Graph Machine Learning

Anh Nguyen
thingocanh.nguyen@tu-dortmund.de
Technical University Dortmund
Germany

ABSTRACT

Graphs are a powerful toolbox to model any complex system in the real world. In the age of information and technology, graphs are increasing in both quantity and quality more than ever. Given the large-scale complexity of new graph datasets, it presents many challenges for the traditional techniques to analyze and extract the potential insights from this data structure. This report gives an overview of the new approach with machine learning to solve graph-related problems.

KEYWORDS

graph machine learning, graph representation learning, graph neural network (GNNs)

1 INTRODUCTION

Graphs and graph databases play a crucial role in storing and representing complex datasets. A graph data structure, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, includes a set of nodes $\mathcal{V}$ in which any pair of 2 nodes connect through an edge that belongs to set $\mathcal{E}$. Table 1 lists some attributes of graphs [5] to show how complex and varied a graph data could be in the real world. The power of graphs is about modelling relations between entities rather than storing only entities' properties. Given a great modelling toolbox and scaled amount of graph datasets, a raising question is how to take advantage of relational structure data in graphs to make a better prediction automatically.

The report is divided into four sections. Section 1 shortly introduces the topic with a brief explanation about graphs. Section 2 is dedicated to graph representation learning, limitations of the previous approaches and how graph neural networks can help overcome those problems. Section 3 presents an experiment with benchmark datasets. The last section summarizes the results and outline the whole approach.

2 GRAPH MACHINE LEARNING

This section goes into graph representation learning methods and graph-related problems. Firstly, it presents an overview of the traditional techniques to learn about the representation of graphs. Then it explains a modern approach to achieve this goal using deep neural networks with graphs that known as graph neural networks.

Machine learning tasks in graphs often do not have a clear border between supervised or unsupervised methods. Sometimes it is referred to as semi-supervised. It comes from the fact that the model may see all nodes and relations in the graph instead of only the training set during the learning phase. Machine learning tasks on graphs can be grouped into different categories depending on different perspectives. Firstly, as graph structure is complex enough to be able to model many hierarchical dependence between objects, depending on the level of interest, machine learning models can make predictions at the corresponding graph, node or edge level. Generally, there are some standard graph-related tasks named node classification, link prediction, community detection, and graph classification [2].

2.1 Motivation

Traditional learning approaches over graph data has been well-studied for years before machine learning emerged. Graph traversal algorithms or PageRank have been successfully deployed to solve real-world applications. Besides that, graph analysis has achieved promising results when using traditional machine learning pipelines on graph features which are extracted by methods of statistics or graph kernel. Those methods are powerful and able to make accurate predictions. However, the qualities of models depend on the (manual) feature-engineering steps, which requires understanding about graph data itself and domain knowledge from a human expert. This approach poses many limitations when dealing with large-scale graphs. Hand-design features are time-consuming and not reusable for a new graph. Figure 1 depicts the working flow this approach [4].

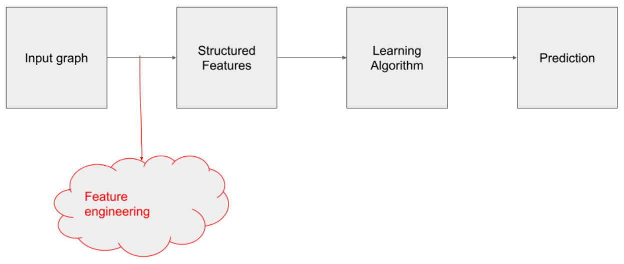


Figure 1: Traditional machine learning approaches with graphs

Graph representation learning (GRL) will be the main discussion over the rest of this report. To overcome limitations from the manual step, such as preparing hand-crafted features, GRL is a technique that aims to learn a set of features (or representation) of graph data in an automatic way from the structure of graph data itself.

2.2 Graph representation learning

GRL is a general task-independent feature learning framework. Graph feature could be understood as node feature, edge feature or feature for an entire graph. The learned features are expected to include characteristics and connectivity between entities in the graph and are induced from the structure of graphs. In addition, the learning processing does not require human input (through feature engineering step) and domain knowledge. The generated

Table 1: Graph Attributes

Attribute	Description
Connected vs Disconnected	A path between any 2 nodes
Directed vs Undirected	(existed) a start/end node
Weighted vs Unweighted	A value for a specific edge/node
Cyclic vs Acyclic	A path starts or ends at the same node
Sparse vs Dense	Ration of node degrees in graph
Monopartite, bipartite, k-partite	Number of node types, and connection between different node types

features can be used as input to solve different types of graph-related problems or specific downstream tasks, however, it does not have to be dependent on the target of upcoming tasks.

