Robust Graph Learning in the Presence of an Eavesdropper

Ahmed Almostafa Gashgash (atg74)

Adviser: Dr. Kirstin Petersen

Project Report

May 20th 2020

Abstract

Alongside text, images, time series and numerical data, learning from graphs has many important and interesting real world applications. Machine and Deep Learning approaches for generating graph embeddings are an effective method to represent graph data in a low dimensional space. However, these embeddings do not account for the privacy of the original graph. We consider the scenario where we aim to fool an eavesdropper who has access to the embeddings, presenting a security constraint. This adversary uses the embeddings to learn useful graph structure information. In this work we present an adversarially trained Graph Convolutional Network for the tasks of link prediction and clustering. We iteratively train our model to minimize our initial task loss while simultaneously maximize the adversaries loss. We obtain relatively similar results to previous work for link prediction, and outperform DeepWalk methods and Graph Autoencoders for clustering while ensuring privacy.

1 Introduction

Many real world systems such as social networks, recommendation systems and biological protein networks can be modelled by graphs. These graphs hold important information about the structure of the system. In social networks links between nodes could represent friendship or a certain connection between two agents, these links could further display a macro structure in the network such as the presence of clusters or communities. Also, inference of link presence in a social network could be used for suggesting friends [1]. In electronic commerce, a recommendation system could exploit the interaction between products and customers to influence future recommendations. In biology, molecule interactions are modeled by graphs and identifying their activity is crucial for drug discovery [2]. In citation networks, papers represent nodes and links between nodes refer to citations. Clusters within a citation network represent different academic categories. These examples display that graphs are not only useful data structures for storing and modelling data, but its also useful for modern Machine Learning tasks. Machine Learning aims to discover new patterns in data, classify data, and make predictions. Besides text, images, time series and numerical data, there is great interest in extending Machine and Deep Learning approaches to graph structured feature data [3] [4].

The main challenge with graph data, from a machine learning perspective, is that encoding this high-dimensional, non-Euclidean information about graphs into a feature vector is a non-trivial task. There are two streams of work to extracting feature vectors from graphs. Traditional approaches rely on graph statistics such as degree coefficients or clustering coefficients, kernel functions or hand crafted feature extractors [5]. The downside of these methods is that they cannot adapt during the learning process, and designing them is a tedious and expensive process. The other line of work seeks to learn representations that encode structural information about graphs. This lies under the theme of *Representation Learning*. The goal here is to learn a mapping function that embeds nodes or a graph into a lower dimensional vector space. By doing so, this lower dimensional embedding would reflect the structure of the original graph and could be used for downstream tasks, such as link prediction and node clustering. The major difference between representation learning and previous traditional methods, is that feature extraction is considered a machine learning task itsself instead of just a data pre-processing step.

After optimizing the mapping function, the learned embeddings hold key features about the structure of the original graph. For example, in a static Facebook social network, the embeddings could hold private information about friendship links between users, or communities to which a user belongs to. In a Wells Fargo transaction network, the embeddings could be used to infer future transactions, or detect fraudulent transactions. In these scenarios and many others where an agents privacy is important, it is crucial that the graph embedding could not be used by adversaries to infer any key structural information about the original graph. Recently many machine learning tasks are performed in a distributed manner. A central server with high computing capabilities would be used for computationally expensive tasks, and then distributes or communicates its learned embeddings to devices (i.e. mobile phones) for them to perform down stream learning tasks. In this scenario, a one point attack in the communication link, or access to the embedding storage would leave information about

the graph structure vulnerable, eliminating user privacy. This scenario of adversarial attacks on graph networks differ from traditional work where adversaries try to corrupt embeddings or fool the encoder [6]. It is more similar to the Wiretap Channel scenario in information theory, where the adversary eave-drops on embeddings with the goal of recovering crucial information about the original graph. It is inspired by the work of Fritschek [7] that uses deep learning for the wiretap channel. Other similar work uses adversarial training to enforce the embedding to follow a predefined probability distribution [8]. In this work we present a learning model that generates secure graph embeddings in the presence of an eavesdropper for the tasks of link prediction and node clustering.

