CSE 676 : Deep Learning Project 2

Karan Bali
Department of Computer Science
SUNY Buffalo
Buffalo, New York
kbali@buffalo.edu

Abstract

The objective of this project was to detect fraudulent bitcoin transactions using a Graphical neural network. This project was directed as a feasibility study of another paper related to the same task. The task was to execute and compare different machine learning algorithms for the given objective of predicting fraudulent bitcoin transactions in the given [2]"Elliptic Bitcoin transaction graph" dataset. The code related to the project was provided in supplementary materials attache to this report.

1 Introduction

The proliferation of the internet has been followed by online financial transactions. More than 70 billion transactions were committed during the year 2020. The online transaction has been a great tool in the hands of governments and citizens to keep the wheels running during the tough times of COVID-19. Basic affirmative actions of the government were implemented via online medium. For example, basic minimum wages, subsidies to the poor, etc were given through online transactions in the account of the beholder.

However, in recent years bitcoin has seen an exponential rise in the share of this "Fin-Tech" world. Similar proliferation has been observed in various crypto-currencies around the world. During the initial time, the Government had been unsure of what to do with these crypto-currencies. They couldn't stop the proliferation even if they wanted to. Thus, a large ecosystem of this crypto world was left unregulated.

The unregulated crypto-currencies became a huge tool in the hands of market-distorting investors like hedge-funds. Thus, we saw a huge boom in the market of bitcoin as an alternative currency. Many started professing crypto-currencies as an alternative to the US Dollar. This was due to many reasons.

First of all, crypto-currency was more universal in nature than the US dollar. It wasn't owned by a single central bank of a single nation. Hence, the market was less summoned to the whims of a single nation. This fact greatly interested those nations and investors, which were against unilateral financial sanctions by a group of countries.

Second, it became a tool to bypass the national taxation system. For, example, a virtual artifact like bitcoin owned in the US could be used in any third country bypassing international remittance taxes. This fact can be used by anyone with malicious intent. A terrorist organization or any inimical entity can use bitcoin to finance its field workers to execute sabotage activities.

After observing all the above factors, many governments around the world started framing some rules and regulations concerning the use of bitcoins. This led to another proliferation of startups related to the ecosystem of crypto-currencies. One of the startups is [2]"Elliptic Ltd", a block-chain analysis company specifically targetting Anti-money laundering analysis using bitcoin.

A paper called [2]"Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolution Networks for financial forensics" was released during KDD 2019. The authors of the paper were affiliated with IBM and MIT. The authors ran a series of experiments to study the effectiveness of different ML algorithms on a dataset that happens to be graphical in construct. We would be performing a feasibility study of the results conclusions provided in the paper. Before that, we'll go over some literature-review required for the understanding of the paper.

Before moving on to the experiment, we'll go through some Literature-review required for the conceptual understanding of the paper.

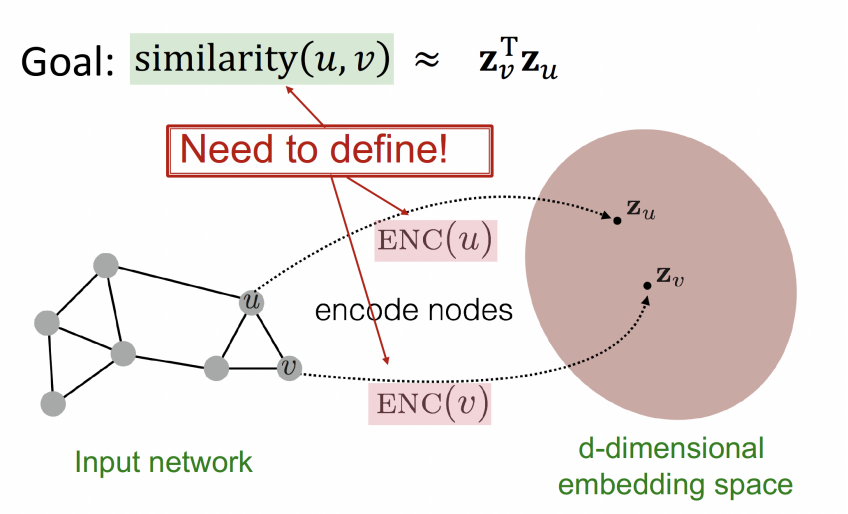
2 Literature Review

2.0.1 Graph Convolution Network

Graphs are ubiquitous in our daily lives. They are present in biological entities (like chemical compounds, an organism's living systems) and are also present in virtual entities (like financial networks, market ecosystems). Researchers have been studying graphs for a long time. Thus, many advances have been made in the field of Graph theory and Graph representation learning.

