ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LPG2vec: Towards learning-enhanced graph databases

Semester paper

Tiancheng Chen

January 1, 2022

Advisors: Dr. M. Besta, Prof. Dr. T. Hoefler

Department of Computer Science, ETH Zürich

Abstract

Graph neural network(GNN), an emerging field in deep learning, is showing great potential in processing structured data. However, most GNNs are designed for homogeneous or heterogeneous graph. In homogeneous graph, vertices and edges belong to same types. And in heterogeneous graph, vertices and edges can hold different types. These graph can be viewed as special cases of Labeled Property Graph (LPG), where vertices and edges can have labels and properties. In this paper, we present LPG2vec, a GNN framework on LPG.

First, we discuss the generation of feature vectors from LPG datasets. We introduce strategies to deal with different types of labels and properties, such as categorical data, numerical data and text-based data. We sort these labels and properties and concatenate features in this order into feature vectors.

Second, we choose node classification as the downstream task to benchamark. We select GAT as the backbone network, and implement two methods to reduce the dimension of feature vectors to increase the training speed and ease the memory consumption pressure without a loss on model accuracy. One is autoencoder, and the other is transfer learning from multi-layer perceptron(MLP) classifiers.

Third, we introduce two ways to modify the attention score function to enable the GNN model to present attention on labels and properties. By extracting the attention parameter, we can go one step further in understanding the mechanism of attention module in GNN networks, and the LPG dataset itself.

Contents

Chapter 1

Introduction

Graph is an important representation of complex data. In common abstraction, graph is composed of nodes and edges that connects these nodes, so that graph can be used to represent not only the information of entities, but the relation between entities as well. This composition enriches the expressivity of graph and leads to broad use of graph structured data. Graph is used to describe many types of data, e.g. citation network[20], knowledge graph[5], biological interaction network[14], point cloud network[26], social network[15](e.g., Reddit, Twitter, Twitch) and so on. The graph analysis tasks mainly focus on discovering the relationship of nodes, such as node classification[3], node clustering[27], link prediction[28] and graph classification[9].

Nowadays, machine learning, especially deep learning[12], becomes the state-of-the-art method on different types of data, such as computer vision[23] on digital images or videos, natural language processing[7] on human language, as well as branches on medical, time series, speech, audio data, etc. Unlike many fields listed above, which requires independent and identically distributed data as the input, nodes in graph are highly related with their neighbours. As a result, direct application of machine learning methods in those fields is inappropriate. So it is natural that the trend comes to deep learning on graphs, graph neural network (GNN)[25], to discover how to incorporate the edge information in representation learning.

In order to lower the computation pressure of graph analytic problems, graph embedding is widely used as the intermediate output of graph analytic pipeline. Graph embedding is the compression of graph data, which preserves both semantic and structural information. Embedding, specifically means a low-dimensional vector of numerical value. Depending on the task, there are three types of embedding focusing on part of the graph (or the whole graph), node embedding, edge embedding and graph embedding. The challenge of different type of embeddings vary. Node embedding should preserve the similarity to neighbouring nodes and nodes with similar neighbouring structure. Edge embedding should preserve the similarity to edges that have similar end points or similar roles in local/global structure. In contrast, graph embedding should incorporate the semantic and structural information of whole graph and subgraphs.

The graph analytic pipeline can be decomposed into two parts, the frontend(encoder) and backend(decoder). The frontend serves as analyzer and encoder, takes graph(s) data as input and generate graph embeddings. The backend serves as decoder, which takes in embedding and output task-required result. The decoupling of two main functionalities make possible separate design of frontend and backend. Currently, the study of graph neural network (GNN) mainly focuses on design of the frontend, and use classic machine learning model as backend. During training, the loss signal is propagated from the backend to GNN layers.

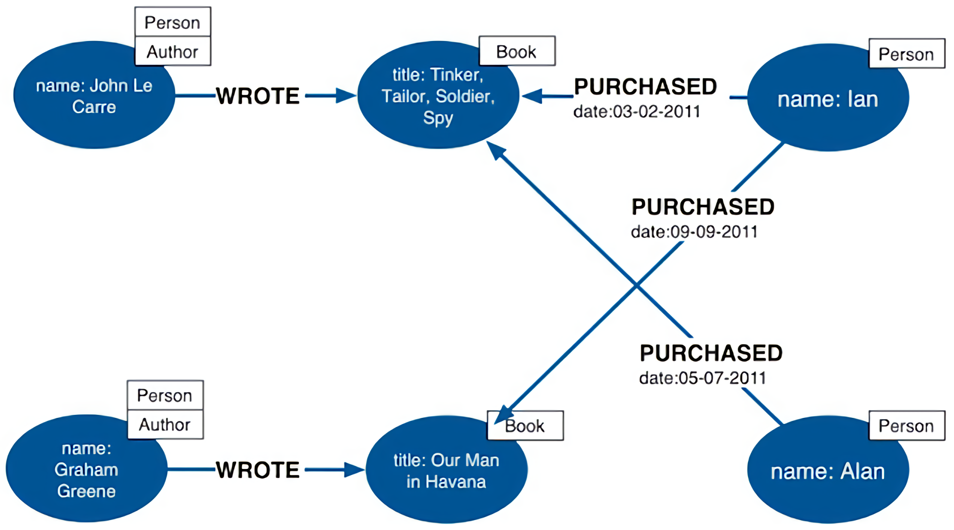


Figure 1.1: An example of LPG data model.

In general, graphs can be categorized into three types in the setting of graph representation learning, homogeneous graph, heterogeneous graph and graph with auxiliary information. The formal definition is provided in the next chapter. Homogeneous graph holds nodes and edges of uni-type. There is a feature vector on each node (sometimes also on each edge) in the same feature space. Usually node and edge do not share the same feature space. Because of the relative simple structure, homogeneous graph embedding is the most well-researched direction in graph embedding. Heterogeneous graph has finite types of nodes and edges. The feature vector of different type of nodes are in different feature space. So the same for edges. The strong expressivity of heterogeneous graph results from the type information, while it also brings new challenges comparing to homogeneous graph. To fuse the information of different types, many heterogeneous graph neural network (HGNN) methods are based on the concept of meta-path. Meta-path is the abstraction of path, which only retains type information. In heterogeneous graph, the local structural information provided by nodes can be quite different if types are taken into consideration, which means type provides rich semantic information. Graph with auxiliary information is the kind of graph that has extra info on node/edge/whole-graph. According to the definition of [4], there are five types of information, label, attribute, node feature, information propagation and knowledge base.

In this paper, a specific type of graph with auxiliary information is considered, that is labeled property graph (LPG). In labeled property graph, nodes and edges can have multiple labels and property pairs. Property pair is defined as key-value pairs. The lack of type information in LPG may cause no two nodes/edges in LPG that have the same labels and property keys, so that nodes can hardly be classified upon labels and property keys, which adds to the difficulty to embed nodes/edges into the same feature space.

Chapter 2

Related works

2.1 Neural network on graphs

Here we list some outstanding work in the GNN field. We also includes pioneer work on embedding of RDF graph, RDF2vec. Our LPG2vec follows the general framework of homogeneous/heterogeneous graph neural networks.

2.1.1 RDF2vec

RDF2vec [19] is a method for node embedding on RDF graphs. It samples random walks to generate visiting sequence of RDF nodes which are used as input of word2vec [17]. In word2vec, each entity and predicate is considered as a word, so the output is embeddings for all entities and predicates.

Though labeled property graph can be transformed into RDF graph, so that RDF2vec is able to generate embeddings for LPG. However, the RDF2vec does not follow the standard GNN framework, but treat random walk on RDF graph as sequences of words and send the sequences into natural language processing (NLP) models, specifically skip-gram model.

2.1.2 GAT

Graph Attention Networks (GAT) [22] generates node embedding on homogeneous graphs. It uses attention mechanism to learn the importance of neighbouring nodes in the message passing phase. Our LPG2vec model uses GAT as backbone network. The details will be discussed in following chapters.

2.1.3 HAN

Heterogeneous Graph Attention Network (HAT) [24] works on heterogeneous graphs. It introduces node-level and semantic-level attentions. Node-level attention learns the importance between a node and its metapath-based neighbors, and semantic-level attention aims to learn the importance of different meta-paths.

2.1.4 MAGNN

Meta-path aggregated graph neural network (MAGNN) [11] is an improved version of HAN. HAN only considered two endpoints of metapath, while MAGNN incorporates all the intermediate nodes in metapath.

2.1.5 GATNE

General Attributed Multiplex HeTerogeneous Network Embedding (GATNE) [6] works on Attributed Multiplex HEterogeneous Network (AMHEN) where nodes are linked with multiple types of edges, and each node has a set of attributes. The authors present two models, transductive model GATNE-T and inductive model GATNE-I. The output of GATNE is embeddings for all nodes for each edge type. The core idea of GATNE is first to aggregate neighbouring embeddings connected with the same type of edge, concatenate the result of all edge types, learn the importance of each edge type using attention, and combine the base embedding (based on graph structure) and importance-vary node embedding to generate node embedding for each edge type.

2.1.6 Simple-HGN

Simple-HGN [16] works on heterogeneous graph. It enforces attention on neighbouring nodes similar to GAT, but taking edge type into consideration. It also unitizes node residual, edge residual and multi-head connection to ensure precision and robustness.