2 Related Work

In this section, various node embedding techniques will be presented. A very popular approach uses the statistics of **random walks** to learn graph embeddings. The idea is to optimize the node embeddings such that nodes which tend to co-occur on short random walks over the graph have similar embeddings. These random walk methods employ a flexible, and stochastic measure of node similarity that has been successful in many settings. Two popular algorithms that rely on random walks are DeepWalk and node2vec [9]. In these two approaches, the idea is to learn embedding $\mathbf{z}$ such that for node i, j:

$$\begin{aligned} \mathrm{DEC}\left(\mathbf{z}_i, \mathbf{z}_j\right) &\triangleq \frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{v_k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}} \\ &\approx p_{\mathcal{G},T}\left(v_j | v_i\right) \end{aligned}$$

where $p_{\mathcal{G},T}\left(v_j|v_i\right)$ denotes the probability of visiting node v_j starting from v_i on a T length random walk.

The problem with random walk approaches is that they fail to leverage node attributes during encoding. In many graphs nodes have attribute information (e.g., user information on Facebook). This information is useful in determining the role or position of the node in the graph. Also, these methods are inherently transductive [10], as in they only generate embeddings for nodes that were present during training and cannot generalize to unseen nodes. This is highly problematic for large graphs that cannot be fully utilized during training, or evolving graphs such as social networks, or for tasks that require generalization after training.

Recent node embedding approaches that leverage deep learning address these issues by designing encoders that rely on a node's local neighborhood and not necessarily the whole graph. A popular method is GraphSAGE (Graph SAmple and aggreGatE), which is a general inductive learning framework for node embeddings. Different to previous transductive approaches, the main idea of GraphSAGE is to learn functions that generate embeddings for a node, instead of learning individual embeddings. Algorithm 1 in [10] describes a forward propagation algorithm that generates embeddings by aggregating feature information from a node's local neighborhood. The weight matrix and the paramateres of the aggregator function in the algorithm are learned by optimizing a graph based loss function using stochastic

gradient descent. The loss function is as follows:

$$J_{\mathcal{G}}\left(\mathbf{z}_u\right) = -\log\left(\sigma\left(\mathbf{z}_u^{\top}\mathbf{z}_v\right)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma\left(-\mathbf{z}_u^{\top}\mathbf{z}_{v_n}\right)\right)$$

where $\mathbf{z}_u, \forall u \in V$ is the node embedding, σ is the sigmoid function, P_n is a negative sampling distribution, and Q defines the number of negative samples. This loss function h pushes nearby nodes to have similar representations in the embedding space and separates distinct nodes. The embedding is generated from neighborhood information rather than training a unique embedding for each node.

The paper also proposes and compares three candidate aggregator functions, which should be symmetric (a permutation of the inputs should not change the result). The first proposed aggregator function is the **Mean Aggregator:**

$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W} \cdot \operatorname{MEAN}\left(\left\{\mathbf{h}_v^{k-1}\right\} \cup \left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right)\right)$$

where the mean operator simply takes the elementwise mean of the vectors. This is close to a linear approximation of a localized spectral convolution. They also propose an **LSTM Aggregator** based on a previous work architecture. These have the advantage of larger expressive capability but are not inherently symmetric therefore making it more complex. The final aggregator considered is the **Pooling Aggregator:**

$$\text{AGGREGATE}_k^{\text{pool}} = \max\left(\left\{\sigma\left(\mathbf{W}_{\text{pool}}\, \mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\right\}\right)$$

here each neighbor's vector is independently fed through a fully-connected neural network; following this transformation, an element wise max-pooling operation is applied to aggregate information across the neighborhood.

In [11] the authors introduce a linear model motivated from a first-order approximation of spectral graph convolutions. Based on the propagation rule, they show the graph convolutional network model is fast and scalable for semi-supervised node classification in a graph. This model will be the basis of our work. The Graph Convolutional Network proposed in this paper will be used for generating node embeddings for our pre-stated tasks. In the following sections we will describe the model in more detail.