The learned features are usually represented in low-dimensional vector embeddings. The structure of graphs is preserved after transforming from the original dataset to embedding spaces. The goal of GRL is to learn function $ENC : \mathcal{V} -> \mathbb{R}^d$, where $\mathcal{V}$ is a set of nodes and d is the dimension of embedding space. That maps the original graph dataset into an embedding space so that the locality information is the same between both original and embedding spaces. However, there are no techniques guarantee to keep 100% of information from the original dataset after the transformation process [2].

For example, if the locality and connectivity between a node and its neighbourhood are the priority for model to learn, it means that the set of nodes that are connected or similar in the original dataset tends to closer in the embedding space (the absolute distances between similar nodes are smaller while the distances between dissimilar nodes are larger). This idea is well illustrated in Figure 2 in which the original dataset contains 4 communities and members of the same community still being closed in the new representation space [7].

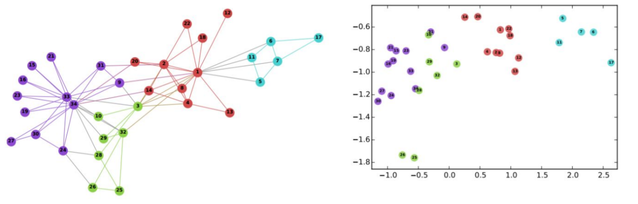

(a) Input: Karate Graph (b) Output: Representation

Figure 2: Graph representation learning in Zachary's Karate Club problem

Sometimes, graph representation learning can be referred to as graph embedding. In this view, GRL can be described as an encoder-decoder framework (Figure 3 ref from [2]).

In this architecture, the objective that guides the learning processing is to preserve the locality of nodes, which means a set of nodes that are directly connected are expected to have similar vector embeddings. The encoder learns to map each node in the graph to an embedding vector while the decoder reconstructs information about the immediate neighbourhood of this node. The vector embeddings contain summarization information about the node's

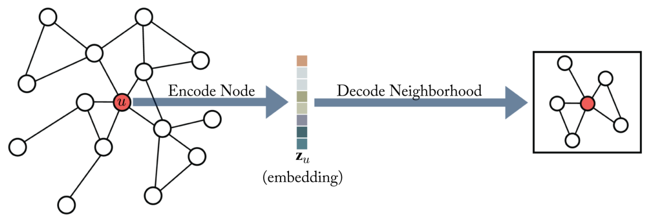


Figure 3: Encoder-decoder approach

position in the graph and the local structure of its surrounding neighbour nodes. In this architecture, the encoder is defined as the farget function *ENC* that mentioned above [2].

Many techniques that follow the above definition are categorized as the shallow embedding approach. The definition also implies major limitations of this approach. The first is a lack of generalization, or called transductive, for new nodes or graphs because the encoder learns and optimizes only for nodes that appear during the training process. Secondly, the learning procedure takes into account only graph structure and ignores potential information from node/edge features. The final limitation is inefficient computation and memory while the model separately optimizes for each node, has no parameter sharing between them and the size of embedding vectors is equal to the number of nodes in graph, $|\mathcal{V}|$, which could be millions in a real-world dataset.

The next section will present the concept of deep graph encoders that use deep neural networks to alleviate those limitations of shallow encoders. Intuitively, the new goal is to learn a deep neural network model that takes input as a whole graph and give any desirable output, for example, node embeddings or graph property prediction.

In addition, other than learning about graph representation separately with the downstream task, the final prediction (of downstream task) can be included in and the whole learning process, which includes the forward and backward phase, can take place [4].

