TechRxiv
Powered by IEEE

On Edge Reweighting for Link Prediction with Graph Auto-Encoders

This paper was downloaded from TechRxiv (https://www.techrxiv.org).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

01-12-2020 / 17-12-2020

CITATION

Huang, William R. (2020): On Edge Reweighting for Link Prediction with Graph Auto-Encoders. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.13312610.v2

DOI

10.36227/techrxiv.13312610.v2

On Edge Reweighting for Link Prediction with Graph Auto-Encoders

William R. Huang
Research Scholar, R.I.S Institute, Taipei, Taiwan ROC

Abstract
During the last five years, graph auto-encoders became popular unsupervised methods, based on graph neural networks, to learn a node embedding from a graph. Researchers train graph auto-encoders by optimizing reconstruction losses that are computed from the connected node pairs (the edges) and non-connected node pairs of the graph. Many graphs being sparse, researchers often positively reweight the edges in these reconstruction losses. In this paper, we report an analysis of the effect of edge reweighting on the node embedding. We show that, on a link prediction problem, results are quite insensitive to edge reweighting, with the exception of very unbalanced reconstruction losses. We also discuss whether training models from perfectly balanced reconstruction losses is optimal or suboptimal, in terms of average scores and of standard deviations.

Keywords: *Graph, Network, Unsupervised Learning, Auto-Encoders, Graph Reconstruction, Edge Reweighting*

I. INTRODUCTION

The research activity related to the development of *machine learning* methods for *graph* (a.k.a. *network*) data has grown at a fast pace over the past few years [1]. It became one the most active and intriguing sub-areas of *deep learning* [2, 3, 4]. Among others, several researchers constructed different *graph neural network* [3, 5, 6] architectures to learn *node embeddings* [1, 2, 3, 4]. A node embedding is a vector space learnt from a graph neural network (in general), in which the nodes from a given graph are represented by some vectors. The similar nodes in the graph will have close vectors in the space. Working with a node embedding instead of working with a graph linking the nodes can be useful to solve machine learning problems involving the nodes [1, 2, 3, 4, 6, 7]. However, a majority of these graph neural networks must be trained in a *supervised* way. Indeed, researchers will often update the parameters of these neural networks by minimizing a loss that involves labels on each node or on a subset of nodes [1, 3, 6]. This is limiting, as such labels are sometimes unavailable.

During the last five years, *graph auto-encoders* [7] became efficient methods extending graph neural networks to learn a node embedding but in as *unsupervised* methods. Instead of using node labels, graph auto-encoders optimize a *reconstruction loss* that must be computed from the connected node pairs (the edges) and non-connected node pairs of the graph. The loss will decrease if the graph auto-encoder can correctly predict, using the node embedding, which node pairs are connected and not connected in the original graph. In other words, we assess the quality of the embedding by check if, starting from this space, it is possible to output a reconstructed graph that is quite similar to the true data. Graph auto-encoders (and their derivatives, as *graph variational auto-encoders* [7]) have been recently used to deal with a wide range of research problems. Some famous examples are: link prediction [7, 8, 9, 10, 11], node clustering [12, 13, 14] and generating some small graphs such as molecules [15, 16, 17]. Some papers also showed that graph auto-encoders give interesting results for very large graphs with several millions of nodes [18, 19].

Many graphs being sparse, researchers almost always *reweight the edges* in the reconstruction loss, with respect to the non-connected node pairs which are more numerous. Most codes simply set a positive *edge reweighting scalar parameter* in the reconstruction loss. A deep experimental analysis of the effect of this reweighing on the node embedding is missing. In this paper, we conduct and report the results of such an analysis. For experiments, we focus on the usage of the node embedding vectors to address the link prediction task [20], because this is the most common task to assess the quality of graph auto-encoders. Our analysis shows that the link prediction results are quite insensitive to unbalanced reconstruction losses, with the exception of extreme cases. Our analysis also tends to show the interest of keeping a quite balanced loss as well as the interest of slightly overweighting edges with respect to non-connected node pairs, on some of our graphs.