3 Problem Setup

3.1 Model Overview

In our setting we are given a graph $\mathbf{G} = \{V, E, X\}$ where $V \in \mathbb{R}^n$ is the set of nodes in the graph, E represents the edges between nodes, and $X \in \mathbb{R}^{n \times f}$ are the features of each node, where f is the feature vector length. The graph could be represented by the pair (A, X) where A is the adjacency matrix. Given n nodes, the matrix $A \in \mathbb{R}^{n \times n}$ is a binary matrix with $a_{ij} = 1$ if there exists an edge between v_i and v_j, hence its a symmetric matrix. The goal is to learn an encoder to generate embedding $\mathbf{Z} = \mathbf{ENC(A,X)}$ such that for any task: $\mathcal{L}_{dec}\mathbf{(Z,A)}$ is minimized, whilst $\mathcal{L}_{adv}\mathbf{(Z,A)}$ is maximized. Where $\mathbf{Z} \in \mathbb{R}^{n \times m}$ is the embedding of size

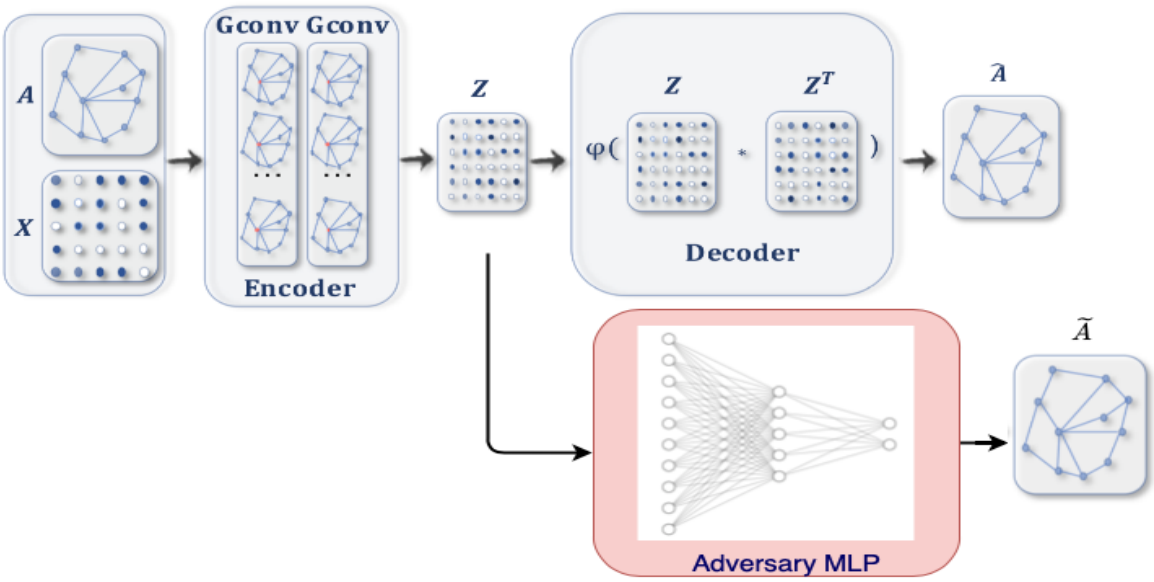


Figure 1: GAE for network embedding with an Adversary Eavesdropper model. Training iterates over the two branches.

m, and $\mathcal{L}$ is the corresponding loss function for the decoder and the adversary accordingly. For the encoder we will use a Graph Convolutional Network (GCN) described in detail in the following section. For the decoder we will use a sigmoid layer for recovering the graph structure. More specifically the probability that a link is present is given by:

$$p\left(\hat{\mathbf{A}}_{ij} = 1 | \mathbf{z}_i, \mathbf{z}_j\right) = \text{sigmoid}\left(\mathbf{z}_i^\top, \mathbf{z}_j\right)$$

.

The adversary is modeled by a two layer neural network of size 32 neurons each. It samples entries from the embedding and aims to learn a mapping to reconstruct useful graph information for the given task. A schematic of our model can be seen in Figure 1.