2.1.7 HGT

Heterogeneous graph transformer (HGT) [13] aims to generate embeddings on large heterogeneous graphs using subgraph sampling. The target node embedding is updated from neighbouring nodes with different importance learned through attention. The computation of attention score is similar to that of Transformer [21] rather than GAT.

Chapter 3

Preliminaries

3.1 Definition of several graph data model

3.1.1 Homogeneous graph

Homogeneous graph is a tuple

$$(V, E) \tag{3.1}$$

V is the set of all vertices and E is the set of all (directed) edges. Each edge $e = (u, v)$ represents a link starting at vertex u and ends at vertex v. All vertices belong to a single vertex type and all edges belong to a single edge type.

Notation	Explanation
z_v	vertex embedding
$\mathbf{X}$	vertex feature matrix
$h_{i,k}^{(l)}$	vertex i's hidden embedding at layer l, head k
v_i	vertex i's (compressed) feature vector
e_{ij}	edge (i,j)'s (compressed) feature vector
a	attention vector
$\hat{\alpha}_{ij,k}^{(l)}$	attention score of edge (i,j) at layer l, head k
M	feature mapping matrix
W_{ij}^{r}	restricted mapping matrix
$\vert\vert$	vertex/matrix concatenation

Table 3.1: Notations and explanations.

3.1.2 Heterogeneous graph

Heterogeneous graph is a tuple

$$(V, E, T_V, T_E, f_V, f_E) \tag{3.2}$$

T_V is the set of all vertex types and T_E is the set of all edge types. $f_V : V \mapsto T_V$ is a vertex type mapping function and $f_E : E \mapsto T_E$ is a edge type mapping function. For heterogeneous graph, $|T_V| + |T_E| > 1$. It is guaranteed that vertices/edges of the same type are in the same feature space. Heterogeneous graph can be viewed as a collection of homogeneous graph defined by elements of $f_V(V) \times f_E(E) \times f_V(V)$.

3.1.3 Resource Description Framework (RDF) graph

The core of Resource Description Framework (RDF) is a collection of *triples*. Each triple consists of a *subject*, a *predicate* and an *object* [2]. Subject can be Uniform Resource Identifier(URI) or blank identifier. Predicate can be URI. And object can be URI, bland identifier or literal values. Based on this definition, an RDF graph can be viewed as a set of RDF triples. So in RDF graphs, it is not allowed to identify instances that represents the same relationship. In another word, every subject-predicate-object is unique in RDF graph.

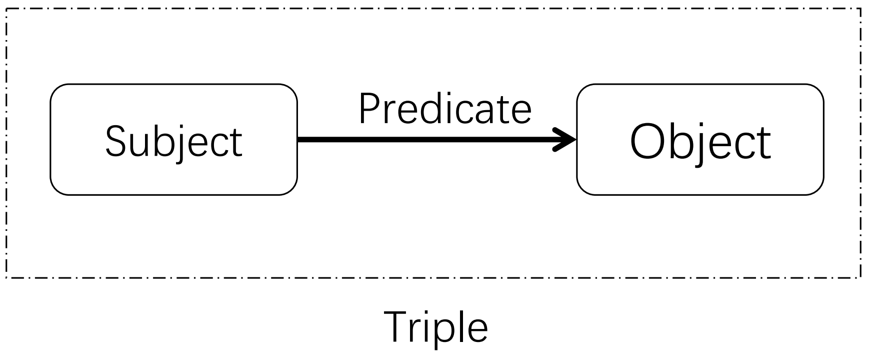


Figure 3.1: RDF triple.

3.1.4 Labeled Property Graph (LPG)

Labeled Property Graph Model (LPG) augments the homogeneous graph model with *labels* that define different classes of vertices and edges and *properties* (some times called *attributes*) in the form of *(key,value)*. *Key* identifies a property and *value* is the corresponding value of this property. LPG is defined as a tuple [2]

$$(V, E, L, l_V, l_E, K, W, p_V, p_E) \tag{3.3}$$

L is the set of labels, $l_V : V \mapsto \mathcal{P}(L)$ and $l_E : E \mapsto \mathcal{P}(L)$ are labeling functions ($\mathcal{P}(L)$ is the power set of L). Thus, each vertex and edge is mapped to a subset of labels. For a property $p = (key, value)$ where $key \in K$ and $value \in W$, $p_V(u)$ denotes the set of property key-value pairs of vertex u, $p_E(e)$ denotes the set of property key-value pairs of edge e.

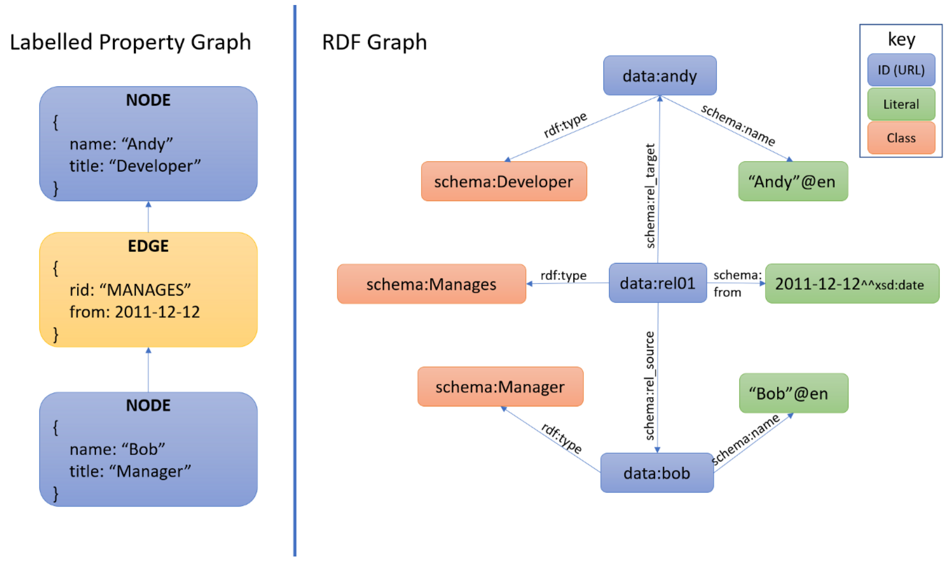


Figure 3.2: A labeled property graph and equivalent RDF graph.

3.2 Graph neural network

Modern GNN models are all based on a concept called message passing. Message passing simply means information as vectors are exchanged between direct neighbouring nodes to update node/edge's own information vector. Here we introduce message passing GNN on homogeneous graphs. For an input graph $G = (V, E)$, the node features are described as $\mathbf{X} \in \mathcal{R}^{|V| \times d_V}$, and edge features $\mathbf{E_{attr}} \in \mathcal{R}^{|E| \times d_E}$. The target of GNN is to generate node embeddings $z_v, \forall v \in V$. During each round of message passing, the hidden embedding $h_v^{(l)}$ corresponding to each node $v \in V$ receives "messages" from neighbouring nodes. The update function can be described as follows:

$$h_i^{(l+1)} = UPDATE^{(l)}(h_i^{(l)}, AGGREGATE_{j \in \mathcal{N}(i)}(MESSAGE(h_i^{(l)}, h_j^{(l)}, e_{ji}))) \tag{3.4}$$

where e_{ji} is the edge feature of edge from node j to node i. The $MESSAGE$ and $UPDATE$ can be any differentiable function and the $AGGREGATION$ is differentiable and permutation invariant function such as sum, mean, max.

In each iteration of message passing, each node forms the message to be sent to neighbouring nodes. Then each node receives messages and aggregate messages into one vector. Finally each node updates its old embedding according to the update function.

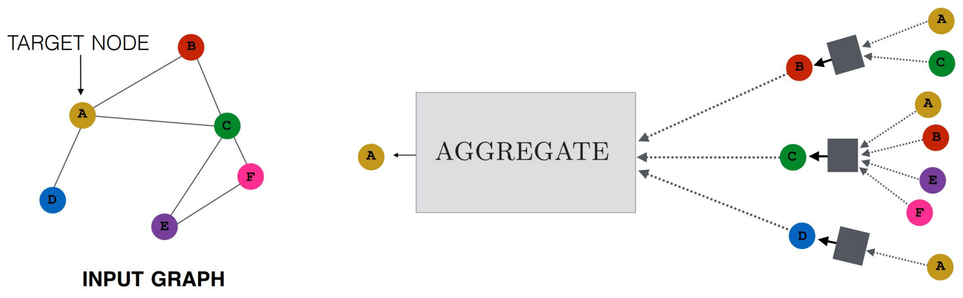


Figure 3.3: Illustration of message passing on graphs.

Chapter 4

Method

To make the LPG2vec a general approach of LPG graph embedding generation, we do not make special assumptions to graph data. Especially, we assume that there is no type info provided, and no clusters of vertices/edges such that vertices/edges in the same cluster have same labels and property keys. Otherwise, this LPG graph can be viewed as heterogeneous graph and can be solved with modern HGNN networks.

So in this chapter, we will first introduce a method to transform LPG graph embedding to homogeneous graph embedding while preserving the information in graph dataset. Secondly, we introduce a method to generate attention scores on labels and properties, so that these attention scores can tell which label/property the GNN is focused on, which may help us understand how GNN learns the information in dataset.

4.1 Pipeline overview

The whole pipeline of LPG2vec is shown in figure 4.3.2. The following sections introduce the essential modules in LPG2vec.