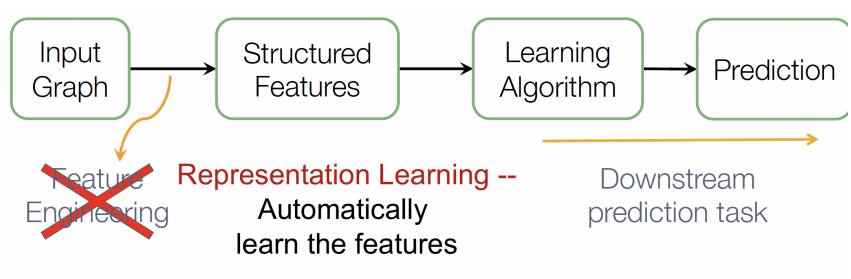
Recent times have seen big breakthroughs in the field of Artificial Intelligence and Neural Networks. The application of CNN for Image processing has been magical. The next obvious question in the mind of researchers has been to apply the advances made in deep learning to the field of Graph Machine Learning.

The whole idea of GML is to learn a function that maps the input space to an embedding space. We use the embedding representation to further classify the nodes, edges, and graphs. This embedding space should be such that similar nodes are embedded nearer to each other in comparison to other nodes. GML tries to learn the appropriate embedding space and function for the given input.



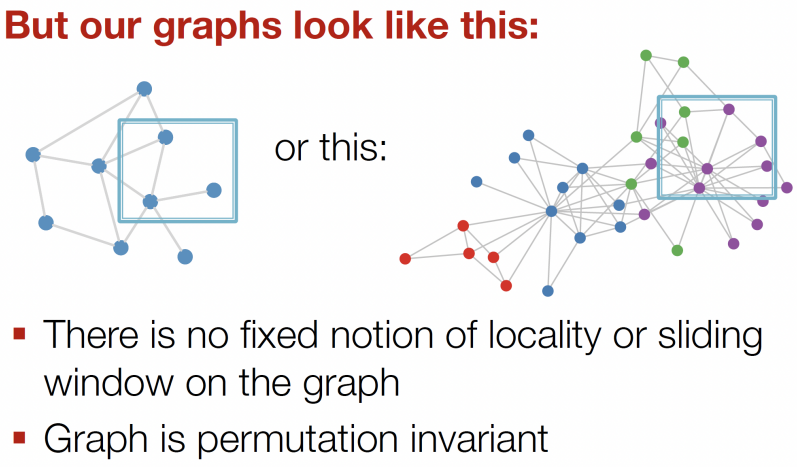
Ref: [1]

The evolution of Graph Machine Learning (GML) has been similar to the evolution of traditional machine learning. Like traditional machine learning, initial GML methods depended upon task-specific hand-crafted features. Thus, feature engineering was a big part of the traditional GML. We all know, how advances made in deep learning replaced the feature engineering part with something more automatic feature learning by neural networks. A similar approach is applied to GML.