3.2 Graph Convolutional Neural Networks

The graph convolutional network aims to embed a graph into a lower dimensional space. The propagation rule in [11] is well-motivated from a $K-$localized approximation of spectral graph convolution. Specifically, for the spectral convolution on graph of a signal $x \in \mathbb{R}^N$, the filter is approximated using Chebyshev expansion:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\Lambda})$$

$\widetilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I$ is the rescaled eigenvalue matrix of the graph Laplacian, and $T_k(x)$ is the Chebyshev polynominal with $T_0(x) = 1$ and $T_1(x) = x$. Thus, the spectral graph convolution

is approximated as:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\widetilde{L})x,$$

with $\widetilde{L} = \frac{2}{\lambda_{max}} L - I$. This approximation can only capture the $K-$order neighbors of the central node. Nevertheless, it reduces the complexity from $O(N^2)$ to $O(|E|)$, because it eliminates the multiplication of eigenvector matrix. Besides, this approximation also does not require the expensive eigenvector decomposition.

The authors further approximate the largest eigenvalue as 2, and fix K as 1. And in order to address overfitting and reduce operation count, a single free parameter θ is adopted to represent θ'_0 and $-\theta'_1$. After those steps, the spectral graph convolution is further estimated as:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I)x = \theta'_0 x - \theta'_1 D^{-1/2} A D^{-1/2} x \approx \theta(I + D^{-1/2} A D^{-1/2})x,$$

where A and D are adjacency and degree matrix respectively. And finally, in order to address the instabilities, a renormalization is performed to replace $I + D^{-1/2} A D^{-1/2}$ by $\widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2}$, with $\widetilde{A} = A + I$ and $\widetilde{D}_{ii} = D_{ii} + 1$. Thus, a layer-wise propagation rule is define as:

$$Z = \widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2} X W,$$

where W is the filter parameter matrix.

Furthermore, motivated by the idea that stacking multiple convolution layers can extract information not limited by the items of the Chebyshev expansion, the authors build a Graph Convolution Network (GCN) which adopts this propagation rule for each layer. In the forwarding, ReLU activation function is applied for the first hidden layer outputs, and a linear function is applied at the end. The same graph structure is shared between layers. This GCN model can be trained to perform the semi-supervised node classification. To this end, the cross-entropy error is measured for all the labelled nodes, and the weights are trained using batch gradient descent. The encoder can be constructed as follows:

$$\begin{gathered} f\left(\mathbf{Z}^{(l)}, \mathbf{A} | \mathbf{W}^{(l)}\right) = \phi\left(\widetilde{\mathbf{D}}^{-\frac{1}{2}} \widetilde{\mathbf{A}} \widetilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{Z}^{(l)} \mathbf{W}^{(l)}\right) \\ \mathbf{Z}^{(1)} = f_{\text{Relu}}\left(\mathbf{X}, \mathbf{A} | \mathbf{W}^{(0)}\right) \\ \mathbf{Z}^{(2)} = f_{\text{linear}}\left(\mathbf{Z}^{(1)}, \mathbf{A} | \mathbf{W}^{(1)}\right) \end{gathered}$$

3.3 Optimizer

The cross-entropy error is measured for all the labelled nodes, and the weights are trained using batch gradient descent. If we parametrize the encoder by H and the adversary by $\mathcal{Q}$ The loss function implemented is as follows:

$$\min_{\mathcal{H}} \max_{\mathcal{Q}} \mathbb{E}_{\mathbf{z} \sim p_z}[\log \mathcal{Q}(\mathbf{Z})] + \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}[\log(1 - \mathcal{Q}(\mathcal{H}(\mathbf{X}, \mathbf{A})))]$$

Notice that when optimizing the adversary using the above equation, the adversary has no influence on the encoder, and is oblivious to the encoders weights. On the other hand when

optimizing the encoder, the state of the adversary is considered and secure embeddings can be generated. The idea is after generating the embeddings, the discriminator samples a set from the embeddings and trains its network for a certain number of epochs, in our case 5 epochs. After the discriminator reduces its loss, the encoder is trained for another epoch and tries to confuse the adversary. This is done iteratively.

3.4 Data

Table 1: Statistics of datasets used in our experiments.

Dataset	Type	Task	Nodes	Edges	Classes	Features
Cora	Citation network	Transductive	2,708	5,429	7	1,433
Citeseer	Citation network	Transductive	3,327	4,732	6	3,703

In order to evaluate our model, we choose two real-world datasets of different sizes. Specifically we will consider the citation network datasets: Cora, and Citeseer to perform transductive node classification and link prediction tasks. The statistics of each dataset are summarized in Table 1. Each node in the graph refers to a document/paper while the edges are citation links. Each dataset contains sparse bag-of-words feature vectors for each node and a list of citation links between documents. These citation links are represented in a binary symmetric adjacency matrix A. The feature vectors for each document are stored in a separate feature matrix X. We generate a diagram of the CORA dataset as can be seen in Figure 2.

