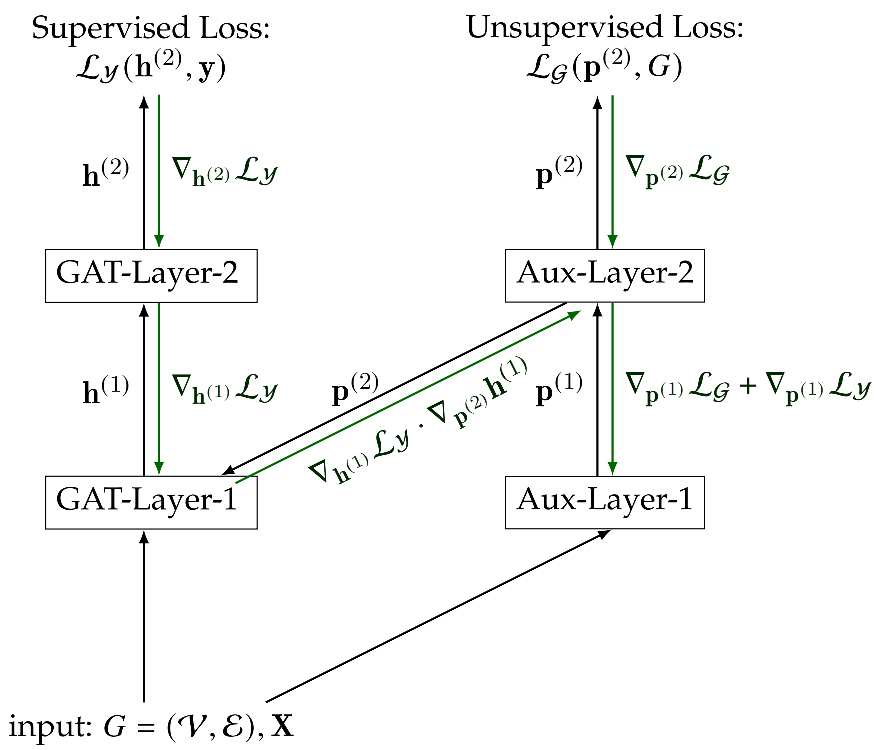
where $P_n(v)$ denotes the distribution negative sampling for node v and Q is the number of negative samples per edge (i.e., for a neighbor of node v, sample Q non-adjacent nodes of node v as negative samples).

The unsupervised loss in other versions proposed follows the same goal and definition without extra specification.

A demonstration of GAT-POS-Transformer with two-layer main model and two-layer auxiliary model ($L = T = 2$) is shown in Figure 3.1.

GAT-POS-CNN

Unlike GAT-POS-Transformer which is very similar to the Transformer, we attempt to build another version closer to CNNs on grids, inspired by [Cordonnier et al., 2020, Zhao et al., 2020]. In this version, the positional embeddings will be considered in attention coefficients computation, but will not be aggregated in the neighborhood aggregation step. The goal is to consider the attention mechanism enhanced by positional embeddings as a generalized localized convolution filter where the learned coefficients for linear smoothing depend on not only the related position information, like conventional CNNs, but also the current node representation. However, the positional embeddings will not be injected into the node representation and aggregated in neighbor aggregation like CNNs. Accordingly, this version is termed GAT-POS-CNN.


Supervised Loss: $\mathcal{L}_{\mathcal{Y}}(\mathbf{h}^{(2)}, \mathbf{y})$
Unsupervised Loss: $\mathcal{L}_{\mathcal{G}}(\mathbf{p}^{(2)}, G)$
$\mathbf{h}^{(2)}$
$\nabla_{\mathbf{h}^{(2)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{G}}$
GAT-Layer-2
Aux-Layer-2
$\mathbf{h}^{(1)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}} \cdot \nabla_{\mathbf{p}^{(2)}}\mathbf{h}^{(1)}$
$\mathbf{p}^{(1)}$
$\nabla_{\mathbf{p}^{(1)}}\mathcal{L}_{\mathcal{G}} + \nabla_{\mathbf{p}^{(1)}}\mathcal{L}_{\mathcal{Y}}$
GAT-Layer-1
Aux-Layer-1
input: $G = (\mathcal{V}, \mathcal{E}), \mathbf{X}$

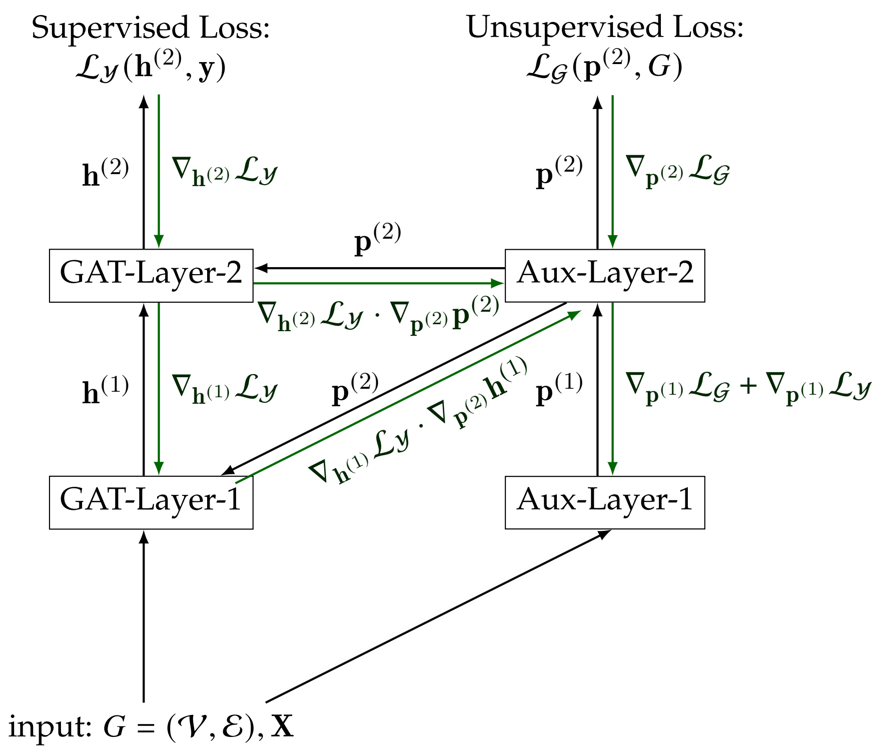
,
where each rectangle denotes the operation of a layer; the black arrows denote forward propagation and the green arrows denote backpropagation; the subscriptions are dropped for simplicity.

Figure 3.1: A Demonstration of GAT-POS-Transformer.

Auxiliary Embedding Models We consider a T-layer auxiliary positional embedding model based on Planetoid-transductive, Planetoid-inductive and GraphSAGE. Note that the positional embeddings of node v from the lth layer of aux-model is denoted as $\mathbf{p}_v^l$.

Main Model with Graph Attentional Layers In this version, we consider the main model with L graph attentional layers, where the lth graph attention layer is defined as follows,

$$\begin{aligned}
\alpha_{vu}^k &= \underset{u \in \mathcal{N}_v \cup \{v\}}{\mathrm{Softmax}}(\mathrm{LeakyReLU}(\mathbf{a}_k^{l\,\intercal}[\mathbf{W}_k^l \mathbf{h}_v + \mathbf{U}_k^l \mathbf{p}_v^T \| \mathbf{W}_k^l \mathbf{h}_u + \mathbf{U}_k^l \mathbf{p}_u^T]) \\
\mathbf{h}_v^l &= \begin{cases} \|_{k=1}^K \sigma(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l) & \text{, if } l < L, \\ \underset{C}{\mathrm{Softmax}}\left(\frac{1}{K}\sum_{k=1}^K \sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l)\right) & \text{, if } l = L. \end{cases}
\end{aligned} \tag{3.9}$$


Supervised Loss: $\mathcal{L}_\mathcal{Y}(\mathbf{h}^{(2)}, \mathbf{y})$
Unsupervised Loss: $\mathcal{L}_\mathcal{G}(\mathbf{p}^{(2)}, G)$
$\mathbf{h}^{(2)}$
$\nabla_{\mathbf{h}^{(2)}}\mathcal{L}_\mathcal{Y}$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_\mathcal{G}$
$\mathbf{p}^{(2)}$
GAT-Layer-2
Aux-Layer-2
$\nabla_{\mathbf{h}^{(2)}}\mathcal{L}_\mathcal{Y} \cdot \nabla_{\mathbf{p}^{(2)}}\mathbf{p}^{(2)}$
$\mathbf{h}^{(1)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_\mathcal{Y}$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_\mathcal{Y} \cdot \nabla_{\mathbf{p}^{(2)}}\mathbf{h}^{(1)}$
$\mathbf{p}^{(1)}$
$\nabla_{\mathbf{p}^{(1)}}\mathcal{L}_\mathcal{G} + \nabla_{\mathbf{p}^{(1)}}\mathcal{L}_\mathcal{Y}$
GAT-Layer-1
Aux-Layer-1
input: $G = (\mathcal{V}, \mathcal{E}), \mathbf{X}$

, where each rectangle denotes the operation of a layer; the black arrows denote forward propagation and the green arrows denote backpropagation; the subscriptions are dropped for simplicity.