2.3 Graph neural network

This section introduces the advanced approach named graph neural network (GNN) which is a very general framework to apply neural networks on graph-structured datasets. Recall goals and problems that were presented in previous sections about current states of graph-related tasks with traditional (machine learning) methods, GNN shows many advantages over those techniques. Firstly, it requires no hand-crafted features for the input, given a raw graph dataset, the model learns and generates representations (graph, nodes) from the structure and attributes of the graph. Secondly, it takes into account node features (if that information is presented) as the initial embedding vectors for the training process. Besides that, GNN presents an end-to-end manner to solve both graph representation learning and downstream tasks in one learning process. Taking an example of node classification problem, rather than training two separated modules (the first model manages representations for graph, the second is another classification model), the available label of nodes could be used to guide GNN model to learn similar embedding vectors for nodes sharing the same label during the training process. Ultimately, GNNs inherent the generalization capacity of a deep neural network and be able to generate a representation for a new node or a new graph. This inductive capability is an important characteristic of GNNs over shallow methods that enable the ability to deal with dynamic graphs [2].

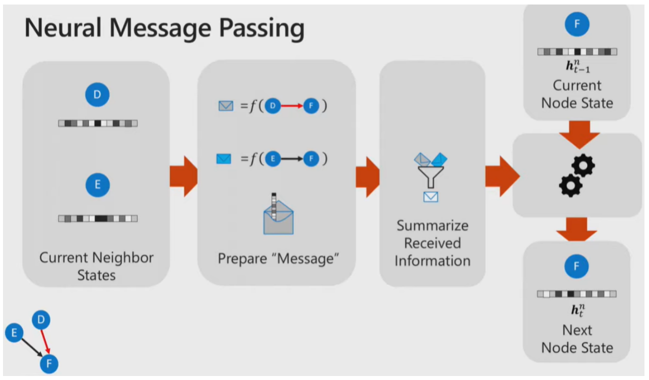
Inspired by the success of convolutional neural networks (CNNs) by applying convolution operators on different (local) parts of images, CNNs can learn about higher representation (or abstract representation) of objects in an image. The same idea is reapplied to graph-structure data, generate node embedding from the nearby neighbourhood for a target node. This idea is formulated by the neural message passing framework. In this framework, information from each node is considered as a message, a set of connected nodes will exchange messages to each other, after receiving and aggregating messages from neighbour nodes, each node will update itself with this new information. Figure 4 (ref [1], [4]) illustrates what happens when applying a neural message passing step for a target node.

Intuitively, after each message-passing iteration, each node learns about its neighbours, then at the next step, it learns about neighbours of neighbours. After many iterations, each node is able to know about its position in the whole graph and repetitively update itself. In that way, the final representation is derived from the internal structure of the graph. The tree structure of 2 message-passing iterations is called the computation graph for a specific target node. The same process is applied for every other node, which means each node will have a different computation graph structure because each node is connecting to different nodes in the network.

GNNs can have a formal definition as follows. Given input with a graph $ENC : \mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a node features matrix $X \in \mathbb{R}^{|\mathcal{V}| \times d}$. z_u is the final representation of node u after the training process. A GNN model can contain any k layer, each layer corresponds to a message-passing iteration run over the whole graph. A message-passing update can be defined as [2]:

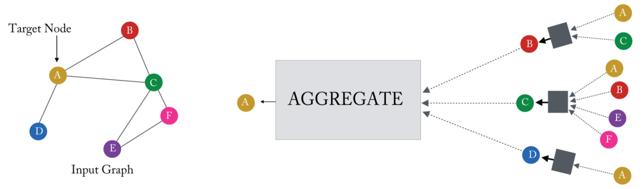
$$\begin{aligned} h_u^{k+1} &= UPDATE^k(h_u^k, AGGREGATE^k(\{h_v^k, \forall v \in N(u)\}) \\ &= UPDATE^k(h_u^k, m_{N(u)}^k) \end{aligned} \tag{1}$$

Where k indicates current iteration, h_u^k corresponding to embedding vector of node u after k-iteration, $N(u)$ is set of neighbour nodes for node u. At iteration $k+1$, the target node u will gather messages passing from its neighbour $N(u)$, compress into a single



(a) with 1-hop

Note. Neural message passing. From *MSR Cambridge Lecture Series: An Introduction to Graph Neural Networks: Models and Applications*, by Miltos Allamanis, 2019, Microsoft Research.



(b) with 2-hop

Note. Overview of how a single node aggregates messages from its local neighborhood. Adapted from *Graph Representation Learning* (p. 53), by Hamilton, 2020, Morgan Claypool Publishers. Copyright 2020 by Hamilton, W. L. Adapted with permission.