Figure 2: The CORA dataset represented as a digraph showing citation relationships between papers

For link prediction, the citation edges for each dataset is split as follows: 5% validation, 10% test and 85% training. More importantly, [12] has shown some pitfalls of these datasets, in which choosing different training/test splits will largely effect the link prediction or the node classification accuracy score. As suggested by [12], we randomly choose different training and test sets for 5 times, and report both the average classification accuracy as well as standard deviation in the evaluation section.

4 Evaluation

We evaluate our model for two popular graph learning tasks: link prediction and node clustering. In link prediction the goal is to predict whether a link is present between two nodes. In node clustering the goal is to correctly classify to which category a node belongs.

4.1 Link Prediction Results

We compare our results with DeepWalk [13], and Variational Graph Autoencoders [14] as our baselines. We train for 200 epochs with learning rate 0.001 and an Adam optimizer. We evaluate the performance using the AUC score (the area under a receiver operating characteristic curve) and average precision (AP) similar to [14]. We average 5 experiments and report the mean and deviation. Scores are reported in Table 2.

Table 2: Link prediction results for Cora and Citeseer

Method	Cora		Citeseer	
	AUC	AP	AUC	AP
DeepWalk [13]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01
GAE [14]	83.1 ± 0.01	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02
GCN+EVE	82.3 ± 0.04	83.5 ± 0.05	79.3 ± 0.05	84.5 ± 0.06

The results are also presented in Figure 3. These curves show the scores vs epochs and provide a visualization of the models performance with training steps. One interesting feature we notice is the shape of the validation and test curves. We notice an early rise, a dip and then another rise in performance. This is expected, since at an early stage of training the embeddings are not informative, therefore the eavesdropper is unable to learn. It also requires some time for the adversary to tune his parameters. As training progresses, the adversary is more competent and is able to optimize his loss function, in this stage the encoder begins to adjust its weights to fool the adversary and trades security for precision, this explains the dip in the curve. As training progresses the performance of our model begins to gradually improve as seen in Figure 3. Our results are close to but do not beat the baseline presented. This is explained since we trade off accuracy and security.