Figure 3.2: A Demonstration of GAT-POS-CNN

where $\mathbf{W}_k^l$ and $\mathbf{U}_k^l$ are the weight matrices for node representations and positional embeddings respectively in the kth attention head at the lth layer; $\mathbf{a}_k^l$ is the weight vector in the kth attention head at the lth layer.

Supervised and Unsupervised Losses The supervised and unsupervised losses in the version are the same as those introduced before.

A demonstration of GAT-POS-CNN with two-layer main model and two-layer aux-model ($T = L = 2$) is shown in Figure 3.2.

GAT-POS-CG

Recalling the GAT-POS-CNN, the bases of the attention mechanism at each layer is the node representations at the previous layer and the positional embeddings shared across layers. Unlike GAT-POS-Transformer, the positional embeddings will not be updated as parts of the node representations. Even though the graph structure retains across layers, each node representation is actually representing a number of nodes aggregated previously and the corresponding structural information might be different at different layers. This is corresponding to the concept of receptive fields of each neuron in CNNs on images.

In other words, after neighborhood aggregation, the corresponding structural information between each node pair is supposed to be corresponding to different order connectivities at different layers.

Therefore, we introduce two variants of GAT-POS-CNN to satisfy this hypothesis.

The first variant proposed is to update the positional embeddings for different graph attentional layers with GNNs (i.e., GraphSAGE in our case) in the auxiliary embedding model. From the perspective of receptive fields, we tend to utilize GraphSAGE to make the receptive fields of the positional embeddings corresponding to the receptive fields of the node preresntations at different layers.

We design the auxiliary embedding model with a combination of Planetoid-inductive and GraphSAGE, in which the unsupervised loss, modeling the first-order proximity in the graph, is applied to a hidden layer of the auxiliary model. We term this variant as GAT-POS-CNN with GNN updated Positional Embedding (shortened as GAT-POS-CG).

Auxiliary Embedding Models In order to validate if the receptive field of the positional embeddings needs to correspond to the receptive field of the node representations, we consider an $(T+L)$-layer auxiliary model which fuses Planetoid-inductive and GraphSAGE.

Intuitively, we need to apply the unsupervised loss capturing the first-order proximity of each node-pair, to the hidden layer before GraphSAGE layers. Therefore, considering the inductive learning, we let the first T layers of auxiliary model follow the Planetoid-inductive architecture while the last L layers follow the architecture of GraphSAGE. Thus, the unsupervised loss is applied to the positional embeddings at Tth layer, learning the first-order proximity of nodes in the graph. The following GraphSAGE layers serve as an encoding function to produce positional embeddings that cover different hops neighborhoods, corresponding to the main model. Note that, the unsupervised loss would not back-propagate gradients to the last L layers of the auxiliary model. The last L layers (GraphSAGE layers) will only receive gradients from the supervised loss, and be guided to encode the positional embeddings. The positional embeddings of node v from the lth layer of auxiliary model is denoted as $\mathbf{p}_v^l$.

Main Model with Graph Attentional Layers Corresponding to the $(T + L)$-layer auxiliary model, we consider a main model with $(L + 1)$ graph attentional layers. The lth graph attentional layer $(1 \leq l \leq L + 1)$ is defined as follows,

$$\begin{aligned}
\alpha_{vu}^k &= \underset{u \in \mathcal{N}_v \cup \{v\}}{\mathrm{Softmax}}(\mathrm{LeakyReLU}({\mathbf{a}_k^l}^\top [\mathbf{W}_k^l \mathbf{h}_v + \mathbf{U}_k^l \mathbf{p}_u^{T+l-1} \| \mathbf{W}_k^l \mathbf{h}_u + \mathbf{U}_k^l \mathbf{p}_u^{T+l-1}]) \\
\mathbf{h}_v^l &= \begin{cases} \|_{k=1}^K \sigma(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l) & \text{, if } l < L + 1, \\ \underset{C}{\mathrm{Softmax}} \left(\frac{1}{K} \sum_{k=1}^K \sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l \right) & \text{, if } l = L + 1. \end{cases}
\end{aligned} \tag{3.10}$$

where $\mathbf{W}_k^l$ and $\mathbf{U}_k^l$ are the weight matrices for node representations and positional embeddings respectively in the kth attention head at the lth layer; $\mathbf{a}_k^l$ is the weight vector in the kth attention head at the lth layer.

Supervised Loss The definition of supervised loss is the same as the one in GAT-POS-CNN.

Unsupervised Loss The unsupervised loss utilized in GAT-POS-CG follows the same goal and definition of the previous ones. However, this unsupervised loss is applied to the positional embeddings from the Tth layer of the auxiliary model.

$$\mathcal{L}_{\mathcal{G}}(\{\mathbf{p}_v^T\}_{v\in\mathcal{V}}, G)) = \sum_{v\in\mathcal{V}} \sum_{u\in\mathcal{N}(v)} \Big(-\log\sigma(\mathbf{p}_v^{T\top}\mathbf{p}_u^T) - Q\cdot\mathbb{E}_{u'\sim P_n(v)}\log(\sigma(-\mathbf{p}_v^{T\top}\mathbf{p}_{u'}^T))\Big) \quad (3.11)$$

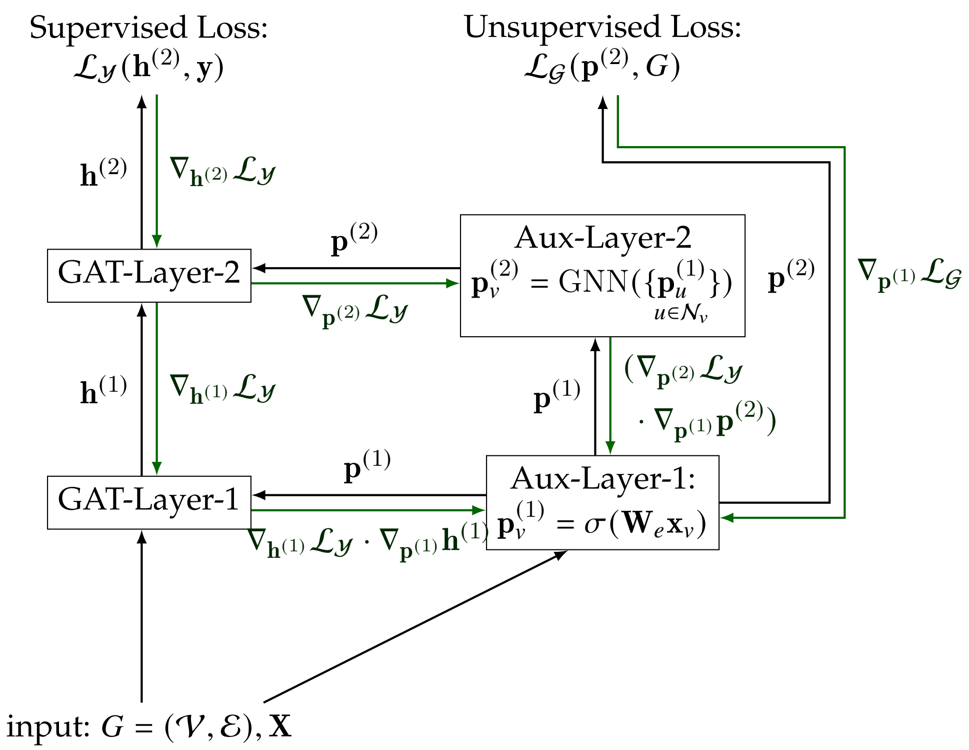
where $P_n(v)$ denotes the distribution negative sampling for node v and Q is the number of negative samples per edge (i.e., for a neighbor of node v, sample Q non-adjacent nodes of node v as negative samples).

A demonstration of the GAT-POS-CG with a two-layer main model and a two-layer auxiliary model ($T = 1, L = 1$) is shown in Figure 3.3.