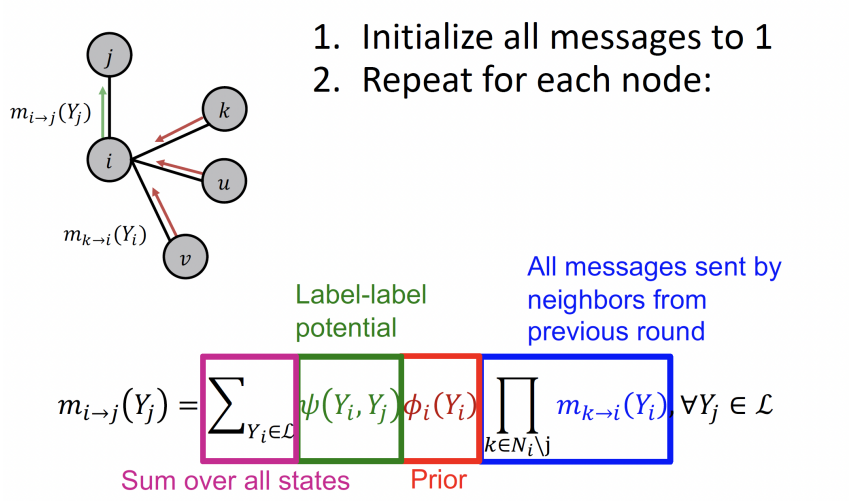
Ref: [1]

However, there's a fundamental difference between a conventional data structure like images and a graphical data structure. Conventional data structures have a sense of ordering (i.e. one can deterministically identify upper neighbors and side neighbors). However, graphical data structures are permutation invariant and lack that sense of ordering (i.e. one can't deterministically identify upper neighbors and side neighbors). Thus, it's not plain simple to apply convolution in a graphical data structure.



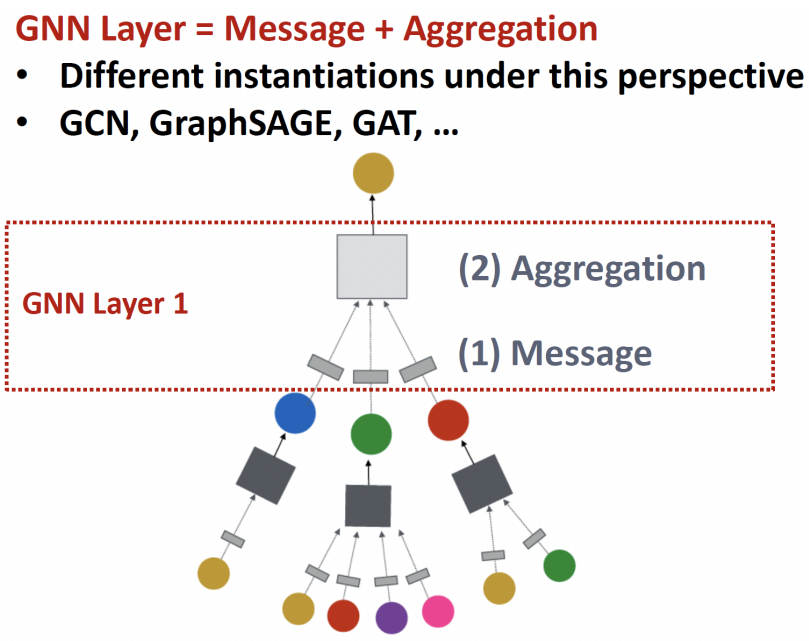
Ref: [1]

We need to get some measure of distance between the nodes. GNN uses the links between a node its neighbors as a measure of nearness. When we apply a convolution function on a graphical dataset, it's called a Graphical Convolutions Network (GCN). A GCN uses the "Message Passing Algorithm" from graph theory and combines it with tools from deep learning. A message-passing algorithm is a type of collective inference classifier, which applies a relational classifier iteratively to each node till the inconsistency between the network settles down.