4.2 Feature vector

Noticing that for each LPG graph, the set of labels and property keys are fixed. It's natural to transform LPG to homogeneous graph problem with one-hot style feature vector for vertices and edges. Though for a large database of thousands of distinct labels and property keys, the length of feature vector becomes a problem. But we can use methods discussed in next section to decrease its dimension or we can use attention locator and view LPG2vec in another way as in section 4.5.

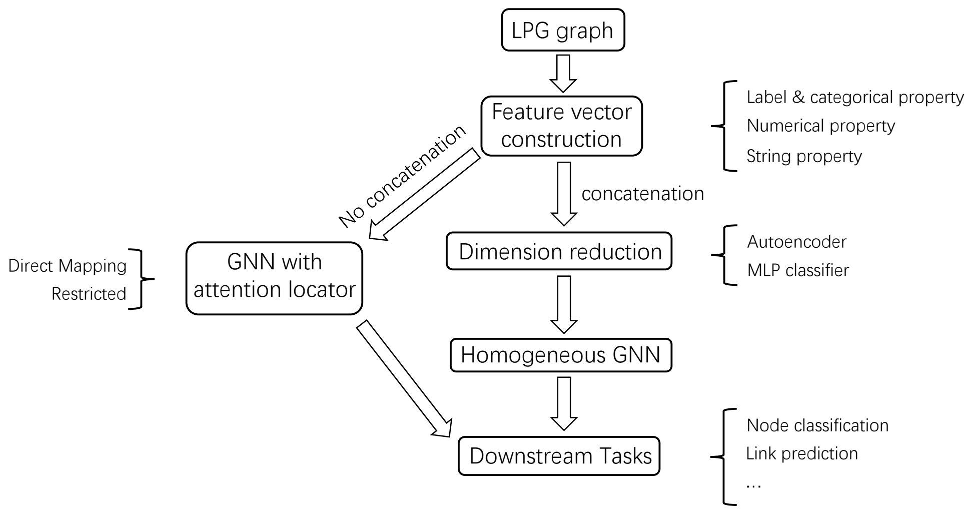


Figure 4.1: The whole pipeline of LPG2vec.

Construction: First, order the elements in Property keys ∪ Labels. This make sure all the vertices and edges have the same ordering in the feature vector.

1. *labels*: Each label need one Boolean entry of $\{0, 1\}$, which means the vertex/edge has the label or not.
2. *properties*:
   a) *finite possible values*: If all the possible values of a key is in a finite set $\mathcal{C}$, then it needs $|\mathcal{C}|$ entries in the feature vector. Each entry is filled with Boolean value as the labels.
   b) *continuous value*: Either normalize the value globally to $[0, 1]$ and fill in as it is, or discretize the value into several ranges and treat it as the case above. Globally means this key of all vertices/edges use the same norm to normalize. The value is discretized into equal-size buckets to avoid the unbalanced distribution of features.
   c) *numerical vector*: Demean and normalize the vector.
   d) *text*: Use swing transformer to transform the string-based properties into numerical embedding. String embeddings are usually much longer than other numerical properties to preserve most information in strings (even if only part of the keywords are necessary for downstream tasks).

Notice that the labels and properties for vertices and edges can be quite different, we can separate the construction of vertex feature vector $x \in \mathcal{R}^n$

and edge feature vector $y \in \mathcal{R}^m$ with different dimensions.

4.3 Dimension reduction

As shown in the experiment chapter, long feature vectors might lead to slow convergence of GNN model and oscillation in loss. Also, storing large feature matrix is costly and can be seen as repetition of graph database.

On the other hand, modern GNN models are designed to treat short feature vectors, as the most frequently used GNN datasets are small in feature vector dimension, represented by few channels in hidden dimension. So those models can hardly extract feature info from long feature vectors. The experiment comparing GAT to MLP classifier also shows this point. To increase the number of hidden layer channels could mitigate this problem, but not effectively solve it as feature vector might be of thousands of channels if most properties are texts rather than numerical values.

4.3.1 Approach 1: autoencoder

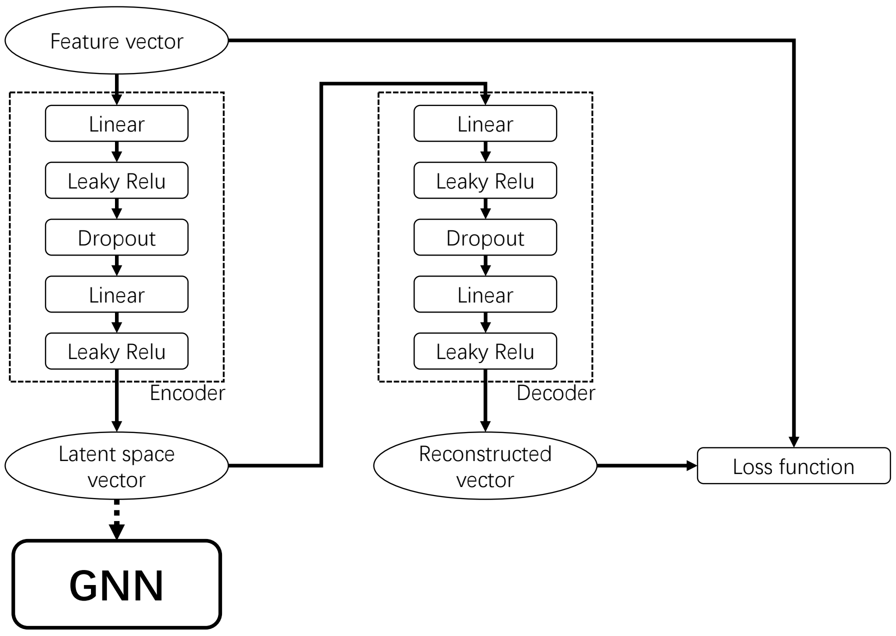


Figure 4.2: An overview of autoencoder model.

Autoencoder network has been vastly used as a differentiable neural network approach to reduce feature dimension. It is composed of two parts, encoder network adn decoder networ. Usually the encoder and decoder networks are MLPs. MSE loss function is the common choice for reconstruction loss as this is a regression problem. E,g, for vertices, $loss_{MSE} = \sum_i ||x_i - decoder(encoder(x_i))||_2$. And the obtained node encoding $v_i \in \mathcal{R}^p$, edge encoding $e_{ij} \in \mathcal{R}^q$,

$$v_i = encoder_v(x_i) \tag{4.1}$$

$$e_{ij} = encoder_e(y_{ij}) \tag{4.2}$$

A possible autoencoder architecture is as follows. $Linear(a, b)$ denotes a linear layer with input dimension a and output dimension b.

Encoder: Linear(input_dim, hidden_dim) → Leaky Relu activation → DropOut → Linear(hidden_dim,output_dim) → Leaky Relu activation.

One proper choice of the hidden layer dimension is

$$\sqrt{dim(input\_dim) \cdot dim(output\_dim)} \tag{4.3}$$

Decoder: symmetrical to encoder.

The autoencoder can be generalized to

$$x \xrightarrow{encoder} z \xrightarrow{decoder} x' \tag{4.4}$$

where z is the latent space feature, and x' is the reconstruction of x. The latent feature vector z is passed to GAT model as input.

4.3.2 Approach 2: transfer learning from MLP classifier

This approach is inspired by transfer learning in computer vision. If we first train a neural network classifier with vertex feature vectors as input. After achieving desired accuracy on this classifier, we can use part of this neural network to encode feature vector, as long as the output dimension of the partial model is as low as expected. This approach solves a problem of using modern GNN models on LPG graphs: no powerful decoding network to decode complex feature vectors.

We still choose a MLP network as the classifier. Many other state of the art neural network structures can be used as classifier, such as transformer. but in this project, we mainly focus on the feasibility of the whole pipeline rather than the extreme model accuracy.

MLP structure: Linear(input_dim, compressed_dim) → DropOut → Relu activation → Linear(compressed_dim,compressed_dim) → DropOut → Relu activation → Linear(compressed_dim,output_dim).

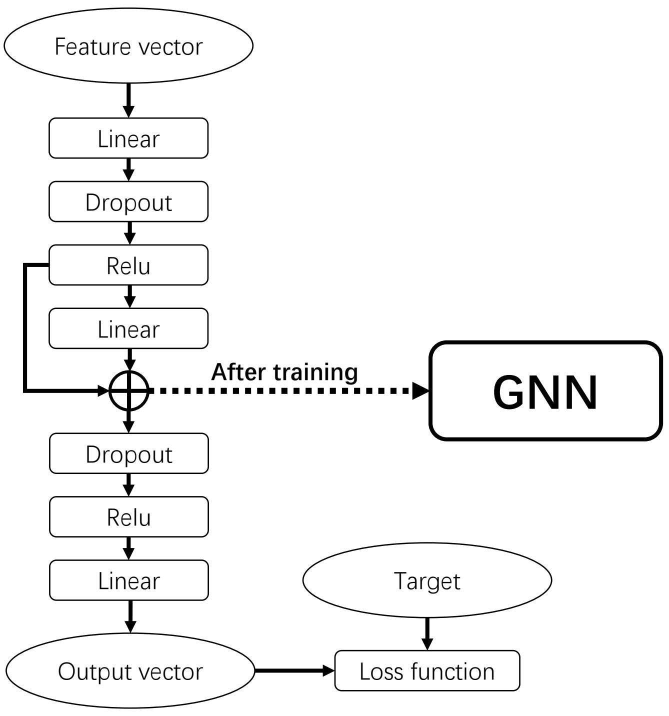


Figure 4.3: An overview of MLP classifier.