GAT-POS-CML

Another variant of GAT-POS-CNN proposed is to assign an unsupervised loss to each GAT layer in order to capture positional embeddings regarding different receptive fields of each node at different layers. In this variant, instead of encoding positional embeddings with GraphSAGE, we learn positional embeddings capturing different order of proximities at different layers by controlling the sets of positive neighbors and negative neighbors for each node in unsupervised losses at different layers. We term this variant as GAT-POS-CNN with Multiple Unsupervised Losses (shortened as GAT-POS-CML).

Auxiliary Embedding Models Similar to the auxiliary model in GAT-Pos-CG, in this version, we consider a $(T + L)$-layer auxiliary model. To validate that assigning different unsupervised losses to different layers can learn different positional embeddings capturing different order of proximity, we utilize the architecture of Planetoid-inductive to get rid of the effect of encoding positional embeddings from GraphSAGE.


Supervised Loss: $\mathcal{L}_{\mathcal{Y}}(\mathbf{h}^{(2)}, \mathbf{y})$
Unsupervised Loss: $\mathcal{L}_{\mathcal{G}}(\mathbf{p}^{(2)}, G)$
$\mathbf{h}^{(2)}$
$\nabla_{\mathbf{h}^{(2)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(2)}$
GAT-Layer-2
Aux-Layer-2
$\mathbf{p}_v^{(2)} = \mathrm{GNN}(\{\mathbf{p}_u^{(1)}\}_{u\in\mathcal{N}_v})$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{p}^{(1)}}\mathcal{L}_{\mathcal{G}}$
$\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{h}^{(1)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(1)}$
$(\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{Y}} \cdot \nabla_{\mathbf{p}^{(1)}}\mathbf{p}^{(2)})$
$\mathbf{p}^{(1)}$
GAT-Layer-1
Aux-Layer-1:
$\mathbf{p}_v^{(1)} = \sigma(\mathbf{W}_e\mathbf{x}_v)$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}} \cdot \nabla_{\mathbf{p}^{(1)}}\mathbf{h}^{(1)}$
input: $G = (\mathcal{V}, \mathcal{E}), \mathbf{X}$

, where each rectangle denotes the operation of a layer; the black arrows denote forward propagation and the green arrows denote backpropagation, the subscriptions are dropped for simplicity.

Figure 3.3: A Demonstration of GAT-POS-CG

There will be an unsupervised loss function capturing the lth order of proximities of nodes based on the positional embeddings at $(T + l - 1)$th layer of the auxiliary model, starting from the Tth layer. The lth order of proximities can be understood as whether a node u is in the lth-hop neighborhood of node v. The positional embeddings of node v from the lth layer of auxiliary model is denoted as $\mathbf{p}_v^l$.

Main Model with Graph Attentional Layers Considering the $(T+L)$-layer auxiliary model, the main model consists of $(L + 1)$ graph attentional layers. Corresponding to the positional embeddings from the $(T + l - 1)$th layer of auxiliary model, the lth graph attentional layer ($1 \leq l \leq L+1$) is defined as follows,

$$\begin{aligned}
\alpha_{vu}^{k} &= \underset{u \in \mathcal{N}_v \cup \{v\}}{\mathrm{Softmax}}(\mathrm{leakyrelu}(\mathbf{a}_k^{l\top}[\mathbf{W}_k^l \mathbf{h}_v + \mathbf{U}_k^l \mathbf{p}_v^{T+l-1} \| \mathbf{W}_k^l \mathbf{h}_j + \mathbf{U}_k^l \mathbf{p}_u^{T+l-1}]) \\
\mathbf{h}_v^l &= \begin{cases} \|_{k=1}^{K} \sigma(\sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l) & \text{, if } l < L+1, \\ \underset{C}{\mathrm{Softmax}} \left(\frac{1}{K} \sum_{k=1}^{K} \sum_{u \in \mathcal{N}_v \cup \{v\}} \alpha_{vu}^k \cdot \mathbf{W}_k^l \mathbf{h}_u^l \right) & \text{, if } l = L+1. \end{cases}
\end{aligned} \tag{3.12}$$

where $\mathbf{W}_k^l$ and $\mathbf{U}_k^l$ are the weight matrices for node representations and positional embeddings respectively in the kth attention head at the lth layer; $\mathbf{a}_k^l$ is the weight vector in the kth attention head at the lth layer.

Supervised Loss The supervised loss is the same as the previous ones.

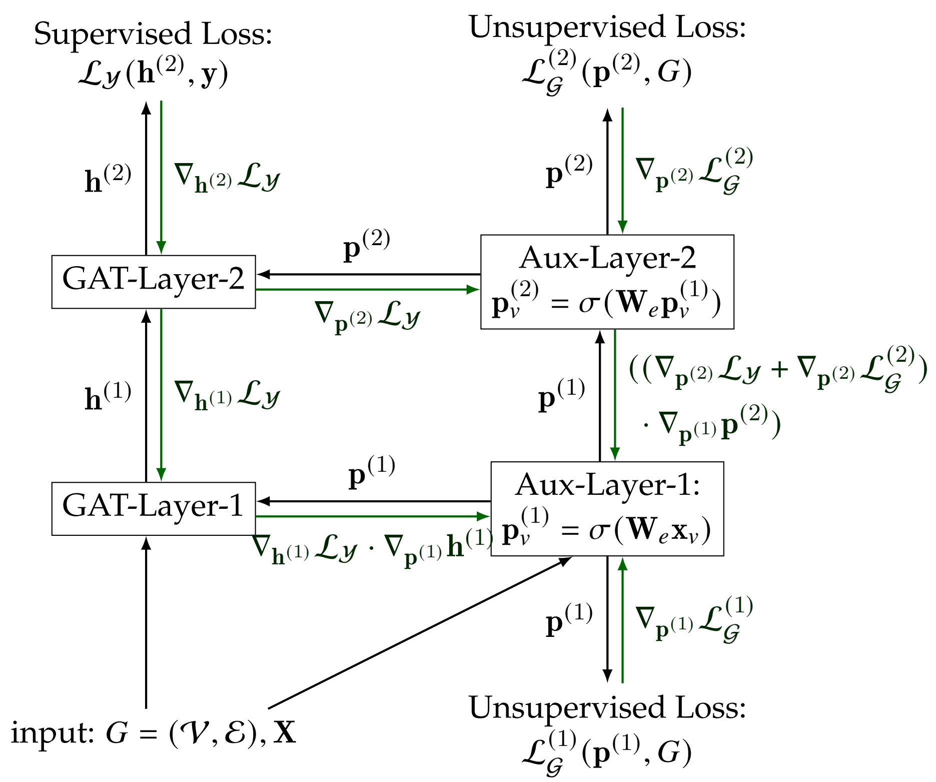
Unsupervised Loss The unsupervised losses are following the goal and definition as the previous ones. The unsupervised loss here will be adapted layer-wise to match the receptive fields at the corresponding layer of the main mode. In particular, we consider an l-hop neighborhood in the unsupervised loss at $(T+l-1)$th layer of auxiliary model.

$$\mathcal{L}_{\mathcal{G}}^{(l)}(\{\mathbf{p}_v^{T+l-1}\}_{v \in \mathcal{V}}, G)) = \sum_{v \in \mathcal{V}} \sum_{u \in \mathcal{N}^h(v)} \Big(-\log \sigma(\mathbf{p}_v^{T+l-1\top} \mathbf{p}_u^{T+l-1}) - Q \cdot \mathbb{E}_{u' \sim P_n(v)} \log(\sigma(-\mathbf{p}_v^{T+l-1\top} \mathbf{p}_{u'}^{T+l})) \Big) \tag{3.13}$$

where $\mathcal{N}^l(v)$ denotes the lth-hop neighborhood of node v.

By considering the positive neighbors of node v as the nodes in $\mathcal{N}^l(v)$, the unsupervised loss can guide the positional embeddings at the corresponding layers to learn the lth order proximities.

A demonstration of GAT-POS-CML with a two-layer main model and a two-layer auxiliary model ($T = 1, L = 1$) is shown in Figure 3.4.