Figure 4: Neural Message Passing flow

message by the *AGGREGATE* operator, then use the *UPDATE* operator to merge with its current state h_u^k, forming a new embedding state h_u^{k+1}. The initial embeddings at iteration $k = 0$ are default to use node features, $h_u^0 = x_u$. Each iteration is equal to a layer of GNN model, the output of the last layer, the last iteration K, is considered as learned embeddings for all nodes. $Z_u = h_u^K$.

The *UPDATE* and *AGGREGATE* operators can be any arbitrary differentiable function, it could be as complex as a neural network layer, or a non-linear transformation such as the $tanh()$ function, or as simple as max or average operator.

It is clearly that different choices of aggregated or updated functions will lead to different variants of GNNs. In general, when define a GNN architecture, it is needed to specify the three following questions: How to prepare message from node neighbors? How to summarize messages from neighbors? How to update current state with new information?

As GNN model required node features as initial vector embeddings, sometimes when this information is not available, the alternative option is using node features from graph structure. The first choice is using an adjacency matrix where each row corresponds to the feature vector of each node. The second choice is to construct a one-hot encoding vector for each node, with node degree used to fill the non-zero element. A promising approach is to generate initial node embeddings using shallow methods, such as DeepWalk or Node2Vec, then use GNN to enhance the quality of embedding vectors. The interesting part of this approach is that the pre-trained vector embeddings contain information about the graph itself, it helps GNN models learn and converge quickly, then GNN combines it with related information about downstream tasks and makes better predictions.

2.3.1 Basic GNNs model. To come up with a specific GNN model from the general definition (Eq. 1), the first step is to specify aggregation and update operators. The very first GNN model is proposed by [6], which is considered as the most simplified version of GNNs variants, is defined as:

$$h_u^k = \sigma(W_{self}^k h_u^{k-1} + W_{neigh}^k \sum_{v \in N(u)} h_v^{k-1} + b^k) \tag{2}$$

When breaking the equation down, it shares many similar characteristics with a traditional multi-layer perceptron (MLP) model. Firstly, the weight matrix W_{self}, W_{neigh} and bias vector b are trainable parameters of the model. The aggregation function basically sums up all messages from the neighbour set to generate the message $m_{N(u)}^k$, with $m_{N(u)}^k = \sum_{v \in N(u)} h_v$, then combines with information from the previous iteration. The update operator is applying a non-linear transformation for the final aggregated message and updating the current state of target node u.

The set of parameters W, b can be shared across all layers of the model, or be separately optimized for each layer. However, unlike standard deep neural network models, the number of parameters and the depth (number of layers) of GNNs model does not tell how complex or how expensive computations of the model is. The complexity of the model is defined in the specific computational graph of each node at each layer.

The basic GNN model above is simple and easy to implement. However, a graph dataset is usually complex and varied as a node can connect to any number of nodes. For an example of a social network, an average node degree could be 200 - 400 while existing a set of particular nodes connected to a few thousand nodes. Inherently, the message passing will apply the same sum (aggregate) operator over all nodes and subsequently poses a high imbalance of aggregated results between nodes. A simple solution is normalizing the aggregated result with node degree after taking the sum.

$$m_{N(u)} = \sum_{v \in N(u)} \frac{h_v}{|N(u)|} \tag{3}$$

or using the symmetric normalization version, which considers both target node and neighbor node, as:

$$m_{N(u)} = \sum_{v \in N(u)} \frac{h_v}{\sqrt{|N(u)||N(v)|}} \tag{4}$$

However, it's not a good idea to always use normalized aggregation. The reason is that messages coming from neighbours and the node itself becomes indistinguishable from the model.

Although the Eq. 1, which uses to define the neural message passing and how GNN works, looks more straightforward. A common way to simplify it is to ignore the update operator and implicitly include this step in the aggregation function. This technique is named message passing with self-loop. Eq. 1 will become as:

$$h_u^k = AGGREGATE(\{h_v^{k-1}, \forall v \in N(u) \cup \{v\}\}) \tag{5}$$

One of the most popular models that employs both self-loop and symmetric-normalized aggregation is Graph Convolution Networks (GCN), proposed by [3]. The message passing function of GCN is defined as

$$h_u^k = \sigma(W^k, \sum_{v \in N(u) \cup \{v\}} \frac{h_v}{\sqrt{|N(u)||N(v)|}}) \tag{6}$$

In comparison with CNN model, the message-passing is analogous with convolution operation. The architecture of GCN models expresses a nice property of parameter sharing when W_{neigh} and W_{self} become one identical weight matrix. The subsequent section of this report will present an experiment with GCN and show the promising outlook with such a simple architecture.