Residual connection can be added to the linear layer of same input and output dimension."compressed_dim" means the dimension of compressed feature vector. After this classifier is properly trained, the output of second linear layer is defined as the compressed feature vector. Because the output of first linear layer is only linear transformation of initial feature vector, and the output of last linear layer only contains the task specific info.

One drawback of this approach is that for every single classification task, we have to train a classifier in advance, which is not feasible when multiple tasks exist. In this case, we can design a network that can be separated into two parts, general info extraction section and task-specific section. We fix the structure of general info extraction section, and concatenate it with different task-specific section and perform multi-task training. The output of general info extraction section can be seen as compressed feature vector.

4.4 Homogeneous GNN

We use GAT[22] as the backbone GNN structure for this project, because it first introduce attention mechanism to GNN and is proved to be highly effective. In this review paper[16], the authors also found rich potential of

GAT on heterogeneous graphs. Also, we can modify the GAT to add our attention locator to it in a clean way.

4.4.1 GAT

The original GAT is a homogeneous GNN targeting vertex embedding. For each vertex in the graph, GAT learns an attention score for all the direct neighbouring vertices. Then vertex embedding is updated according to its old embedding and embedding of neighbouring vertices weighted by attention score.

Attention score

a is a learnable weight vector. $h_{i,k}^{(l)}$ is hidden embedding of vertex i, head k at layer l. $W^{(l)}$ is a learnable linear transformation on vertex embedding. $\hat{\alpha}_{ij,k}^{(l)}$ is the attention score of vertex j to vertex i at layer l and head k.

$$\hat{\alpha}_{ij,k}^{(l)} = \frac{exp(LeakyReLU(a^T[W^{(l)}h_{i,k}^{(l-1)}||W^{(l)}h_{j,k}^{(l-1)}]))}{\sum_{n\in\mathcal{N}_i} exp(LeakyReLU(a^T[W^{(l)}h_{i,k}^{(l-1)}||W^{(l)}h_{n,k}^{(l-1)}]))} \tag{4.5}$$

With a simple modification, we can add edge feature vectors to the attention score equation. Now the a' is a learnable weight vector, in which the last entries represent attention on edge feature vector. $W'^{(l-1)}$ is linear transformation on edge features.

$$\hat{\alpha'}_{ij,k}^{(l)} = \frac{exp(LeakyReLU(a'^T[W^{(l)}h_{i,k}^{(l-1)}||W^{(l)}h_{j,k}^{(l-1)}||W'^{(l)}e_{ij}))}{\sum_{n\in\mathcal{N}_i} exp(LeakyReLU(a'^T[W^{(l)}h_{i,k}^{(l-1)}||W^{(l)}h_{n,k}^{(l)}||W'^{(l-1)}e_{in}]))} \tag{4.6}$$

Update function

The attention score determines how influential the neighbouring vertex is to the central vertex. K-head attention is applied to add to robustness and expressivity of the GAT model.

$$\hat{h}_{i,k}^{(l)} = \sum_{j\in\mathcal{N}_i\cup\{i\}} \hat{\alpha}_{ij,k}^{(l)} W^{(l)} \hat{h}_{j,k}^{(l-1)} \tag{4.7}$$

Then apply K-head concatenation:

$$h_i^{(l)} = (||_{k=1}^{K} \hat{h}_{i,k}^{(l)}) \tag{4.8}$$

For last layer, simply take the average of k-head embeddings instead.

$$z_i = \frac{1}{|K|}\sum_{k=1}^{K} \hat{h}_{i,k}^{(l)} \tag{4.9}$$

The initialization of embedding is as follows.

$$h_i^{(0)} = (||_{k=1}^{K} v_i) \tag{4.10}$$

4.4.2 Attention locator

A question about GNN is where does GNN focus on to ensure high accuracy. Take CNN as an analogy, after training the network, we can visualize the coefficients of 2D convolution kernels to understand how CNN grasp information in pictures.

For general GNN, we can examine its focus point by extracting parameters of the model. In GAT, the attention score on each edge is computed by eqn 4.5. The value of attention score determines the importance of neighbouring vertex to the central vertex. So it is necessary to investigate the attention score equation to understand the attention mechanism. In eqn 4.5, the importance of each edge is computed by $a^T[Wh_i||Wh_j]$ (some subscripts are removed for simplicity), where both a^T and W are learnable parameters, h_i and h_j is vertex embedding of last layer. Then if only the attention on entries of vertex feature vector is needed, we can extract the a and W of first GNN convolution layer, $a^T[W||W]$ is the concatenation of attention on entries of source/target vertex.

However, attention on feature vector entries is not enough to understand GNN behavior. The entries of feature vector does not necessarily have independent meanings. For example, string embeddings are usually high dimensional vectors, the embedding is only meaningful when comparing different strings by computing similarity of strings through similarity function on string embeddings.

For LPG graphs, as feature vectors are constructed by simply concatenating labels and processed properties in section 4.2, it provides some possibility to achieve attention on labels and properties. Here, two different approaches on uncompressed feature vectors are given, and the experiments on accuracy and attention score evolution are discussed in following chapters. Then methods on compressed feature vectors will be introduced.

Feature mapping

Feature mapping is the core of attention on labels and properties for both approaches. The target is generate one-to-one mapping of label/property and entries in the feature vectors.

If a feature (label/property) takes entries from index *start* to *end* in original feature vector. Then construct a mapping vector of the same length

$$m := \begin{cases} 1/\sqrt{k} & \text{if entry i} \in \text{[start, end]} \\ 0 & \text{else.} \end{cases} \tag{4.11}$$

The mapping matrix is the stacking of mapping vector of all labels and properties, $M = [m_1, m_2, ...]^T$. The coefficient $1/\sqrt{k}$ is to ensure the L2 norm of each mapping vector is 1.

For example, if vertices of a LPG graph has two labels and one property of three entries, then the mapping matrix M is defined as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{pmatrix} \tag{4.12}$$

Approach 1: Direct mapping attention score function

In the eqn 4.5, we substitute the learnable matrix W with the feature mapping matrix M. So the attention score function becomes

$$\alpha_{ij} = \frac{exp(LeakyReLU([\frac{a_{src}}{||a_{src}||_2} \ \frac{a_{dst}}{||a_{dst}||_2}]^T [Mv_i||Mv_j]))}{\sum_{n \in \mathcal{N}_i} exp(LeakyReLU([\frac{a_{src}}{||a_{src}||_2} \ \frac{a_{dst}}{||a_{dst}||_2}]^T [Mv_i||Mv_n]))} \tag{4.13}$$

Here we notice the attention vector a can be decomposed into attention on source and target node, $a := [a_{src} \ a_{dst}]$, and we normalize the a_{src} and a_{dst} because the attention vector tends to shrink in magnitude during training. The normalization is performed only for evaluating attention evolution. When testing, the additional normalization has no effect on the test accuracy.

Now the learnable vector a is of length $2 \times$ (number of labels& properties), representing attention score on source/target labels and properties. This modified graph convolution layer replaces the first graph convolution layer in GAT network to enable attention on labels/properties.

For the update function, we maintain another learnable matrix as in equation 4.7.

$$h_i = \sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} W v_j \tag{4.14}$$

The problem of this method is that the multiplication of mapping matrix M and feature vector can be seen as averaging on entries of each label/property. This will sacrifice expressivity of the model, as the learnable linear transformation W is replaced by determinate matrix M. Also average on entries of label/property does not make sense for high dimensional data. In order to solve this problem, the restricted attention score function is introduced.

Approach 2: Restricted attention score function

We introduce a method to restrict the learnable matrix W in the attention score function 4.5. The new W^r is of the same shape as mapping matrix M and elements of W^r is defined as

$$W^r_{ij} = \begin{cases} w^r_{ij} & \text{(learnable) if } M_{ij} \neq 0 \\ 0 & \text{(fixed) else.} \end{cases} \tag{4.15}$$

To make the restricted matrix W^r mimic the behavior of the direct mapping matrix M, we put constraints on elements of W^r: 1. $w^r_{ij} \in (0,1), \forall w^r_{ij}$. 2. each row of W are normalized. Practically, this is done by:

$$W^r_i \leftarrow \frac{sigmoid(W^r_i)}{||sigmoid(W^r_i)||_2}, \forall i \in [0, \text{num\_features}) \tag{4.16}$$

where W^r_i is the i-th row of W^r.

For example, the restricted matrix W^r corresponding to mapping matrix 4.12 is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & w^r_{3,3} & w^r_{3,4} & w^r_{3,5} \end{pmatrix} \tag{4.17}$$

where $(w^r_{3,3})^2 + (w^r_{3,4})^2 + (w^r_{3,5})^2 == 1$. Clearly, for properties that occupies multiple entries, the restricted W^r can preform a linear transformation on it, where the direct mapping M only performs averaging.

The attention score function is

$$\alpha_{ij} = \frac{exp(LeakyReLU([\frac{a_{src}}{||a_{src}||_2} \frac{a_{dst}}{||a_{dst}||_2}]^T[W^r v_i||W^r v_j]))}{\sum_{n \in \mathcal{N}_i} exp(LeakyReLU([\frac{a_{src}}{||a_{src}||_2} \frac{a_{dst}}{||a_{dst}||_2}]^T[W^r v_i||W^r v_n]))} \tag{4.18}$$

This approach can be seen as an improvement of approach 1, where we perform linear transformation to each label and property separately, rather than viewing the feature vector as a whole and perform linear transform using fully learnable matrix W.