Supervised Loss:
$\mathcal{L}_{\mathcal{Y}}(\mathbf{h}^{(2)}, \mathbf{y})$
Unsupervised Loss:
$\mathcal{L}_{\mathcal{G}}^{(2)}(\mathbf{p}^{(2)}, G)$
$\mathbf{h}^{(2)}$
$\nabla_{\mathbf{h}^{(2)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(2)}$
$\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{G}}^{(2)}$
GAT-Layer-2
Aux-Layer-2
$\mathbf{p}_v^{(2)} = \sigma(\mathbf{W}_e\mathbf{p}_v^{(1)})$
$\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{h}^{(1)}$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}}$
$\mathbf{p}^{(1)}$
$((\nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{Y}} + \nabla_{\mathbf{p}^{(2)}}\mathcal{L}_{\mathcal{G}}^{(2)}) \cdot \nabla_{\mathbf{p}^{(1)}}\mathbf{p}^{(2)})$
GAT-Layer-1
Aux-Layer-1:
$\mathbf{p}_v^{(1)} = \sigma(\mathbf{W}_e\mathbf{x}_v)$
$\nabla_{\mathbf{h}^{(1)}}\mathcal{L}_{\mathcal{Y}} \cdot \nabla_{\mathbf{p}^{(1)}}\mathbf{h}^{(1)}$
$\nabla_{\mathbf{p}^{(1)}}\mathcal{L}_{\mathcal{G}}^{(1)}$
input: $G = (\mathcal{V}, \mathcal{E}), \mathbf{X}$
Unsupervised Loss:
$\mathcal{L}_{\mathcal{G}}^{(1)}(\mathbf{p}^{(1)}, G)$

,
where each rectangle denotes the operation of a layer; the black arrows denote forward propagation and the green arrows denote backpropagation, the subscriptions are dropped for simplicity.

Figure 3.4: A Demonstration of GAT-POS-CML

3.4 Comparison with Related Works

The most closely related works of our proposed approach are the very recently introduced Geom-GCN [Pei et al., 2020] and DE for GNNs [Li et al., 2020b], both of which we discussed in section 2.6.

3.4.1 Comparison with Geom-GCN

There are three differences between Geom-GCN and GAT-POS. First, the embedding model in GAT-POS is jointly trained with the main architecture simultaneously as an auxiliary model, which allows the positional embeddings to capture structural information of the graph while taking the downstream task into consideration. With the joint-training, GAT-POS supports end-to-end training like typical GNNs.

Second, instead of applying an adaptive filter with a weighting function based on structural embeddings, we utilize a graph attentional aggregator, which incorporates both semantic information and structural and positional information. It is worth mentioning that the attention mechanisms have been proven to benefit from both kinds of information in language modeling and image classification tasks [Vaswani et al., 2017, Zhao et al., 2020]. In addition, some previous works have shown that feature extractors considering both types of information are more expressive than those only considering structural, positional, or locational information in image classification tasks [Zhao et al., 2020,Cordonnier et al., 2020].

Third, for a node v, Geom-GCNs propose an extended neighborhood that includes nodes near node v in the space of positional embeddings as extra neighbors in aggregation. The extended neighborhood scheme assumes that the nearness in the embedding space indicates a potential interaction between two nodes. However, we are not sure how extending a neighborhood will affect the attention coefficients computed since there is a normalization over the neighborhood in attention computation. Thus, we leave it as a potential future direction.

3.4.2 Comparison with Distance Encoding

Compared to the embedding models in Geom-GCNs and GAT-POS, DE in DEAGNN and DEGNN is an encoding approach without training with respect to an objective function. This is more close to the positional encoding utilized in Transformer [Vaswani et al., 2017] and many variants of it [Dai et al., 2019,Cordonnier et al., 2020,Zhao et al., 2020]. Advantageously, DE enjoys a better generalization for inductive learning because it can directly apply to unseen examples without retraining on them or generalizing previously learned information to them based on node attributes [1]. For instance, in the most extreme case that there are no node attributes available, DE can still support inductive learning.

However, without training with objective functions, intuitively, the structural information captured by distance encoding might be limited and highly dependent on the choice of the encoding scheme.

[1]The generalization of a model to unseen examples usually requires the observed examples and unseen examples share properties of the same feature space.

Chapter 4

Experiments

4.1 Models in Comparison

In the experiments, we include three models, GCN [Kipf and Welling, 2017], GAT [Velickovic et al., 2018] and Geom-GCN [Pei et al., 2020], into the comparison as baselines. Firstly, through experiments, we show that our proposed approach can effectively enhance GATs by fully exploring and balancing the structural information and the semantic information. Thus, we include the standard GAT as one of our baselines in order to validate our claims. Furthermore, the GCN is also included as a representative of spectral GNNs based on graph Laplacian.

Moreover, we aim to emphasize the benefits of allowing the attention mechanism of GAT to consider both node content representations and the positional ones, as well as the joint training of positional embeddings and the main architectures. Thus, the Geom-GCN, an enhancement framework with a similar architecture to GAT-POS, is included in the comparison as our direct opponent. Specifically, three variants of Geom-GCNs coupled with different embedding methods are considered, namely Geom-GCN with Isomap (Geom-GCN-I), Geom-GCN with Poincare embedding (Geom-GCN-P), and Geom-GCN with struc2vec embedding (Geom-GCN-S).

GAT-POS-T-GS	GAT-POS-T-PI	GAT-POS-T-PT
GAT-POS-C-GS	GAT-POS-C-PI	GAT-POS-C-PT
GAT-POS-CG-GS	GAT-POS-CML-PI	GAT-POS-CML-PT

where the GAT-POS-T and GAT-POS-C denote GAT-POS-Transformer and GAT-POS-CNN, respectively. The suffixes GS, PI and PT indicate the auxiliary models based on GraphSAGE, Planetoid-inductive and Planetoid-transductive, respectively.
Note that for GAT-POS-CG and GAT-POS-CML, the auxiliary models are specially defined; details are discussed in section 3.3.3.

Table 4.1: Table of GAT-POS Variants in Comparison.

Since the work from [Li et al., 2020b] was under review and no public code was available at the time of writing this thesis and the experiments of the thesis were already reaching the end, we did not include GNNs enhanced by Distance Encoding (DE) in our comparisons considering the submission time limit.

To fully understand our framework through experiments, we consider the two main versions of our framework, GAT-POS-Transformer and GAT-POS-CNN, coupled with three choices of auxiliary embedding models, based on Planetoid-transductive, Planetoid-inductive, and GraphSAGE, respectively. Besides the comparison between different versions, we also focus on comparing different auxiliary embedding models. The comparison between GraphSAGE and Planetoid-inductive illustrates whether considering graph structure during inference can preserve more useful positional information when parameterizing the positional embeddings to a function of node attributes. In order to measure the information loss due to the parameterization to a function of node attributes in inductive auxiliary embedding models, the Planetoid-transductive is also included in the comparison. Additionally, we also include two variants of GAT-POS-CNN, GAT-POS-CG and GAT-POS-CML, coupled with their corresponding auxiliary embedding models.

In total, we consider nine variants of our frameworks, as shown in Table 4.1.

4.2 Datasets

Recalling the previous discussion about typical GNNs and our proposed frameworks in section 3.1, we aim to show through experiments that the framework of GAT-POS can enhance the capacity of GATs to learn on non-homophilic graphs. Meanwhile, GAT-POS is expected to maintain a comparable performance of typical GNNs on homophilic graphs.

As previously discussed, homophilic graph-structured datasets, in which edges indicate semantic similarity between nodes, are relatively less challenging for typical GNNs. Typical GNNs, including spectral and spatial GNNs, have been proven effective on homophilic graphs, owe to their neighborhood aggregation architectures. We include three widely used citation networks (i.e., Cora, Citeseer and Pubmed) [Yang et al., 2016, Kipf and Welling, 2017, Velickovic et al., 2018] as representatives of homophilic datasets. Since typical GNNs have already performed well on homophilic datasets, the goal is to give a sanity check that our proposed approach can reach a comparable performance in this setting.

The non-homophilic graph-structured datasets are more challenging since the edges within them only indicate the existence of interactions rather than semantic similarity. Under this setting, GNNs with filters blurring each local area might fail to capture local phenomena if connected nodes are semantically similar in their representations. Without exploiting the structural, positional and/or locational information, GNNs with content-based adaptive filters, e.g. GATs, might end up with a blurring kernel and suffer from the non-homophilic setting. As such, we anticipate that the positional embeddings of GAT-POS will prove themselves especially beneficial for non-homophilic datasets. Thus, we focus on the experiments on non-homophilic datasets to demonstrate that our proposed approach is indeed able to enhance the capacity of GAT under non-homophilic setting by combining semantic and structural information. We consider two Wikipedia page-page networks (i.e., Chameleon and Squirrel) and the Actor co-occurrence network (shortened as Actor) following [Pei et al., 2020] as our non-homophilic datasets.