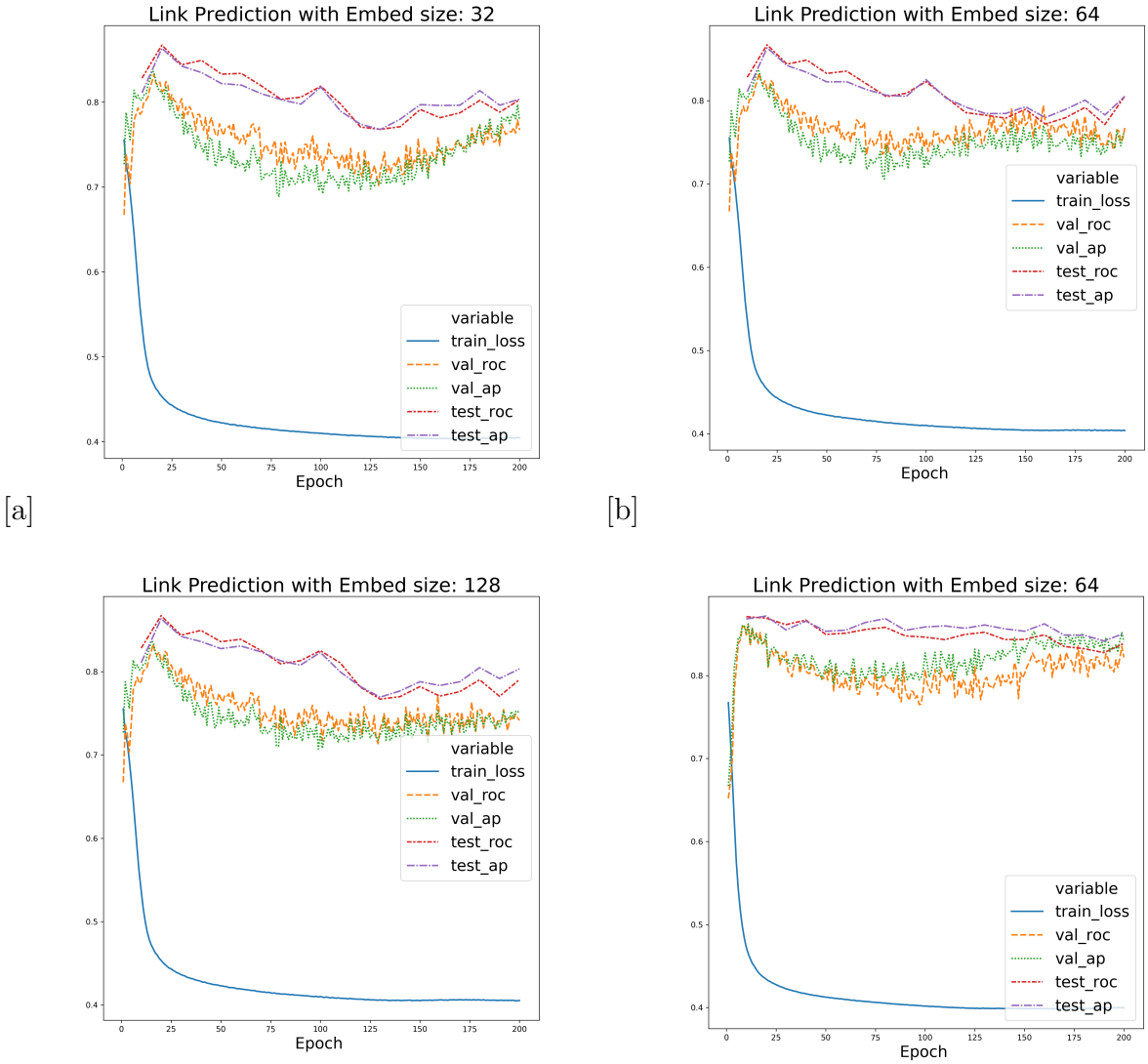


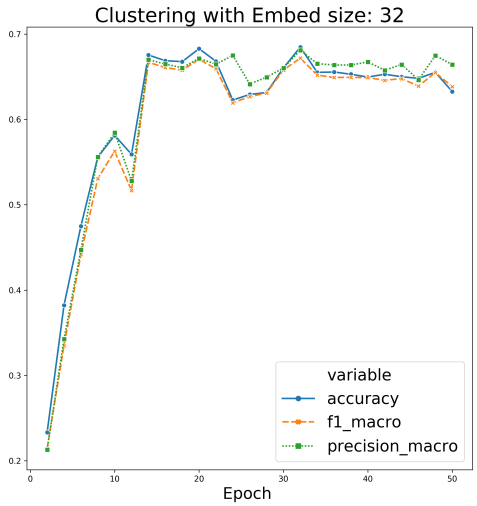
Figure 3: (a) Cora link prediction with emd size 32 (b) Cora link prediction with emd size 64 (c) Cora link prediction with emd size 128 (d) Citeseer link prediction with emd size 64

4.2 Clustering Results

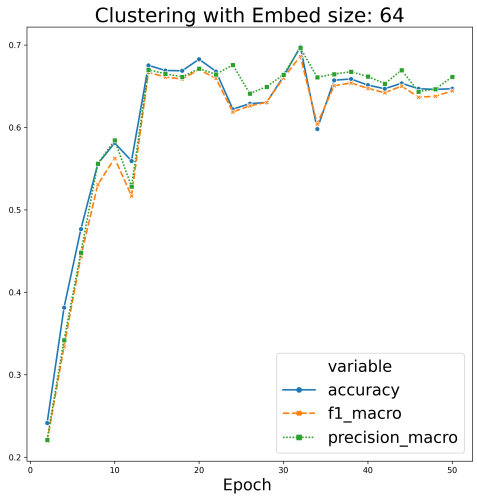
For clustering we first learn an embedding and then apply a K-means algorithm classify each node. Again we compare to DeepWalk and GAE. The metrics used to evaluate our model are the F1 score, accuracy and precision. The results are reported in Table 3. And the training curves are presented in Figure 4.