The update function is the same as equation 4.14.

4.4.3 Downstream tasks

Once vertex embeddings are obtained from GNN, they can be put into downstream tasks such as vertex classification, link prediction, etc.. In our project, to simplify the assessment of our methods on limited datasets, we choose vertex classification as downstream task. Denote C as number of classes,

$x \in \mathcal{R}^{|C|}$ as vertex embedding, $y \in \{0, 1, ..., C-1\}$ as target. Then the NLL loss function is computed as

$$loss(x, y) = -log(softmax(x[y])) \tag{4.19}$$

where x[i] is the i-th element of x.

4.5 Another view of LPG2vec: with attention locator

For large graph database, millions of nodes and labels/properties, it is sometimes impossible to iterate through the whole database and construct the whole feature vector by concatenation as done in section 4.2. Also for large database, the feature vector will be extremely sparse. Is it possible to avoid the concatenation procedure, so that each vertex can generate embedding only based on the labels and properties it possesses? Actually with GAT backbone, and the attention locator we defined, it is possible. Take the restricted matrix approach as example.

Suppose the graph database has m unique labels and n unique property keys. Denote l_i as the i-th label, and p_j as the j-th property. l_i takes binary value(0 or 1), p_j can be arbitrary length numerical vector within range $[0, 1]$.

In the restricted attention score function 4.18, the restricted matrix W_r is block-diagonal, and the same shape as feature mapping matrix M, so the multiplication $W_r v_i$ can be interpreted as stacking of linear transformation of labels and properties:

$$v \mapsto W^r v \Leftrightarrow [l_1 \,...\, l_m \quad p_1 \,...\, p_n] \mapsto [\theta_{l_1} l_1 \,...\, \theta_{l_m} l_m \quad \theta_{p_1} p_1 \,...\, \theta_{p_n} p_n] \in \mathcal{R}^{m+n} \tag{4.20}$$

where θ_l and θ_p transform label and property to scalar value. Remember in the feature vector construction, for those labels and properties that a vertex does not hold, the value in corresponding entry is simply zero, meaning the θ transformation result in zero. In brief, only the labels and properties one vertex holds affects the calculation of $a^T[W^r v_i || W^r v_j]$.

On the other hand, the matrix W in update function 4.14 can be sliced in columns as $[W_{l_1} || ... || [W_{l_m} || W_{p_1} || ... || W_{p_n}]$. Same as the discussion above, missing labels or properties do not contribute to the update function.

We have shown that the construction of sparse feature vector can be avoided as long as global parameters such as W^r, a and W are visible to each vertex and the start and end position of each label/property in the "virtual" feature vector.

Chapter 5

Implementation

5.1 Datasets

5.1.1 Datasets from applications

The LPG datasets are downloaded from Neo4j[18] github repo (https://github.com/neo4j-graph-examples). The problem is that vertices only have one or two labels and edges only have one label. So these datasets can be trivially transformed to heterogeneous datasets. However, in this project, these datasets will be viewed as LPG graphs as the main goal of this project is to investigate how GNN learn from labels and properties.

In this project, we focus on node classification task, so the selection of prediction target is as follows: We select a property which is shared by most vertices and is either categorical or numerical. For numerical property, we discretize the range into several bins and try to balance the number of vertices that are in different bins, which can make the classes less biased. The default number of classes is $10 + 1 = 11$ consisting of 1 class for blank entry and 10 classes for values.

The datasets used in this project:

1. **"recommendation"**
   This dataset contains information of movies and ratings from users.
   Labels: Director, Actor, Genre, Movie, Person, User.
   Properties: Budget, Countries, ImdbVotes, Name, Plot, Revenue, Runtime, Title, Year.
   Classification target: ImdbRating.
   Edge features: Type, Rating, Role, Timestamp.
   This dataset is the only two genuine one that some vertices has more than one labels (The other one is "fraud-detection". On the other hand, the rest two datasets can be degenerated to heterogeneous datasets.

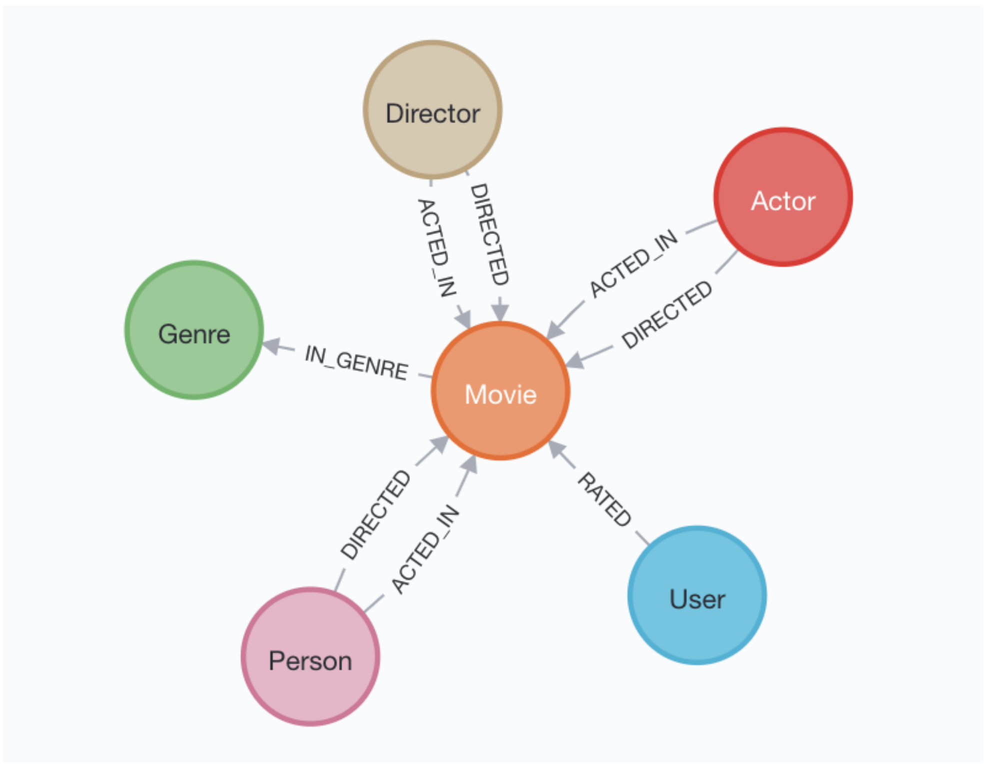


Figure 5.1: An abstraction of ”recommendation” dataset model.

2. **”citations”**
   This dataset contains information of papers, authors and citations.
   Labels: Article, Author.
   Properties: Abstract, Title, Year.
   Classification target: citations.
   Edge features: Type.
   The author names are not selected as a feature because one type of node is author which has links to paper vertices, so the authors are uniquely identified in this dataset. Also, encoding names with string transformer could not bring extra information because the similarity of two names is not important in citation networks.

3. **”fraud-detection”**
   This dataset contains information of fraud transactions among bank accounts.
   Labels: Merchant, Bank, Transaction, Client, SSN, Email, Phone.
   Properties: Name.
   Classification target: Amount.
   Edge features: Type.
   There are several properties not considered, such as Email, Phone, SSN.

Dataset	Vertices			Edges		
	Num	Label Dim	Property Dim	Num	Label Dim	Property Dim
"recommendation"	28863	6	1343	166261	5	391
"citations"	132259	3	779	221237	3	0
"fraud-detection"	326310	13	384	708301	6	0
"fincen"	7524	3	1174	40835	5	0

Table 5.1: An overview of real datasets from Neo4j repo.

These properties do not provide more information than names. There is one Boolean property named "Fraud", this property is not used as classification target because True/False rate is heavily biased for classification, leading to high accuracy (over 99%) and poor precision.

4. **"fincen"**
   This dataset contains information of movies and ratings from users.
   Labels: Filling, Entity, Country.
   Properties: Begin_date, Beneficiary_bank, End_date, Name, Originator_bank.
   Classification target: Amount.
   Edge features: Type.

5.1.2 Artificial Dataset

As real public dataset that meets LPG standard is scarce, we choose to generate artificial datasets. The generation of graph data is tricky, as vertex labels/properties, edges connecting vertices and edge labels/properties have to be generated simultaneously.

[8] discussed generation of graph data aiming at node classification. It uses Stochastic Block Model[1] (SBM) to generate edge connectivity. A SBM assigns each vertex to communities as follows: vertices are directly connected with probability p if in same community, probability q if in different community. As for vertex features, this paper trivially use random number generator.

In PyTorch Geometric[10], there are also generators for homogeneous (FakeDataset) and heterogeneous (FakeHeteroDataset) graphs. Both generators use random generation for vertex features and edge connectivity. FakeHeteroDataset first generates vertex types and then for each edge type (defined by the type of start and end vertex) generates edge connectivity separately.

In this project, class ArtificialDataset can be used to generate LPG graph data, which follows the above two approaches. User can define number of labels and properties for vertices/edges, and probability that each vertex/edge holds that label/property. The number of properties is equivalent to property dimension as we set the dimension of single property to 1. For randomly generated features, the above setting is clear.