The homophily level of those datasets can be summarized by the mean of the index introduced in [Pei et al., 2020], which is computed as follows,

$$\beta = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \frac{\#\ v\text{'s neighbors who have the same labels as } v}{\#\ v\text{'s neighbors}}. \tag{4.1}$$

This index measures the similarity between a node and its neighbors in terms of node labels. A large β value implies that the graph is homophilic. Note that a small β value can only indicate the non-homophily rather than heterophily (the tendency to connect semantically different nodes). Correspondingly, it is worth mentioning that an edge in non-homophilic datasets indicates the existence of interaction or relationship rather than the semantic dissimilarity.

The homophily measure values of the datasets in our experiments are summarized in Table 4.2 to justify our choices of homophilic and non-homophilic datasets.

Dataset	Homophilic Datasets			Non-homophilic Datasets		
	Cora	Citeseer	Pubmed	Chameleon	Squirrel	Actor
β	0.83	0.71	0.79	0.25	0.22	0.24

Table 4.2: The Homophily Measure β on Datasets in our Experiments.

The basic statistics of datasets in experiments are provided in Table 4.3.

	Cora	Citeseer	Pubmed	Chameleon	Squirrel	Actor
# Nodes	2708	3327	19717	2277	5201	7600
# Edges*	5429	4732	44338	36101	217073	33544
# Features	1433	3703	500	2325	2089	931
# Classes	7	6	3	5	5	5

*: the directed graphs are converted to undirected graphs in preprocessing.

Table 4.3: Summary of the Datasets Used in our Experiments.

Note that, following the vast majority of the graph neural network literature, we converted all graphs to undirected graphs in the experiments.

4.3 Experimental Setup

For all graph datasets, we randomly split nodes of each class into 60%, 20%, and 20% for training, validation and testing. The final experimental results reported are the average classification accuracies of all models on the test sets over 10random splits for all graph datasets. This is similar to the experimental setup in [Pei et al., 2020].

4.3.1 Hyperparameter Setting

We utilize the final hyperparameter settings of baseline models from [Pei et al., 2020] in our experiments, since they have already done extensive hyperparameter search for all methods on the validation set of each dataset. For our proposed GAT-POS, we perform hyperparameter search on the number of hidden units, the initial learning rate, the weight decay, and the probability of dropout. For fairness, the hyperparameter search is performed on the same searching space as the models tuned in [Pei et al., 2020]. Specifically, the number of layers of GNN architectures is fixed to 2 and the Adam optimizer [Kingma and Ba, 2014] is used to train all models.

Final Hyperparameter Setting of Baseline Models

The final hyperparameter setting is dropout of $p = 0.5$, initial learning rate of 0.05, patience of 100 epochs, and weight decay of 5e-5.

In GCN, the number of hidden units is 16 (Cora and Citeseer), 64 (Pubmed), 48 (Chameleon and Squirrel), and 32 (Actor). Note that, in Geom-GCN, the number of hidden units is 8 times as many as the number of hidden units in GCN since Geom-GCN has 8 virtual nodes (virtual relations) [Pei et al., 2020]. For each attention head in GAT, the number of hidden units is 8 (Cora, Citeseer, Pubmed), 48 (Chameleon and Squirrel), and 32 (Actor). GAT has 8 attention heads in the hidden layer for all datasets, 8 (Pubmed), or 1 (the other datasets) attention heads in the output layer. We use ReLU as the activation for Geom-GCN and GCN, and ELU for GAT.

Hyperparameter Setting of GAT-POS

In GAT-POS, we tune the hyperparameters, including the probabilities of dropout, the initial learning rate, the weight decay, the number of attention heads in each layer of the main model, as well the number of layers, the number of hidden units and the activation of the main model and the auxiliary model. For fair comparison, we follow the constraint on the main architecture from [Pei et al., 2020] and set the number of layers as 2 for both the main architecture and the auxiliary embedding model.

For the hyperparameters of the probabilities of dropout, the initial learning rate, the weight decay and the activation, we perform a manual hyperparameter search according to previous works [Kipf and Welling, 2017, Velickovic et al., 2018] as well as prior empirical evidence. The hyperparameter tuning for the auxiliary embedding models is performed similarly based on the results from [Yang et al., 2016,Hamilton et al., 2017]. The manual hyperparameter search is conducted by attempting some previously utilized settings of hyperparameters without searching algorithms. Even though the searching space of hyperparameters is smaller, considering the limited time and computation resources, we perform a manual hyperparameter search on those less sensitive hyperparameters.

For the hyperparameters of the main architectures (i.e., modified GATs), including the number of hidden units, the number of attention heads at each layer and residual connection, the hyperparameter search is performed with a hyperparameter optimization algorithm called Asynchronous Successive Halving Algorithm (ASHA) [Li et al., 2020a][1]. Note that the hyperparameter searching space of the main architecture is the same as the models tuned in [Pei et al., 2020] for fairness.

[1] Also known as AsyncHyperBand, the Async version of HyperBand [Li et al., 2017]

The hyperparameter search is performed monitoring the validation loss. The splitting scheme of datasets in the hyperparameter search is introduced in detail as follows.

Data Splitting in Hyperparameter Search The hyperparameter search is supposed to be performed monitoring the classification accuracies on the validation set.

However, considering that we are following the random splitting scheme of [Pei et al., 2020] in the final evaluation, in order to avoid overfitting a certain data split or being exposed to the test set, we introduce the following data splitting scheme for hyperparameter search for datasets without a fixed public split. Note that the hyperparameter search is performed for each version of GAT-POS on each dataset independently.

For each variant of GAT-POS on each dataset, the hyperparameter search is performed as follows. First, we get 10 splits by random splitting for the final evaluation, which are utilized by all models. In particular, we use the splits provided by [Pei et al., 2020] for a fair comparison with other baselines. For each aforementioned split, termed outer-split, the test set is retained and ignored during the hyperparameter search. Then, we would get 5 random splits for training and validation with the test set retained, termed inner split, for tuning hyperparameters. Afterward, a hyperparameter search is performed on each outer split and each inner split independently.

The best configuration of hyperparameters of a variant of GAT-POS on a dataset is decided based on the average validation losses over all inner and outer splits.

It is worth mentioning that the hyperparameter search by ASHA strictly follows the scheme aforementioned, while the manual search might not go over all inner and outer splits for some GAT-POS variants and datasets.

Hyperparameter Searching Space As previously discussed, the searching spaces of hyperparameters is decided according to [Pei et al., 2020, Kipf and Welling, 2017, Velickovic et al., 2018, Yang et al., 2016, Hamilton et al., 2017] and prior empirical evidence. Note that the searching space of the main architecture strictly matches the setting in [Pei et al., 2020] for fairness.

Hyperparameter	Searching Space	Searching Method
Initial learning rate	$[5\text{e-}1, 5\text{e-}2, 5\text{e-}3]$	Manual
Weight decay	$[5\text{e-}3, 5\text{e-}4, 5\text{e-}5, 5\text{e-}6]$	
Dropout probabilty	$[0, 0.5]$	
# hidden units (auxiliary)	$[16, 32, 64, 128]$	
Activation (auxiliary)	[ReLU, ELU]	
Activation (main)	[ReLU, ELU]	
# hidden units (main)*	$[8, 16, 32, 48, 64]$	ASHA
# attention heads (main)	hidden-layer: $[1, 4, 8, 12, 16]$ output-layer: $[1, 4, 8, 12, 16]$	
Residual connection (main)	[True, False]	

where main and auxiliary denotes the hyperparameters for main architecture and auxiliary embedding model respectively.
*: for graph attentional layers in main architecture, # hidden units is for each attention head.

Table 4.4: Summary of Searching Spaces for Hyperparameters

We provide a summary of searching spaces for hyperparameters, as shown in Table 4.4.

Final Hyperparameter Setting of GAT-POS According to the result of the manual hyperparameter search, we set the initial learning rate as 5e-3, weight decay as 5e-4, the dropout of $p = 0.5$ and the number of hidden units for the auxiliary model as 64 for all variants of GAT-POS on all datasets in our experiments. Besides, the activation functions of the auxiliary model and the main architecture are set as ReLU and ELU, respectively.

We notice that the different GAT-POS variants almost shared the same optimal hyperparameter setting according to the hyperparameter searching results from ASHA[2]. Thus, we set all GAT-POS variants with the same hyperparameter setting, summarized in Table 4.5.

Similar to the setting in [Pei et al., 2020], we also use an early stopping strategy on the cross-entropy loss on the validation set, with a patience of 100 epochs.

[2]Some variants have several configurations of hyperparameters with tied performance.