2.3.2 GNNs in practice. The previous section describes how the GNNs framework works through the concept of neural message passing. In theory, GNNs are suitable to solve four graph-related tasks: graph representation learning, node classification, relation prediction and graph classification. But how to train GNN models to solve a wide range of problems in practice? This section provides an overview of the training procedure when applying GNN to node classification tasks. Then the subsequent section is about hands-on experiments with a simple application and real datasets.

Depending on the target problem and characteristics of the graph dataset, training a GNN model can be approached either by supervised learning or unsupervised learning methods. Similar to the traditional machine learning pipeline, Figure 5 shows a (general) training pipeline when applying GNN to graphs.

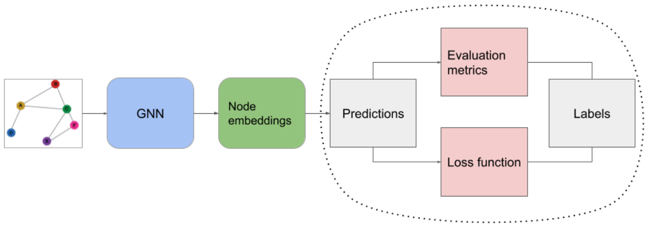


Figure 5: GNN training pipeline

According to the application, the expected predictions and available labels, specific loss function and evaluation metric must be defined to guide the learning process. For example, how to calculate error loss and run backward procedures to update the model's parameters?

For node classification tasks, a GNN model can be trained in a supervised or semi-supervised manner. In detail, all nodes are separated into training and test set. The model knows about true labels from the training set and uses this information to optimize its parameters. Labels from the test set are hidden from the model and will be presented only during the test phase. However, when doing the neural message passing, messages from all connected nodes are exchanged and passed to equation Eq. 1. Therefore, considering a GNN model is a set of weight matrix and bias vector, the model sees all node data during the training process. That makes the training

Table 2: GNNs with applications

Domain	Task	Category	Brief description
Social networks	Bot detection	Node classification	Each node represents a user which could be a human or bot. A subset of users is verified as human or bot. Semisupervised method to predict labels for the rest of user nodes.
Natural language processing	Topic modelling (classification)	Node classification	Each document (paper) is a node and those documents are linked by citations. Predict research area given document content and citations.
Computer vision	Identifying human action	Graph representation learning	A sequence of frames composes a spatial-temporal graph. Each node corresponds to a human joint at a time frame, spatial-temporal connections are relations between nodes. GCN is used to perform feature extraction as input to a classifier (CNN) which performs human action recognition tasks.
Ecommerce	Recommendation	Link prediction	A node represents a user or an item. Any reaction (buy, view) from a user to an item node is defined as edges in the graph. The task is to predict the probability of non-existing relation between any pair of a user-item node.
Traffic	Estimate Time Arrival	Prediction at graph-level	Divide road networks into many supersegments, each supersegment is modelled as a graph $G = (S, E)$ where each node is a road segment, an edge is formed between any two connected road segments. A node has length and speed features.
Social networks	Classify a Reddit thread/discussion	Graph (level) classification	Each thread is modelled by a graph with nodes being users and edges being a reaction (comment, reply) between them.

process semi-supervised. If the set of test nodes is completely hidden from the model, it is considered fully-supervised. However, since GNN requires to know about structure and relation in graph data to infer about its representation, removing a part of graph structure is not preferred because it causes information loss and could be a negative impact on the learning of model about the graph. Denotes y_u is the actual label of node u, representing as a one-hot vector, and z_u is the vector embedding of node u that was generated by the GNN model. A standard loss function such as cross-entropy can use to measure the performance of the model, then apply the stochastic gradient descent (SGD) algorithm to optimize model parameters.

For graph classification tasks, an operator that needs to be defined further is how to aggregate vector embeddings from all nodes to have a single representation for the whole graph. The dataset contains many graphs that share similar characteristics. Two subsets of graphs can be completely separated into training and testing sets so that fully supervised is usually applied to train a graph classifier.