We tested the classification task of artificial dataset on GAT network, and the test accuracy approaches to $1/num\_classes$, which is equivalent to a random classifier. We tried to fuse the classification target information into the graph dataset to improve the result as the Pytorch Geometric package does: add the class number to each entry of feature vector. Additionally, we add the sum/difference of class numbers of start and end vertex of each edge to every entry of its edge feature. However, from experiment results, it seems models can hardly generalize from these info fusing.

5.1.3 SBM dataset

In order to increase the classification accuracy on artificial dataset, we follow the path in [8], by using stochastic block model to define edge connectivity. We modified the class "StochasticBlockModelDataset" in PyTorch Geometric. The vertex feature is generated by the function "make_classification" from sklearn. The graph is separated into blocks and each block contains several clusters. All vertices in the same block are assigned with a unique class. Then a hypercube whose vertices become the centroids of clusters is constructed. Vertex feature is sampled from a standard normal distribution with a shift to the centroid of the cluster it belongs to. Additionally, to increase the difficulty of classification, vertex feature is augmented by linear combination or repetition of existing feature, and random noise. A user defined probability "flip_y" controls the proportion of classification target that is replaced with random class label.

There is no longer clear separation of labels and properties in vertex feature, but we can view it as a special case where each label/property occupies one entry in the feature vector.

5.2 Code framework

The methods mentioned in the Method chapter are all implemented with Pytorch Geometric(PyG) in the GitHub repo https://github.com/tiachen-ctc/LPG2vec. Though we mentioned the construction of feature vector is not necessary for our method in section 4.5, we choose to construct it as the testing datasets are of moderate size and this way is more compatible to the PyG infrastructure.

Chapter 6

Experiments

6.1 What contributes to classification accuracy

We investigated the contribution of additional data to the improvement in test accuracy. This part of analysis is based on original GAT, which has 2 GATconv layers, dropout layers and elu activation. Each GATconv layer has 8 heads and 16 hidden channels.

Four cases are examined: 1. Only structural information. The vertex features are generated from uniform distribution on $[0, 1)$ of dimension 50 to represent ″no″ vertex feature info. 2. Structural info and vertex labels. 3. Structural info and full vertex info (including vertex labels and vertex properties). 4. Structural info, vertex info and edge info.

Also, in order to investigate the performance of neural network without structural info, we designed a MLP network, which has 2 hidden layers of dimension $\sqrt{in\_dim * out\_dim}$ with dropout layer and elu activation between every linear layers. A residual connection is applied between the hidden layers.

Dataset	Test Accuracy ± std.		
	blind	v_l	v_l&v_p
″recommendation″	0.688±0.003	0.698±0.019	0.761±0.005
″citations″	0.718±0.001	0.717±0.002	0.728±0.002
″fraud-detection″	0.099±0.001	0.234±0.005	0.258±0.001
″fincen″	0.402±0.008	0.434±0.027	0.588±0.01
artificial	0.102±0.006	0.104±0.01	0.091±0.032

Table 6.1: Test accuracy of MLP network on different amount of provided info. blind: no info. v_l: vertex labels. v_p: vertex properties.

Dataset	Test Accuracy ± std.			
	structural	v_l	v_l&v_p	v_l&v_p&e
"recommendation"	0.705±0.009	0.713±0.012	0.754±0.003	0.757±0.003
"citations"	0.718±0.001	0.716±0.002	0.725±0.002	0.726±0.002
"fraud-detection"	0.1±0.001	0.241±0.001	0.239±0.002	0.212±0.012
"fincen"	0.415±0.008	0.395±0.011	0.555±0.009	0.553±0.005
artificial	0.101±0.002	0.1±0.002	0.1±0.002	0.099±0.002

Table 6.2: Test accuracy of GAT network on different amount of provided info. structural: only connectivity info. v_l: vertex labels. v_p: vertex properties. e: edge labels and properties.

The datasets are randomly split into train set, validation set and test with ratio 50:20:30. An early stopping mechanism is applied to the end of each epoch of training. Early stopping checks if validation accuracy is not decreasing for a consecutive time (in this experiment, 50). In this case, training stops to avoid overfitting. Then the model with highest validation accuracy is restored.

Analysis:

1. **The baseline accuracy for some datasets is high.**
   For example, in the "recommendation" dataset, test accuracy of MLP with only randomly generated feature is around 68.8%. As the neural network is not feed with informative data, this can be viewed as "blind" guess of neural network. The reason is because the way we construct vertex feature vector. There are entries for each label or properties in the feature vector even if the vertex does not have this property. And also the vertex classification task is performed on all the vertices. This means if the selected classification target is only shared for a small amount of vertices, then a blind guess of class "Not a number" will achieve rather high accuracy. In the "recommendation" dataset, the classification target is "ImdbRating", which is a property only possessed by "movie" vertices. Only about 31.6% of vertices have this property, so the accuracy for blind guess is around 70%. On the other hand, the test accuracy of MLP with no info can be seen as a baseline for other results.

2. **The improvement from additional information varies among datasets.**
   The datasets are not created or collected for training neural networks. Even though the selected classification target is chosen as the most informative label/property, the impact of additional information on datasets shows great difference.
   1. "recommendation": The MLP model with full vertex info reaches top test accuracy of 76.1%, which outperforms the result from GAT. It

Model	Test Accuracy ± std.				
	Feature Dimension				
	50	100	200	400	800
GAT	0.901±0.002	0.896±0.011	0.724±0.062	0.426±0.156	0.128±0.036
MLP	0.388±0.011	0.394±0.005	0.336±0.01	0.291±0.005	0.262±0.011

Table 6.3: Test accuracy of GAT and MLP model on SBM dataset with different feature dimension. Both trained for 400 epochs.

seems GAT utilizes the structural info and gains 1.7% more accuracy over blind guess of MLP. With extra info from node and edges, the accuracy of GAT gradually improve but slightly lower than MLP's. This is probably because the original GAT structure only designed for feature vector of low dimension. In the update function 4.7, only a linear transformation is applied to feature vector, which makes GAT hard to extract useful info from high dimensional data.
2. "citations" and "fincen": The GAT and MLP model accuracy benefits mostly from the vertex property info for an amount of 0.9% and 14.0%. GAT still has difficulty to extract info from high dimensional input data. The best performance of GAT is in the case of full vertex info, where GAT is still 0.3% lower than MLP.
3. "fraud-detection": Both MLP and GAT model degenerate to random classifier when no vertex and edge info are provided. This may indicate that the classification target we selected is independent on the edge connectivity. In GAT, the accuracy with additional vertex label information improves 14.1% comparing to data with only structural info.
4. artifical dataset: Neither GAT nor MLP generalize from random data, even if the classification target is fused to vertex and edge features. A better artificial dataset that inherits classification target in both structural and vertex/edge info may provide better result.

6.2 SBM dataset: a better artificial dataset

The tested SBM dataset contains 10000 vertices, 10 classes (1000 vertices/-class), and 50 dimensional vertex feature vector. The edge probability inside each block and between blocks is 0.04 and 0.001. The "flip_y" equals 0.1.

We first tested the SBM dataset with original GAT, only exposing structural info, which means the vertex feature is sampled from uniform distribution on $[0, 1)$. The models are trained up to 400 epochs. However, the performance is as poor as the result from artificial dataset of last section, where MLP and GAT only result in a random classifier. If class number is added to each entry of feature vector, GAT is still a random classifier while MLP

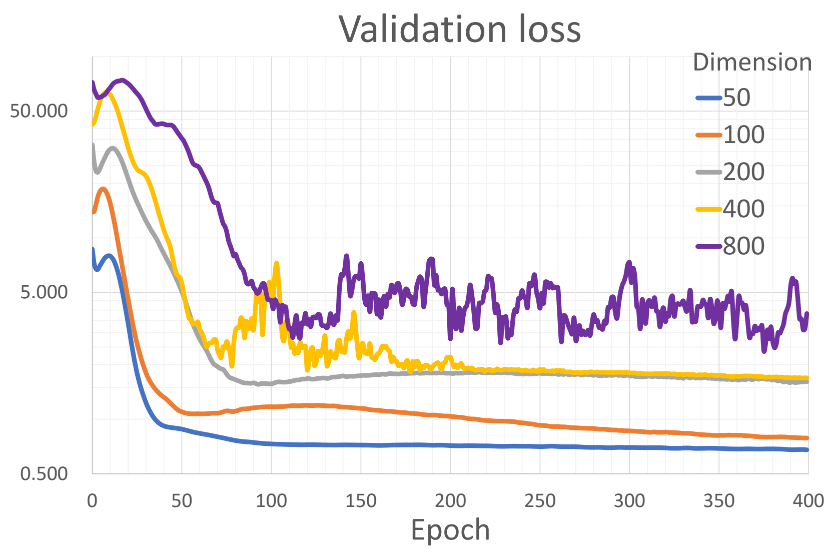


Figure 6.1: Validation loss of GAT model on SBM dataset with vertex feature dimension 20, 40, 60, 80 and 100.

gains accuracy about 20% to 30%.

Then we tested SBM dataset with full vertex info. The dimension of feature vector is set to 50,100,200,400 and 800 to simulate real LPG vertex feature vector. When the size of feature vector is low, the test accuracy of GAT leaps to around 90%, which is about the best possible score, as the "flip_y" is set to 0.1, meaning 10% of test labels are intentionally set wrongly. The result implies GAT can extract info from both vertex features and edge connectivity. It is also shown in the test accuracy of MLP model, which is around 40%, less than half of that of GAT.