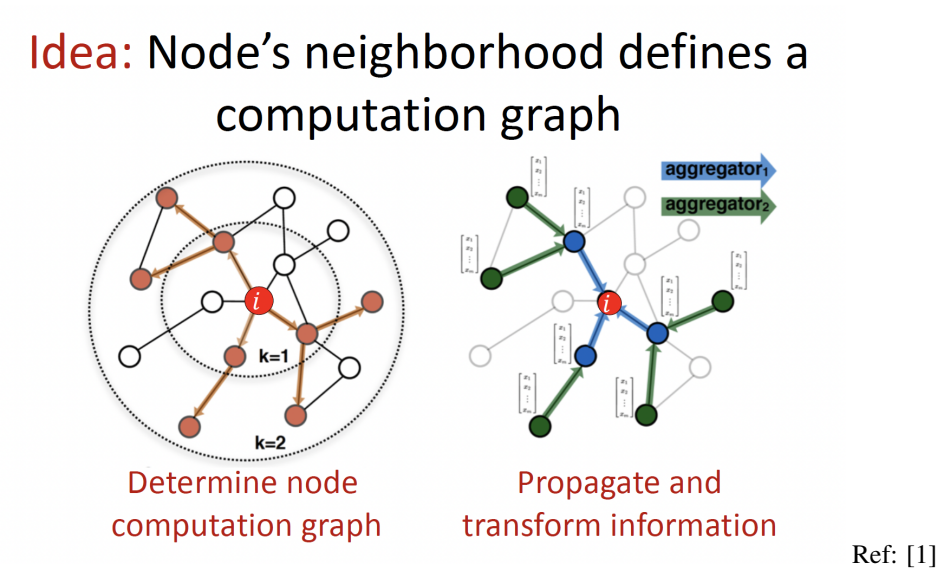


Ref: [1]

A GCN can consist of many layers. Each GCN layer has two main components, "Message Formulation" and "Message Aggregation". During Message formulation, input from K-hop neighbors is fed to a neural network. This transforms the input features. Finally, "Message Aggregation" aggregates the output from the neural network to formulate a single embedding for the current GNN layer. This embedding is passed onto the next GNN layer. Hence, this continues till we get a final node embedding. All GNN layers have different sets of neural networks with a different set of weights and biases. However, all the respective layers of GNN are shared by all the nodes. Hence, each node gets trained on the same set of GNN layers. Thus, each node dynamically generates a computation graph for itself.

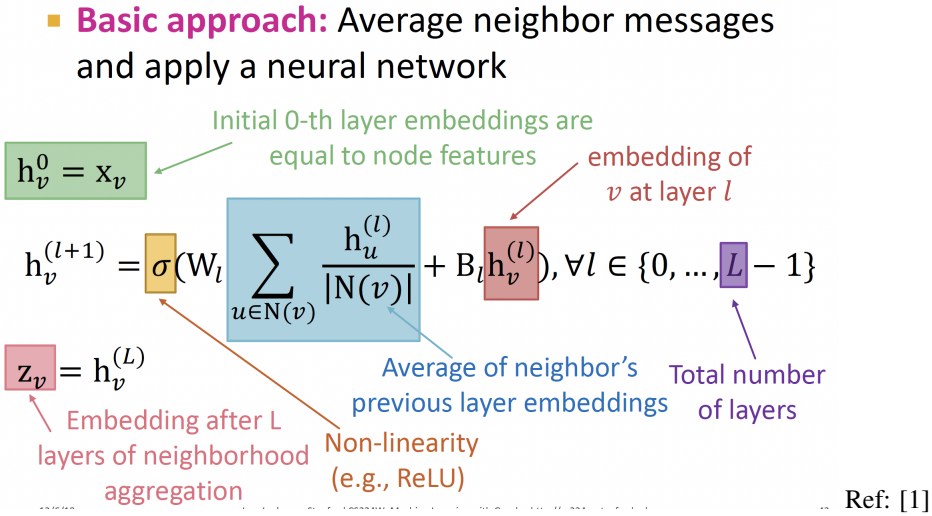


Ref: [1]



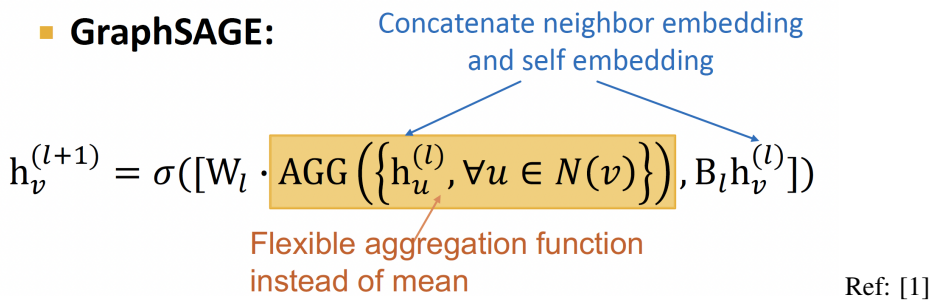
Ref: [1]

The equations for the GCN are clearly explained in the figure given below. The message formulation, aggregation, trainable weights and biases are color marked.