Hyperparameter	Cora	Citeseer	Pubmed	Chameleon	Squirrel	Actor
# hidden units	8	8	8	32	8*	32
# attention heads	$[8, 1]$	$[8, 1]$	$[8, 1]$	$[16, 1]$	$[16, 1]$	$[16, 1]$
Residual connection	True	True	True	True	True	True

where $[8, 1]$ in # attention heads denotes 8 attention heads in hidden layer and 1 attention head in output layer.
*: some configuration of hyperparameters are ignored due to out of GPU memory.

Table 4.5: Summary of the Final Hyperparameters Setting of GAT-POS

4.4 Experimental Results

As previously mentioned, for each model and dataset, experiments are run with 10 random splits with splitting ratios $60\%, 20\%$ and 20% for training, validation and testing respectively. The specific splits utilized are provided by [Pei et al., 2020]. For each split, we have 10 runs of experiments with different random seeds for model initialization.

The results of our comparative evaluation experiments are summarized in Table 4.6 and 4.7. We report the mean classification accuracy (with standard deviation) on the test set.

Results on Homophilic Datasets The experimental results on homophilic graph datasets are presented in Table 4.6.

On the homophilic datasets in our experiments, most instantiations of GAT-POS reach comparable results with typical GNNs. Note that although there are on average differences across different GAT-POS variants and typical GNNs, most results are within the standard deviation of each other. One interesting exception to note might be GAT-POS-T-PT, which is worse than other variants of GAT-POS. According to the previous discussion and the experimental results on non-homophilic graphs discussed in the following, we argue that transductive auxiliary embedding models can learn more dedicated positional embeddings. Therefore, we suspect that injecting positional embeddings from Planetoid-transductive directly to the node representations leads to redundant structural informa-

	Homophilic Datasets		
Accuracy	Cora	Citeseer	Pubmed
GCN	$85.67 \pm 0.94\%$	$73.28 \pm 1.37\%$	$88.14 \pm 0.32\%$
GAT	$87.06 \pm 0.98\%$	$74.79 \pm 1.89\%$	$87.51 \pm 0.43\%$
GeomGCN-I	$84.79 \pm 2.04\%$	$\mathbf{78.84 \pm 1.51}\%$	$\mathbf{89.73 \pm 0.54}\%$
GeomGCN-P	$84.68 \pm 1.59\%$	$73.77 \pm 1.59\%$	$88.15 \pm 0.57\%$
GeomGCN-S	$85.25 \pm 1.46\%$	$74.42 \pm 2.52\%$	$84.80 \pm 0.62\%$
GAT-POS-T-GS	$\mathbf{87.90 \pm 0.97}\%$	$75.30 \pm 1.10\%$	$88.17 \pm 0.54\%$
GAT-POS-T-PI	$85.90 \pm 1.47\%$	$72.88 \pm 1.98\%$	$86.41 \pm 0.69\%$
GAT-POS-T-PT	$82.32 \pm 1.75\%$	$66.68 \pm 2.01\%$	$84.68 \pm 0.82\%$
GAT-POS-C-GS	$87.31 \pm 1.40\%$	$74.71 \pm 1.33\%$	$87.59 \pm 0.38\%$
GAT-POS-C-PI	$87.29 \pm 1.52\%$	$74.67 \pm 1.31\%$	$87.57 \pm 0.48\%$
GAT-POS-C-PT	$86.61 \pm 1.13\%$	$73.81 \pm 1.27\%$	$87.56 \pm 0.48\%$
GAT-POS-CG-GS	$87.01 \pm 1.64\%$	$74.70 \pm 1.31\%$	$87.58 \pm 0.51\%$
GAT-POS-CML-PI	$87.60 \pm 1.69\%$	$74.77 \pm 1.34\%$	$87.58 \pm 0.48\%$
GAT-POS-CML-PT	$86.25 \pm 1.29\%$	$74.65 \pm 2.01\%$	$87.53 \pm 0.48\%$

The best performing method is highlighted.

Table 4.6: Summary of Results in terms of Classification Accuracies for Homophilic Datasets.

tion on homophilic datasets. Thus, GAT-POS-T-PT suffers from a more severe overfitting problem than other variants.

These observations are in line with our expectations on homophilic datasets since merely learning underlying group-invariances can lead to good performance on graph datasets with high homophily. The evaluation experiments on homophilic graphs serve as a good sanity check for the proposed approach prior to verifying its potential on more challenging non-homophilic datasets.

Note that the performance improvements of GeomGCNs are highly dependent on the choices of embedding models. Overall, GeomGCN-I outperforms all other models on average on Citeseer, but tend to be on par with their baseline (GCN) on other homophilic datasets. Moreover, for Cora, GeomGCN seems to exhibit higher standard deviations than the baselines.

	Non-Homophilic Datasets		
Accuracy	Chameleon	Squirrel	Actor
GCN	$65.22 \pm 2.22\%$	$45.44 \pm 1.27\%$	$28.30 \pm 0.73\%$
GAT	$63.88 \pm 2.42\%$	$41.19 \pm 3.38\%$	$28.49 \pm 1.06\%$
GeomGCN-I	$57.35 \pm 1.85\%$	$31.92 \pm 1.04\%$	$29.14 \pm 1.15\%$
GeomGCN-P	$60.68 \pm 1.97\%$	$35.39 \pm 1.21\%$	$31.92 \pm 0.95\%$
GeomGCN-S	$57.89 \pm 1.65\%$	$35.74 \pm 1.47\%$	$30.12 \pm 0.92\%$
GAT-POS-T-GS	$65.99 \pm 2.66\%$	$47.79 \pm 2.16\%$	$34.75 \pm 1.16\%$
GAT-POS-T-PI	$63.79 \pm 2.35\%$	$47.96 \pm 3.29\%$	$34.59 \pm 1.01\%$
GAT-POS-T-PT	$65.55 \pm 2.38\%$	$51.62 \pm 1.84\%$	$34.97 \pm 1.27\%$
GAT-POS-C-GS	$64.39 \pm 2.16\%$	$43.56 \pm 3.15\%$	$34.78 \pm 1.00\%$
GAT-POS-C-PI	$67.65 \pm 2.82\%$	$42.58 \pm 1.66\%$	$34.65 \pm 1.21\%$
GAT-POS-C-PT	$\mathbf{67.76 \pm 2.54}\%$	$\mathbf{52.90 \pm 1.55}\%$	$34.89 \pm 1.38\%$
GAT-POS-CG-GS	$65.92 \pm 2.29\%$	$48.39 \pm 1.61\%$	$34.89 \pm 1.08\%$
GAT-POS-CML-PI	$66.67 \pm 2.15\%$	$47.44 \pm 2.33\%$	$34.98 \pm 1.04\%$
GAT-POS-CML-PT	$67.48 \pm 3.08\%$	$50.94 \pm 2.77\%$	$\mathbf{35.03 \pm 1.44}\%$

The best performing method is highlighted.

Table 4.7: Summary of Results in terms of Classification Accuracies for Non-homophilic Datasets.

Results on Non-homophilic Datasets The experimental results on non-homophilic graph datasets, summarized in Table 4.7, demonstrate that our framework reaches better performance than baselines in general.

Unexpectedly, GeomGCNs do not show significant improvement compared to typical GNNs on undirected graphs. One possible reason is that their extended neighborhood scheme cannot include extra useful neighbors in the aggregation, but rather introduces redundant operations and parameters to train on undirected graphs.

Concerning the different architectures of GAT-POS, GAT-POS-C slightly outperforms the other variants overall. The potential reason is that the learned semantic node representations are not stained by injecting positional embeddings explicitly.

Regarding the choices of the auxiliary embedding models, there is no remarkable difference between GraphSAGE and Planetoid-inductive according to the experimental results.

Even though the datasets in our experiments are under the transductive setting, we involve the comparisons between the transductive-based and inductive-based embedding models to understand the information loss due to the parameterization of positional embeddings as a function of node attributes with or without graph structures. On Chameleon and Actor datasets, the three auxiliary embedding models have comparable results, which illustrates that the framework of GAT-POS has the potential to support the inductive setting.

However, on Squirrel, the auxiliary embedding models based on Planetoid-transductive remarkably outperform the inductive-based auxiliary embedding models. These observations are probably because the space of node attributes is insufficient to support the parameterized function to output positional embeddings. For instance, the auxiliary embedding models based on Planetoid-inductive will fail to produce different positional embeddings for two nodes with the same node attributes. Thus, supporting inductive learning when the node attributes are not informative enough will be the future research direction of our framework.