Table 3: Clustering results for Cora and Citeseer

Cora	Accuracy	Precision	F1	Citeseer	Accuracy	Precision	F1
DeepWalk	0.484	0.361	0.392	DeepWalk	0.337	0.248	0.270
GAE	0.596	0.596	0.595	GAE	0.408	0.418	0.372
GCN+EVE	0.643	0.652	0.613	GCN+EVE	0.560	0.541	0.482

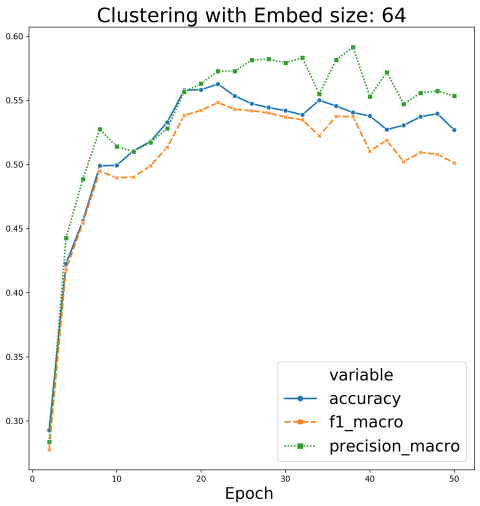


[a]



[b]

[c]



[d]

Figure 4: (a) Cora clustering with emd size 32 (b) Cora clustering with emd size 64 (c) Cora clustering with emd size 128 (d) Citeseer clustering with emd size 64

For this task our model outperforms the baseline for all metrics. Its also surprising that the curves do not show the presence of the adversary. We also notice that changing the embedding size does not alter the performance drastically. It could be different if a much

smaller size is used. Since these networks are considered small compared to other real world networks, the performance wouldn't greatly increase with a larger embedding size since the low level features to be learned are limited. It's important to to point out that a larger embedding space could result in overfitting, whilst a smaller space would hinder the models accuracy. The embedding size is considered. hyperparameter in our model.

5 Conclusion

In this work we present an adversarially trained Graph Convolutional Network for the tasks of link prediction and clustering. We consider the scenario where an eavesdropper has access to the generated embeddings and presents a security constraint. We iteratively train our model to minimize our initial task loss while simultaneously maximize the adversaries loss. We report our results for both tasks, and while we do not outperform previous work for link prediction, our clustering results beats DeepWalk methods and Graph Autoencoders.

6 Future Work

A major improvement for this work is to incorporate a security metric for our analysis. Including a security metric when reporting our results allows for a fair comparison with other work and could influence other researchers to include security measures in their models. Another direction is to give the user more control of the trade-off between security and accuracy before training the model. This includes adjusting the loss function by adding parameters that control this trade-off. Citation Networks are a very limited domain to evaluate our model, its interesting to see how it performs with real world networks such as protein-protein networks, social networks or dynamic settings such as Robot Swarms placement networks. More experiments could be further conducted for more accurate and insightful results, for example running at least 100 experiments for link prediction and reporting smaller variance.

7 Acknowledgments

Part of the code used for programming our model was borrowed from https://github.com/tkipf/gae [14]. New code was written for the optimizer and the model.

7.1 Individual Contributions:

Ahmed: Developed the initial idea of the project. Worked on data engineering, building the model, optimizing, training and testing using Tensorflow in Python. Also wrote this final report.

Kevin: Worked on visualization of the CORA dataset, analysis and plotting of the training curves using Seaborn. Also, worked on writing the initial project proposal.

References

[1] Lars Backstrom and Jure Leskovec. Supervised random walks: Predicting and recommending links in social networks. *CoRR*, abs/1011.4071, 2010.

[2] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *CoRR*, abs/1509.09292, 2015.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, 2016.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[5] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, page 556–559, New York, NY, USA, 2003. Association for Computing Machinery.

[6] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, Jul 2018.

[7] Rick Fritschek, Rafael F. Schaefer, and Gerhard Wunder. Deep learning for the gaussian wiretap channel, 2018.

[8] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding, 2018.

[9] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

[11] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016.

[12] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation, 2018.

[13] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.

[14] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.