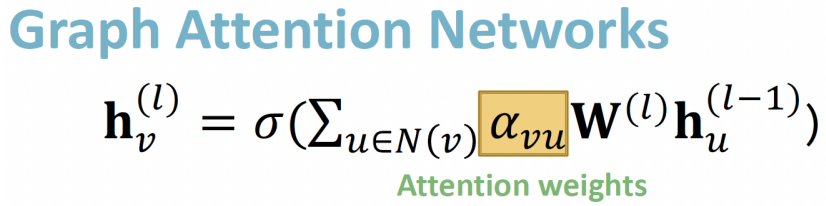
Ref: [1]

The original GCN used SUM (summation) as an aggregation function. However, one can use a variety of Aggregation functions like Average, Mean, Max pooling even an LSTM (However, LSTM assumes some orderly notation in the graph). This variation to the GCN is called GraphSage.



Ref: [1]

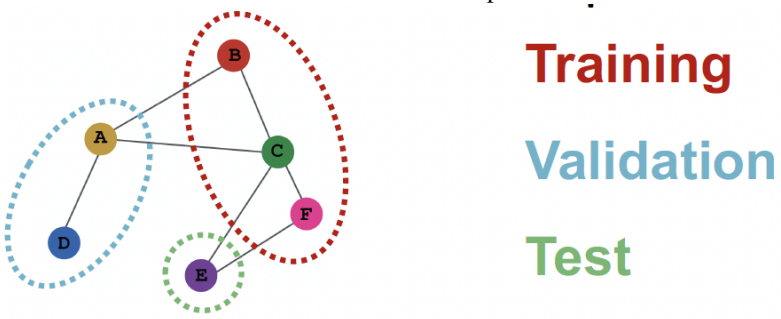
We can also have a situation where different neighbors of a node might have different importance. We use the concept of "Attention" from deep learning and apply it in a GCN setting. This variation is called Graphical Attention Network (GAT). GAT has specific learnable weight parameters for each neighbor. These weights can be learned by various attention techniques.



Ref: [1]

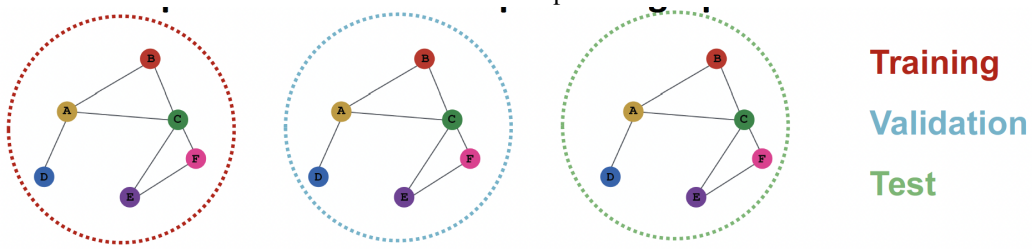
One of the unique challenges of training a GCN is figuring out how to split the data. There are two ways to split data from a GCN perspective. The "Transductive method" uses the same graph for training, evaluation, and testing purposes. It achieves this by masking particular nodes. The "Inductive method" uses different data for training, evaluation, and testing purposes.

Transductive split



Ref: [1]

Inductive split



Ref: [1]

2.0.2 Multi-Layer Perceptron

Multi-Layer Perceptron is one of the simplest feed-forward neural networks. It has 3 layers with one input layer, one hidden layer, and one output layer. It uses backpropagation to learn the optimal weights for the input data.

2.0.3 Logistic Regression

Logistic regression was designed as an extension to Linear regression. Linear regression tries to fit a linear hyperplane to separate and classify the data. It doesn't provide a probability. Thus, a logistic regression predicts a probability of an outcome. So although the name suggests Regression, in reality, it's used for classification. It uses a sigmoid function to calculate the probability distribution among the possible outcomes.