Even though with shortcomings in supporting the inductive setting under certain conditions, the experimental results on the non-homophilic datasets illustrate the enhanced ability of our proposed approach to learn on non-homophilic graphs.

4.5 Ablation Study on Joint Training

The framework of GAT-POS consists of two parts, the main model with graph attentional layers incorporating positional embeddings and the auxiliary model learning the positional embeddings jointly trained with the main model. One remarkable difference between GAT-POS and the previous work Geom-GCN is that we utilize a joint-training scheme for the support of end-to-end training as well as the extra guidance to the embedding model from the supervised task. In order to understand the contribution from each component of GAT-POS, we present an ablation study to evaluate the contributions from

Datasets	Chameleon		Squirrel		Actor	
Joint-Training	True	False	True	False	True	False
GAT-POS-T-GS	**65.99 ± 2.66%**	62.94 ± 1.88%	**47.79 ± 2.16%**	42.50 ± 1.74%	34.75 ± 1.16%	**34.75 ± 1.10%**
GAT-POS-T-PI	**63.79 ± 2.35%**	63.73 ± 1.92%	**47.96 ± 3.29%**	42.30 ± 1.50%	34.59 ± 1.01%	**34.78 ± 1.24%**
GAT-POS-T-PT	**65.55 ± 2.38%**	65.42 ± 2.13%	**51.62 ± 1.84%**	50.79 ± 1.35%	**34.97 ± 1.27%**	34.66 ± 1.17%
GAT-POS-C-GS	**64.39 ± 2.16%**	63.18 ± 1.43%	**43.56 ± 3.15%**	42.75 ± 1.85%	**34.78 ± 1.00%**	34.68 ± 0.99%
GAT-POS-C-PI	**67.65 ± 2.82%**	63.14 ± 1.02%	**42.58 ± 1.66%**	42.49 ± 1.68%	34.65 ± 1.21%	**34.91 ± 1.00%**
GAT-POS-C-PT	**67.76 ± 2.54%**	65.75 ± 1.81%	**52.90 ± 1.55%**	50.63 ± 1.29%	34.89 ± 1.38%	**34.95 ± 0.95%**
GAT-POS-CG-GS	**65.92 ± 2.29%**	63.38 ± 1.74%	**48.39 ± 1.61%**	42.68 ± 1.63%	34.89 ± 1.08%	**35.13 ± 0.99%**
GAT-POS-CML-PI	**66.67 ± 2.15%**	63.31 ± 2.11%	**47.44 ± 2.33%**	42.07 ± 1.41%	**34.98 ± 1.04%**	34.87 ± 0.86%
GAT-POS-CML-PT	**67.48 ± 3.08%**	63.33 ± 1.93%	**50.94 ± 2.77%**	42.26 ± 1.42%	**35.03 ± 1.44%**	34.92 ± 1.41%

The winning variants in the comparison are highlighted.

Table 4.8: The Ablation Study of Joint-Training of Auxiliary Embedding Model

joint-training. In the ablation study, we consider a variant of GAT-POS without jointly-trained auxiliary embedding models. Instead, the embedding models will be trained with the unsupervised task in advance to learn positional embeddings. Afterward, the main model will be trained with the supervised task solely, incorporating positional embeddings from the embedding models as extra node attributes. For fairness, the architecture of each version of GAT-POS will remain the same except that the main model and the embedding model are trained separately. The results of the comparison are summarized in Table 4.8.

On Chameleon and Squirrel, most instantiations of GAT-POS with joint-training reach better results than without joint-training overall. However, the joint-training also leads to relatively larger standard deviations.

On Actor datasets, the difference of performances between the instantiations of GAT-POS with joint-training and those without joint-training is not notable. This might be because, on Actor dataset, the positional embedding is probably less affected by the supervised signals.

Therefore, in most of the cases, enabling joint training of the main model and the auxiliary embedding model is beneficial.

Moreover, it is worth noting that the variants of GAT-POS without joint-training still outperform Geom-GCNs by a notable margin, which implies the advantage of having an attention mechanism incorporate semantic and structural information, especially when compared to the weighting function utilized in the filter of Geom-GCNs, which only considers the structural information.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

In this thesis, we have introduced a framework to enhance the GAT models with a positional embedding to better explore both the semantic information of each node and the structural information contained in the graph. In particular, we extended the standard GAT formulation by adding an auxiliary embedding model, connected to the supervised GAT model, and its respective unsupervised objective function to predict the graph context. Although we focused on extending the original GAT formulation, our proposed framework is compatible with most current graph deep learning based models, which leverage graph attentional layers. Moreover, it is worth mentioning that this framework is agnostic to the choice of auxiliary embedding models as well as the particular setup of the graph attentional layers.

Experiments on a number of graph-structured datasets, especially those with more complicated structures and edges joining nodes beyond simple feature similarity, suggest that this framework can effectively enhance GATs by leveraging both graph structure and node attributes for node classification tasks, resulting in increased performances when compared to baselines as well as recently introduced methods.

When comparing different types of positional embeddings, we notice that in datasets where the node attributes are not discriminative enough, such as in the Squirrel dataset, transductive auxiliary embedding models outperform the inductive ones. This indicates that the positional embeddings learned cannot be mapped to the space of the node attributes injectively. Therefore, the parameterization of positional embeddings by means of a function of node attributes leads to a loss of structural information. Even though our proposed framework can assist GATs better balance the structural information and semantic information of the nodes, we have not been able to improve the capacity of GAT-based approach when generalizing to unseen nodes, i.e. the inductive learning settings. The research on improving the generalization capacity is left for future work.

Finally, through an ablation study, we have further emphasized the benefits of our proposed framework by showing that the performance improvements do not only come from the joint training of both parts of the model but rather from endowing GATs with node positional information.

5.2 Future Works

There are a number of directions related to our framework worth exploring as the future works.

The first is about what structural information captured by the positional embeddings is actually needed. The learned positional embeddings might be able to capture more complicated structural information while the fixed and encoded positional embeddings are usually more computationally efficient. In the case of sequences, the learned positional embeddings and the encoded positional encoding do not show significant differences in the performance of Transformer [Vaswani et al., 2017]. It is worth exploiting whether in the cases of more complicated structures, i.e., grids, trees and general graphs, the learned positional embeddings will better benefit the attention mechanism than encoded positional embeddings. One recent fixed and encoded positional embeddings for general graphs is distance encoding [Li et al., 2020b]. It is an interesting direction to comprehensively compare different types of learned positional embeddings and encoded positional embeddings on general graphs.

Second, as mentioned in the previous section, the bottleneck of the capacity to generalize to unobserved nodes lies on the auxiliary embedding models in our frameworks. Currently, most embedding models on graphs are parameterized by means of the functions of node attributes to support the inductive setting. Therefore, the generalization ability of our framework is at most as good as inductive GNNs. Thus, how to improve the generalization ability of the embedding models on graphs in the inductive setting will also be a direction worth investigating.

Another direction that can be investigated is to directly design a framework of adaptive spatial convolutional kernels generated by operations exploiting the structure within each local area (i.e., neighborhood), without positional embeddings. The spatial GNNs with trainable aggregation coefficients, such as MoNets [Monti et al., 2017], MPNNs [Gilmer et al., 2017] and GATs [Velickovic et al., 2018], can be viewed as a generalization of adaptive convolutional kernels [Zamora Esquivel et al., 2019] to the graph domain. Regardless of the other benefits introduced by adaptive convolutional kernels on images, for graphs, at least spatial trainable convolutional kernels can be built upon it. Note that the information captured by the adaptive convolutional kernels highly depends on how the coefficients are generated [1] Currently, the generating functions of the adaptive kernels in the above-mentioned GNNs cannot fully explore the structure within each local area because of treating the nodes in each neighborhood as a multiset. For example, the filter coefficients of GATs (i.e., the attention scores) are generated by the attention mechanism, which views the nodes in each neighborhood as a multiset and ignores the structural dependencies among them. This is the reason why we need to incorporate the extra structural information via positional embeddings in attention mechanisms. Thus, alternatively, we

[1] The concept of generation here is different from the one in generative models; here the generation only denotes the coefficients are produced by a function as the output. We utilize the term *generate* following [Zamora Esquivel et al., 2019].

could eventually design an operation, which is inherently able to explore structural dependencies in each locality, to generate the adaptive kernels. For instance, one could replace the standard content-based attention mechanism in GATs with the structured attention mechanism of [Kim et al., 2017]. The structured attention mechanism describes attention computation as a conditional random field (CRF) and computes the attention scores as marginal probabilities with inference algorithms in probabilistic graphical models. With inference algorithms for CRF, the learned filter coefficients are supposed to capture the structural information in each local area. Furthermore, we can abandon the attention mechanism and alternatively learn filter coefficients with inference algorithms on each local area represented as a CRF. Such operations can be parameterized as a neural network according to previous works [Zheng et al., 2015, Gao et al., 2019], which represent the whole input image or input graph as a CRF. Since we consider each locality as a CRF, exact inference algorithms are feasible in each neighborhood. For instance, the belief propagation is an exact inference algorithm applicable to a local neighborhood represented as a star-graph (i.e., a depth-1 tree). Therefore, the operation of learning filter coefficients is computationally tractable and guaranteed to converge within two steps. Compared to our current framework, this type of adaptive convolutional kernels can be applied to many GNN-based frameworks, by replacing other GNNs, without extra architecture.