However, the test accuracy of GAT decreases when the dimension of feature vector is increasing. As we only tested the model with 400 epochs, models with high dimension feature vector have not yet converged, which can be proved by the validation loss figure 6.1. The validation loss oscillates in the first epoch and climb to peak then decrease slowly. With higher dimension feature vector, the peaking point in terms of epochs tend to come later.

This can be explained by the update function and attention score calculation of GAT. GAT simply use a matrix that multiplies to vertex embedding of last step to map the feature vector from high dimension space to embedding dimension space. When the dimension of feature vector is comparable to the hidden dimension of GAT (in another word, the embedding dimension), the linear transform can be trained to extract feature info within small number of epochs. Unlike the GAT model, the simple MLP model focus entirely on the info extraction. The two hidden layers and activation layers add to the expressivity of MLP model. Also, the residual connection and dropout layers enhance the robustness of MLP model and prevents overfitting.

For real LPG datasets introduced in the last section, the dimension of feature vector is much greater than 100, where the embedding of text generated by string transformer occupies most entries in the feature vector. Even though increasing both hidden embedding dimension and number of training epochs may mitigate the high validation loss, this approach does not scale with high dimension input. To apply GAT-like GNN to LPG dataset, we need to enhance the info extraction capability of GAT.

6.3 Using autoencoder to reduce input dimension

Dataset	Test accuracy ± std.			
	uncompressed	compressed dimension		
		20	40	60
"SBM"	0.385±0.126	0.144±0.022	0.122±0.011	0.14±0.017
"recommendation"	0.75±0.005	0.731±0.002	0.738±0.005	0.74±0.003
"citations"	0.724±0.002	0.716±0.001	0.717±0.004	0.719±0.002
"fraud-detection"	0.246±0.002	0.24±0.003	0.241±0.003	0.24±0.003
"fincen"	0.544±0.011	0.4±0.011	0.411±0.013	0.453±0.02

Table 6.4: Test accuracy of GAT model on several datasets with autoencoder to decrease the feature vector dimension to 20, 40 and 60. The SBM dataset uncompressed feature dimension is set to 400.

A straight forward idea to scale the GNN to high dimensional input and enhance the info extraction capability is to use autoencoder to reduce the dimension of GNN input, while keeping the information within the feature vector intact. The whole model is separated into two parts. First part is an autoencoder. It is composed of two MLP network with symmetrical structure but independent weights, named encoder and decoder. The encoder encodes the input and transform it into low dimensional vector. Then the decoder transform it back to original space. Second part is the GAT model. GAT uses the compressed feature vector as input and performs the vertex classification task.

The encoder used in following experiment is two linear layers combined with dropout layer and Leaky Relu activation. The autoencoder is trained with MSE loss for 50 epochs. The SBM dataset feature vector dimension is set to 400. The input feature vectors are compressed to dimension of 20, 40, and 60. The GAT model uses the same hyperparameters as previous experiments. The SBM dataset uses the same generation parameters as above but output 100 dimensional feature vector. The GAT model is trained with NLL loss for 400 epochs. Need to mention that both the autoencoder and GAT use the same training, validation, test separation to avoid test data leak, so that the autoencoder will have no clue about how to compress the feature vectors in validation and test set.

Analysis:

1. ”SBM” dataset: As shown in previous experiment, high dimensional feature vector results in oscillation of validation loss and hinders fast convergence of model. After compression with autoencoder, the test accuracy under 400 epochs of training is much lower than that of uncompressed data, which means autoencoder fails to learn the centroid info incorporated in the feature vector. It is partly because of the high compression rate, from 400 dimension to only 20,40 or 60. We also conducted an experiment where the uncompressed dimension is set to 100, and the autoencoder successfully learns the crucial info, achieving test accuracy around 60% - 70%.

2. real datasets: The test accuracy on ”recommendation”, ”citations” and ”fraud-detection” shows that feature vectors compressed by autoencoder have similar performance to uncompressed vectors. However, the test accuracy is significantly lower than the result from MLP classifier, which means the compressed output of autoencoder also pressure on the decoding capability of linear transform layer in GAT. For these datasets, autoencoder fails to extract useful information for classification. As for dataset ”fincen”, the result of compressed feature vector is about 5% lower than uncompressed feature vector, which means autoencoder fails to compress important info that helps classification.

Drawbacks of autoencoder:

1. The MSE loss enforce equal weights on all dimensions, which prevents the autoencoder to put attention on more important entries. We also test a weighted MSE loss, which balances the losses of labels/properties, so that label/property that occupies overwhelmingly many entries (e.g. string properties) will not dominant the MSE loss. However the performance of weighted MSE loss is almost identical to that of MSE loss.

   As we used one-hot encoding and disretized continuous value into bins during feature processing, it is not suitable to use MSE loss to define the distance between reconstructed vector and original vector. In this case, cross-entropy or NLL loss should be used. For embedding of string properties that generated by swing transformer, MSE loss is a proper loss function. Then the problem becomes how to combine two types of losses and effectively backpropogate the error signal. In the next section, a method is introduced to bypass this issue.

2. The separate training of autoencoder and GAT model harms the classification accuracy. The task of autoencoder is only to generally compress the feature vectors under some loss function. Without the class info, autoencoder could not perform info selection, and the pressure of decoding the feature vector comes to linear layers in GAT.

6.4 Utilizing the MLP classifier to improve GAT model

The previous experiment shows that autoencoder is not the proper method to decrease feature vector dimension. Another idea comes from transfer learning. Need to notice that the classification accuracy of MLP model outperforms that of GNN model in the first experiment, which indicates MLP model could learn more useful info for classification tasks. Then a natural approach is to combine the power of MLP and GNN, where MLP extract info from feature vectors and the GNN model fuses it with newly learned structural info from edge connectivity.

We follow the structure of MLP model in previous sections. The MLP models consists of three linear layers where dropout layer and Relu activation lies in between. The second and third layer has a residual connection to increase training robustness. The output layer of MLP is log softmax compatible with NLL loss. The MLP model is first trained with feature vectors and classification target in the training set for 200 epochs. The SBM dataset feature dimension is also set to 400. Then the output of second linear layer (with residual connection) is viewed as the compressed version of original feature vectors. Different from previous experiments, the MLP hidden layer dimension is no longer computed as $\sqrt{in\_dim * out\_dim}$, but set to 20, 40, and 60 to be compared with the result of autoencoder.

Analysis:

1. On "SBM" dataset, the MLP compressor ensures fast convergence of GAT model and outperforms autoencoder for a large margin under all compressed dimensions. This indicates the MLP compressor successfully extracts feature info and is more compatible with linear layers in GAT. The cooperation of MLP and GAT on "SBM" dataset is a good example of fusing structural info into feature info to achieve better classification accuracy than traditional neural networks.

2. On real datasets, the MLP compressor achieves similar accuracy to autoencoder.

Dataset	Test accuracy ± std.		
	compressed dimension		
	20	40	60
"SBM"	0.816±0.018	0.791±0.03	0.563±0.074
"recommendation"	0.684±0.004	0.687±0.007	0.683±0.004
"citations"	0.732±0.002	0.738±0.001	0.737±0.001
"fraud-detection"	0.217±0.017	0.236±0.011	0.242±0.005
"fincen"	0.397±0.008	0.404±0.009	0.366±0.082

Table 6.5: Test accuracy of GAT model on several datasets with MLP classifier to decrease the feature vector dimension to 20, 40 and 60.

An interesting finding is that the increase in compressed dimension does not necessarily lead to increase in test accuracy. For dataset "fincen", the middle compressed dimension gains the highest test accuracy. The high variance in test accuracy of compressed dimension 60 points out the possibility that the model performance is influenced by weight initialization and the separation of training and test set. Also it is possible that a 60 dimension input fed from MLP classifier compressor is pressuring the linear layer of GAT. For a larger hidden dimension, the MLP classifier might inject some noise that can be easily smoothed out by the dropout and activation layer, but not the GAT model.

The overall accuracy of MLP enhanced GAT is close to that of pure MLP classifier, which means the extra structural info hardly improves the classification accuracy. Several factors contribute to this phenomenon. For example, it is likely that the classification target we selected during the generation of datasets is more likely a feature of vertex itself, not highly related to its neighbouring vertices. It is also likely that the GAT model decreases the classification accuracy rather than improves it. However, according to the update function and attention score function, GAT is able to learn identical mapping. If the model is trained effectively and sufficiently, the result of MLP enhanced GAT should be at least comparable to MLP classifier.

6.5 Attention locator

Section 4.4.2 introduced 2 approaches to modify the GAT convolution layer to achieve direct attention on labels and properties. The modified GAT model should at least keep the accuracy as the original model, otherwise the attention value might not be instructive on graph comprehension as it fails to utilize the structural and feature info effectively. So the benchmark

Dataset	Test accuracy ± std.		
	Without Locator	Direct mapping	Restricted
″recommendation″	0.754±0.003	0.753±0.004	0.755±0.003
″citations″	0.725±0.002	0.717±0.002	0.718±0.001
″fraud-detection″	0.239±0.002	0.236±0.01	0.236±0.009
″fincen″	0.555±0.009	0.555±0.013	0.548±0.012

Table 6.6: Test accuracy of GAT model with two types of attention locator on real datasets.

of attention locator should focus on two parts, model accuracy and attention evolution.