2.0.4 Random Forest Classifier

Multi-Layer Perceptron is one of the simplest feed-forward neural networks. It has 3 layers with one input layer, one hidden layer, and one output layer. It uses backpropagation to learn the optimal weights for the input data.

3 Dataset

Recently, elliptic published a bitcoin transaction graph dataset[2]. The dataset had 3 files defining the edges of the transaction graph, classes (i.e. licit and illicit transactions), and features of each transaction. Thus, each transaction had 166 features out of which 96 are local features and the remaining 72 are aggregated features from the neighbors of the transaction node. Similar to the paper, we would be using one of the temporal features to split the dataset into a training and testing dataset.

The dataset is available at Kaggle on the following link: https://www.kaggle.com/ellipticco/elliptic-data-set

4 Experiments

The paper performs a series of experiments on the [2]elliptic dataset. In total, variants of 4 methods (i.e. Random Forest classifier, Logistic Regression, MLP GNN) were trained. The paper also had variants that used only local features or embeddings from a GNN. As this is a feasibility study, We would be training the main variant of each 4 methods that use all 166 node features. We would also train a custom GNN model and compare the results with other algorithms.

We'll use a similar set of parameters and hyper-parameters used in the original paper. For example, GCN will use the "Inductive method" to split the data. It will use 2 layer GCN with 100 size hidden embeddings. Both MLP and GCN will use Adam optimizer with a learning rate of 0.001. GCN train for 1000 epochs and MLP trains for 200 epochs. Random forest classifier (50 max features and 50 estimators) and logistic regression will use default settings from the scikit package. Some of the settings and parameters for the experiments might seems like most common on the internet, but i just went by the values provided in the original paper. Had it been my own choice, I might have chosen other values.

5 Related Code and files

There 2 python notebooks in the folder provided as supplementary material to this report. The first file contains 2 GCNS. The first GCN is a normal GCN as mentioned in the paper. **The second GCN is a custom GCN coded by me, which uses an "Attention Layer" in a GCN**. The second file contains the MLP, random forest classifier, and logistic regression classifier. The data-related CSV files are also provided in the folder.

6 Results and Evaluation

Paper uses F1 score and MicroAvg F1 score to compare different methods. As the dataset is highly skewed towards positive class, MicroAvg F1 score is a better metric to gauge the accuracy of different methdods.

****NOTE: GAT-GCN is coded and implemented as an add-on experiment by myself. I wanted to see how an attention layer and GAT would perform for a dataset which provides a better result for Random Forest classifier in comparison to a plain GCN. The idea was that may be GAT would give more weightage to important relational features in comparison to local features. However, GAT didn't perform accordingly.**

Results provided in the paper:

Method	F1	MicroAvg F1
GCN:	0.628	0.961
MLP:	0.653	0.962
Logistic Regression:	0.481	0.931
Random Forest Classifier:	0.788	0.977

Results achieved by me:

Method	F1	MicroAvg F1
GCN:	0.665	0.953
MLP:	0.483	0.966
Logistic Regression:	0.67	0.937
Random Forest Classifier:	0.90	0.988
GAT-GCN:	0.54	0.93

7 Conclusion and Future work

Contrary to the intuition that GCN would work better on learning a graphical dataset, the Random Forest classifier came up as the best algorithm among all the experiments. This result asks for further research and analysis as a random forest classifier is a fairly simple model in comparison to a GNN or MLP. Our results were more are less in line with the results provided in the paper. Actual values might have differed, but the larger trend of Random Forest classifier performing better than a GCN and other algorithms was visible. One of the future tasks mentioned in the paper asks for using intermediate embeddings from a GCN and injecting them into a Random Forest classifier.

References

[1] Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, Link: http://web.stanford.edu/class/cs224w/

[2] arXiv:1908.02591, Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics, MIT-IBM Watson AI Lab (mitibm.mit.edu), https://arxiv.org/pdf/1908.02591.pdf