Chapter 6

Bibliography

[Acharya and Gill, 1981] Acharya, B. and Gill, M. (1981). On the index of gracefulness of a graph and the gracefulness of two-dimensional square lattice graphs. *Indian J. Math*, 23(81-94):14.

[Alekseyev and Pevzner, 2009] Alekseyev, M. A. and Pevzner, P. A. (2009). Breakpoint graphs and ancestral genome reconstructions. *Genome research*, 19(5):943–957.

[Alexe et al., 2012] Alexe, B., Heess, N., Teh, Y. W., and Ferrari, V. (2012). Searching for objects driven by context. In *Advances in Neural Information Processing Systems*, pages 881–889.

[Atwood and Towsley, 2016] Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pages 1993–2001.

[Ba et al., 2014] Ba, J., Mnih, V., and Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*.

[Belkin and Niyogi, 2002] Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591.

[Bello et al., 2019] Bello, I., Zoph, B., Vaswani, A., Shlens, J., and Le, Q. V. (2019). Attention augmented convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3286–3295.

[Bruna et al., 2014] Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, pages http–openreview.

[Cai et al., 2018] Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637.

[Chen et al., 2020] Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2020). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445.

[Cheng et al., 2016] Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Chollet, 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258.

[Chung and Graham, 1997] Chung, F. R. and Graham, F. C. (1997). *Spectral graph theory*. American Mathematical Soc.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

[Cordonnier et al., 2020] Cordonnier, J.-B., Loukas, A., and Jaggi, M. (2020). On the relationship between self-attention and convolutional layers.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.

[Dauphin et al., 2017] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941.

[Defferrard et al., 2016] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.

[Duvenaud et al., 2015] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232.

[Frasconi et al., 1998] Frasconi, P., Gori, M., and Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786.

[Gao et al., 2019] Gao, H., Pei, J., and Huang, H. (2019). Conditional random field enhanced graph convolutional neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 276–284.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252.

[Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272.

[Gori et al., 2005] Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

[Graves et al., 2014] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

[Gutmann and Hyvärinen, 2012] Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The journal of machine learning research*, 13(1):307–361.

[Hamilton et al., 2017] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Huang et al., 2018a] Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2018a). Music transformer. *arXiv preprint arXiv:1809.04281*.

[Huang et al., 2018b] Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2018b). Music transformer. *arXiv preprint arXiv:1809.04281*.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[Itti et al., 1998] Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259.

[Kim et al., 2017] Kim, Y., Denton, C., Hoang, L., and Rush, A. M. (2017). Structured attention networks. In *ICLR (Poster)*. OpenReview.net.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kipf and Welling, 2017] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net.

[Larochelle and Hinton, 2010] Larochelle, H. and Hinton, G. E. (2010). Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pages 1243–1251.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[Lee et al., 2019a] Lee, J., Lee, I., and Kang, J. (2019a). Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743.

[Lee et al., 2019b] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019b). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753.

[Li et al., 2017] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

[Li et al., 2020a] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. (2020a). A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246.

[Li et al., 2019] Li, P., Chien, I., and Milenkovic, O. (2019). Optimizing generalized pagerank methods for seed-expansion community detection. In *Advances in Neural Information Processing Systems*, pages 11710–11721.

[Li et al., 2020b] Li, P., Wang, Y., Wang, H., and Leskovec, J. (2020b). Distance encoding–design provably more powerful gnns for structural representation learning. *arXiv preprint arXiv:2009.00142*.

[Li et al., 2018] Li, Q., Han, Z., and Wu, X. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, pages 3538–3545. AAAI Press.

[Li et al., 2015] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

[Liao et al., 2019] Liao, R., Zhao, Z., Urtasun, R., and Zemel, R. S. (2019). Lanczosnet: Multi-scale deep graph convo-lutional networks. In *7th International Conference on Learning Representations, ICLR 2019*.

[Lin et al., 2017] Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

[Lu et al., 2016] Lu, J., Yang, J., Batra, D., and Parikh, D. (2016). Hierarchical question-image co-attention for visual question answering. In *Advances in neural information processing systems*, pages 289–297.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

[Maron et al., 2019] Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019). Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167.

[McPherson et al., 2001] McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mnih and Teh, 2012] Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 419–426.

[Monti et al., 2017] Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124.

[Morin and Bengio, 2005] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer.

[Morris et al., 2019] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609.

[Murphy et al., 2019] Murphy, R., Srinivasan, B., Rao, V., and Riberio, B. (2019). Relational pooling for graph representations. In *International Conference on Machine Learning (ICML 2019)*.

[Nickel and Kiela, 2017] Nickel, M. and Kiela, D. (2017). Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347.

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

[Parikh et al., 2016] Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255.

[Paulus et al., 2018] Paulus, R., Xiong, C., and Socher, R. (2018). A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

[Pei et al., 2020] Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020). Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*.

[Perozzi et al., 2014] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

[Pham et al., 2018] Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

[Ribeiro et al., 2017] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Scarselli et al., 2008] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

[Schlichtkrull et al., 2018] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

[Shaw et al., 2018] Shaw, P., Uszkoreit, J., and Vaswani, A. (2018). Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

[Sperduti and Starita, 1997] Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735.

[Srinivasan and Ribeiro, 2020] Srinivasan, B. and Ribeiro, B. (2020). On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*.

[Susnjara et al., 2015] Susnjara, A., Perraudin, N., Kressner, D., and Vandergheynst, P. (2015). Accelerated filtering on graphs using lanczos method. *arXiv preprint arXiv:1509.04537*.

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Tang et al., 2015] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077.

[Tenenbaum et al., 2000] Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Velickovic et al., 2018] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *ICLR (Poster)*. OpenReview.net.

[Von Luxburg, 2007] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

[Wang et al., 2016] Wang, D., Cui, P., and Zhu, W. (2016). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234.

[Wang et al., 2019] Wang, X., Tu, Z., Wang, L., and Shi, S. (2019). Self-attention with structural position representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1403–1409.

[Weisfeiler and Lehman, 1968] Weisfeiler, B. and Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16.

[Weiss, 2000] Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural computation*, 12(1):1–41.

[Weston et al., 2008] Weston, J., Ratle, F., and Collobert, R. (2008). Deep learning via semi-supervised embedding. In *Proceedings of the 25th international conference on Machine learning*, pages 1168–1175.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

[Xu et al., 2019] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

[Xu et al., 2018] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462.

[Yang et al., 2016] Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 40–48. JMLR. org.

[Yedidia et al., 2003] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2003). Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239.

[Yoon et al., 2019] Yoon, K., Liao, R., Xiong, Y., Zhang, L., Fetaya, E., Urtasun, R., Zemel, R., and Pitkow, X. (2019). Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 868–875. IEEE.

[Yu et al., 2019] Yu, Z., Yu, J., Cui, Y., Tao, D., and Tian, Q. (2019). Deep modular co-attention networks for visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6281–6290.

[Yu et al., 2017] Yu, Z., Yu, J., Fan, J., and Tao, D. (2017). Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 1821–1830.

[Zamora Esquivel et al., 2019] Zamora Esquivel, J., Cruz Vargas, A., Lopez Meyer, P., and Tickoo, O. (2019). Adaptive convolutional kernels. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0.

[Zhao et al., 2020] Zhao, H., Jia, J., and Koltun, V. (2020). Exploring self-attention for image recognition. *arXiv preprint arXiv:2004.13621*.

[Zhao and Akoglu, 2020] Zhao, L. and Akoglu, L. (2020). Pairnorm: Tackling oversmoothing in {gnn}s. In *International Conference on Learning Representations*.

[Zheng et al., 2015] Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. (2015). Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537.

[Zhu et al., 2017] Zhu, C., Zhao, Y., Huang, S., Tu, K., and Ma, Y. (2017). Structured attentions for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1291–1300.