The two modified GAT networks are tested on real datasets. The SBM dataset does not have a clear separation of labels and properties. Even if we view each entry of SBM feature vector as a single label/property, the recorded attention evolution can hardly provide understanding of graph. Remember the SBM graph consists of many cluster of vertices, and vertices of each cluster has the same class label. The feature vector is generated upon Gaussian noise and the coordinate of central vertex of the cluster it belongs to. In brief, attention evolution of SBM dataset won't provide any useful information.

We substitute the first GAT convolution layer in the GAT network with modified ones. The model is trained for 400 epochs. The early stopping is enabled and patience set to 60, because the model tends to overfit after a number of epochs, which may indicate the attention locator reduce the robustness of original model. Other hyperparameters are kept the same as previous experiments.

6.5.1 Accuracy

The test accuracy of direct mapping attention score function and restricted attention score function are similar to each other, and it is also comparable to that of original GAT network without compressor or locator. The GAT with attention locator can substitute the original model without a sacrifice of accuracy on these real datasets.

In theory, the direct mapping approach is a special case of restricted approach. Because if all the learnable parameters in restricted approach are set according to feature mapping matrix, then the restricted approach should produce the same result as the direct mapping approach. However, on the tested real datasets, two approaches have similar accuracy. One possible reason is that the classification on these datasets most rely on existence rather than value of vertex features.

6.5.2 Attention evolution

The final goal of attention locator is to compare the importance of labels and properties. The parameter vector a of dimension ($2 \times$ num_features) seems to be the key vector that enclose the importance among features. However, this tempting target is hard to achieve because of normalization. The labels are categorical data, and the properties can be categorical or numerical. Section 4.2 introduced several ways to transform types of properties to feature vector, including demeaning and normalizing large number (e.g. timestamp), strings to string embeddings and so on. In brief, direct comparison of feature importance is only doable if each feature has the same importance when each feature vector entry equals one. Such feature "calibration" is not studied in this project.

However, we can at least compare the attention on each label/property of current epoch and previous ones. In this way, the evolution of attention vector a in attention score function 4.5 can tell if a feature is gaining more attention from the model in the training phase.

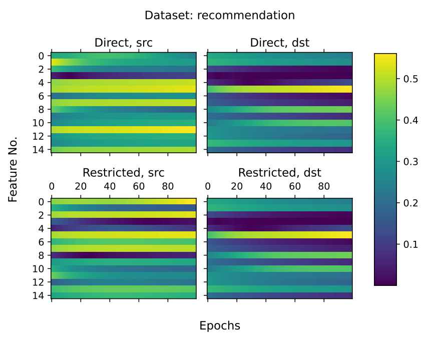
By the attention score function 4.5, the attention vector a can be decomposed as concatenation of a_{src} and a_{dst} of length $num\_features$. a_{src} and a_{dst} can be seen as attention on source/destination vertex labels and properties. The product $[a_{src}||a_{dst}]^T[h_i||h_j]$ is passed through leaky relu activation and a softmax function.

Denote a_n as the attention vector after n epochs training. To study the relative shift of attention during training, we normalized the attention vector a in the update function 4.14. We take the absolute value in the plots so that an increasing element indicates higher attention during training. As the training and validation loss do not decrease much in the last 3/4 epochs, we plot the attention evolution in the first 100 epochs.

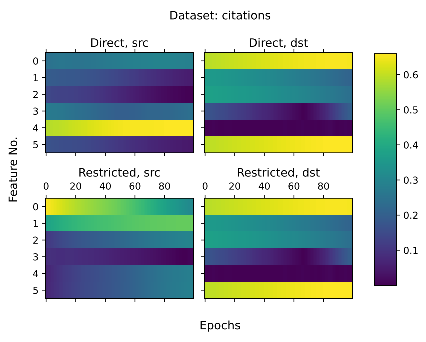
Observations:

1. The attention evolution of source and destination vertex shows different pattern. This means when the GNN determines the importance of edges, it might focus on different labels/properties of source and destination vertices.

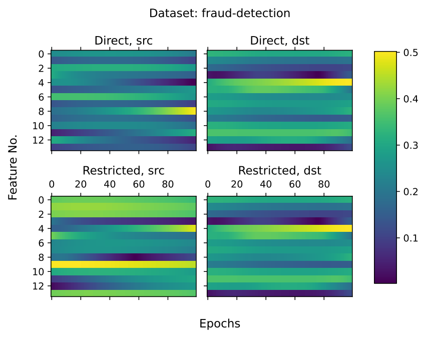
2. The evolution of a_{dst} in direct mapping approach and restricted matrix approach have similar evolution pattern, while evolution of a_{src} is more versatile. From section 6.1, we found the classification accuracy of MLP is similar to that of GAT for real datasets even though the MLP network has no information of vertex neighbours. This is probably because the vertex feature rather than edge connectivity contributes more to classification accuracy. Then it is reasonable that attention evolution of a_{dst} (vertex's own feature) is more crucial to higher accuracy, where



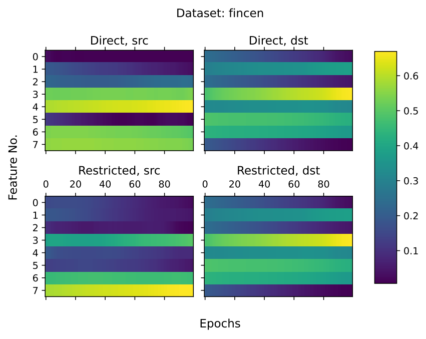
(a) ”recommendation” dataset



(b) ”citation” dataset



(c) ”fraud-detection” dataset



(d) ”fincen” dataset

Figure 6.2: Attention evolution on real datasets

two approaches behave the same. On the other hand, the GNN does not rely much on the a_{src}, so the evolution of a_{src} shows more randomness.

Chapter 7

Future work

1. **Find datasets or generate datasets that fit more into the scheme of LPG.**
   The four datasets imported from Neo4j can be seen as heterogeneous datasets. Each vertex only has one or two labels, and these labels clearly determine the type of itself and the property it holds. Also the classification target focus too much on vertex features. As a result, the edge connectivity is not covered much in this paper. The generated SBM dataset does not separate labels and properties, and also lacks edge features.

2. **Use a better compressor to compress feature vectors.**
   In this paper, we focus on feasibility of methods, so we only tested autoencoder and MLP networks, two classic models. To increase the performance and efficiency of compressor, novel neural network architectures can be integrated.

3. **Substitute the backbone GAT model.**
   In this paper, we fully rely on the GAT model as backbone. It would be interesting to try other backbone networks, and rethink how to integrate attention locator in the network.

4. **Quantify the attention evolution.**
   In section 4.4.2, we only investigated how attention score on each label/property evolves in training period. And the direct comparison of label/property's importance to downstream tasks is avoided, because the normalization of label/property's importance is hard to achieve and needs detailed discussion.

Bibliography

[1] Emmanuel Abbe. Community detection and stochastic block models: recent developments, 2017.

[2] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries, 2021.

[3] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. *Social Network Data Analytics*, page 115–148, 2011.

[4] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques and applications, 2018.

[5] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1306–1313. AAAI Press, 2010.

[6] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. Representation learning for attributed multiplex heterogeneous network. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2019.

[7] K. R. Chowdhary. *Natural Language Processing*, pages 603–649. Springer India, New Delhi, 2020.

[8] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020.

[9] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[10] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[11] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. *Proceedings of The Web Conference 2020*, Apr 2020.

[12] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer, 2020.

[14] Kexin Huang, Cao Xiao, Lucas Glass, Marinka Zitnik, and Jimeng Sun. Skipgnn: Predicting molecular interactions with skip-graph networks. *arXiv preprint arXiv:2004.14949*, 2020.

[15] Kevin Lewis, Jason Kaufman, Marco Gonzalez, Andreas Wimmer, and Nicholas Christakis. Tastes, ties, and time: A new social network dataset using facebook.com. *Social Networks*, 30(4):330–342, 2008.

[16] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. *Are We Really Making Much Progress? Revisiting, Benchmarking and Refining Heterogeneous Graph Neural Networks*, page 1150–1160. Association for Computing Machinery, New York, NY, USA, 2021.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[18] Neo4j. Neo4j - the world's leading graph database, 2012.

[19] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In Paul Groth, Elena Simperl, Alasdair Gray, Marta Sabou, Markus Krötzsch, Freddy Lecue, Fabian Flöck, and Yolanda Gil,

editors, *The Semantic Web – ISWC 2016*, pages 498–514, Cham, 2016. Springer International Publishing.

[20] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: Extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 990–998, New York, NY, USA, 2008. Association for Computing Machinery.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[23] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, Eftychios Protopapadakis, and Diego Andina. Deep learning for computer vision: A brief review. *Intell. Neuroscience*, 2018, jan 2018.

[24] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. Heterogeneous graph attention network, 2021.

[25] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[26] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[27] O. Younis, M. Krunz, and S. Ramasubramanian. Node clustering in wireless sensor networks: recent developments and deployment challenges. *IEEE Network*, 20(3):20–25, 2006.

[28] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 5171–5181, Red Hook, NY, USA, 2018. Curran Associates Inc.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

LPG2vec: Towards learning-enhanced graph databases

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Chen	Tiancheng

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 01/03/2022	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.