An example with link prediction tasks was presented during the presentation session.

GNNs can be applied to solve many tasks in different domains. The task needs to be formulated appropriately with one of the standard graph-related problems which are described in the introduction section. Table 2 presents some well-known applications that GNNs open a new approach to solving the problem over traditional methods.

3 EXPERIMENT

Node classification will be the focus task in this experiment. In order to show the power of the GNNs model, a simple GCN [3] is chosen with 2 neural message passing iterations. The experiment is conducted on 2 datasets, which are selected from different domains. The first dataset is a well-known benchmark Cora dataset from [9]. The second is a product dataset from [8]. Those 2 datasets are different in both quantity and quality. The description of datasets and details of the experiment can be found Table 3 in the section Appendix.

To sum up, the selected GCN model performs well in both datasets. These 2 datasets are sharing the same node features, which are represented by text features, such as bag-of-word. Given the textual datasets, the existing techniques can be able to solve the problem of text classification and representation learning, such as topic modelling or word embedding. However, these techniques are not able to explicitly take advantage of relations between those documents. It illustrates the very important property of GNNs over the other standard approaches.

Finally, the experiment shows the challenge of runtime and memory when computing messages through the computation graphs. When the size of graph datasets grows, or more iterations are computed, the learning time for GNNs increases dramaticcaly.

4 SUMMARY

The data today is extremely complicated. Graphs, graph data structure, and graph databases come to deal with data storage and data management. Graph analysis provides a set of techniques to extract knowledge and insight from graph databases. Graph machine learning is the new approach to solve graph-related problems efficiently and effectively.

Graph representation learning helps transform large-scale information from graphs into numerical vectors; from that, machine learning algorithms can be applied to graph datasets and make predictions. Besides that, GNNs define a framework, or a recipe, to generalize the way doing machine learning in graphs. While there are more and more problems being modelled by graphs, GNNs present promising results over other techniques. The key advantage of GNNs over other techniques is GNNs can take into account information from relation while the others fail to directly include that information.

Graphs are complex and dynamic, the same graph can be represented in different ways. It means when a graph is transformed to another format, it is likely that the GNNs model will learn different representations. This claim implies that to have a better prediction, the quality of graph data is more important than building a complex model. The previous experiment contributes to supporting this claim when using a simple GCN model, it is able to perform well on different datasets, given the condition that graph datasets are well examined.

ACKNOWLEDGMENTS

A special thanks to my supervisors who are carrying the seminar "Neural Networks and Deep Learning", in the winter semester 2021/22, to give me a chance to follow my desirable topic.

REFERENCES

[1] Miltos Allamanis. 2019. MSR Cambridge Lecture Series: An Introduction to Graph Neural Networks: Models and Applications. https://www.microsoft.com/en-us/research/video/msr-cambridge-lecture-series-an-introduction-to-graph-neural-networks-models-and-applications/. Accessed: 2021–12-21.

[2] William L. Hamilton. [n. d.]. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14, 3 ([n. d.]), 1–159.

[3] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs.LG]

[4] Jurij Leskovec. 2021. CS224W: Machine Learning with Graphs. http://web.stanford.edu/class/cs224w/. Accessed: 2021–12-21.

[5] Amy E. Hodler Mark Needham. 2019. Graph Algorithms. (2019).

[6] Christian Merkwirth and Thomas Lengauer. 2005. Automatic Generation of Complementary Descriptors with Molecular Graph Networks. *Journal of Chemical Information and Modeling* 45, 5 (2005), 1159–1168. https://doi.org/10.1021/ci049613b arXiv:https://doi.org/10.1021/ci049613b PMID: 16180893.

[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk. https://doi.org/10.1145/2623330.2623732

[8] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Pitfalls of Graph Neural Network Evaluation. arXiv:1811.05868 [cs.LG]

[9] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. arXiv:1603.08861 [cs.LG]

A EXPERIMENT

Table 3: Node classification with GCN: datasets and results

Attribute	Cora	Amazon
#node	2708	13752
#egde	10556	491722
#node features	1433	767
#node classes	7	10
#avr node degree	4	36
#training	140	1000
#testing	1000	2750
#iteration	100	800
accuracy	82%	69.81%
excuted time	2 secs	6.3 mins

B ONLINE RESOURCES

The code and detail steps to produce the experiment can be found at Google Colab notebook.