II. METHODS

We have an undirected graph $G = (V, E)$ with $|V|$ nodes and $|E|$ edges. We summarize G by the adjacency matrix A of dimension $|V| \times |V|$ which is defined as: $A_{uv} = 1$ if the node pair (u, v) is connected by an edge ($(u, v) \in E$) and $A_{uv} = 0$ otherwise. We want to give to each node u a vector z_u of dimension $d < |V|$ in a node embedding space $\mathcal{Z}$. Also, in next paragraphs, Z will be the matrix of size $|V| \times d$ inside which the row n°v will correspond to the vector z_u.

A graph auto-encoder [7] is an unsupervised model with an *encoder* and a *decoder*. The *encoder* is a parametrized function ; in modern research this function will be a graph neural network [3, 5, 6]. Its input is A and its output is Z. In our experiments, we will follow the first works of Kipf and Welling [6, 7] and our encoder will thus be a *2-layer graph convolutional network*:

$$Z = GCN(A) = \tilde{A}\, ReLU\left(\tilde{A}\, W_0\right) W_1.$$

Here:

- $\tilde{A}$ the normalized version of the adjacency : $\tilde{A} = D^{-1/2}(A + I_{|V|})D^{-1/2}$, with D the degree matrix of $A + I_{|V|}$ ($I_{|V|}$ is the identity matrix of dimension $|V| \times |V|$),
- $ReLU(x) = \max(x, 0)$,
- W_0 an $n \times d_{hidden}$ weight matrix and W_1 another $d_{hidden} \times d$ weight matrix (to learn).

Kipf and Welling [7] set $d_{hidden} = 32$ and $d = 16$, and we will choose similar dimensions in our experiments. Once the matrix Z (the node embedding vectors) is computed, the *decoder* will reconstruct an approximate version of A, named $\hat{A}$ and of dimension $|V| \times |V|$, as follows:

$$\forall\, (u, v) \in V \times V, \quad \hat{A}_{uv} = \sigma(z_u^T z_v) = \frac{1}{1 + e^{-z_u^T z_v}} = \frac{1}{1 + e^{-\sum_{i=1}^{d} z_u[i] z_v[i]}} \in\,]\, 0, 1\, [.$$

Intuitively, the node embedding vectors in Z will be of "good quality" if the reconstructed matrix $\hat{A}$ is equal or very close to the true initial data A. So, researchers train the graph auto-encoder by gradient descent to minimize a *reconstruction loss* [7]:

$$L_{reconstruction}\left(A, \hat{A}\right) = \frac{1}{|V|^2}\, [\, \gamma \times \sum_{(u,v) \in E} L(A_{uv}, \hat{A}_{uv}) + \sum_{(u,v) \in V \times V \setminus E} L(A_{uv}, \hat{A}_{uv})\,].$$

Kipf and Welling [7] set:

$$L\left(A_{uv}, \hat{A}_{uv}\right) = -A_{uv} \log\left(\hat{A}_{uv}\right) - (\, 1 - A_{uv}) \log\left(1 - \hat{A}_{uv}\right).$$

We note the presence of an *edge reweighting scalar parameter* γ (usually $\gamma > 1$) in $L_{reconstruction}\left(A, \hat{A}\right)$. Indeed, many graphs being sparse, researchers felt the need to positively reweight the edges in the reconstruction losses, with respect to the non-connected node pairs which are more numerous ($|V \times V \setminus E|$ vs $|E|$). For instance, Kipf and Welling [7] set $\gamma = \frac{|V \times V \setminus E|}{|E|}$ and this choice ensures that the "positive" (edges) and "negative" (non-connected node pairs) parts of the reconstruction loss have the same relative importance.

However, the choice of γ and the effect of edge reweighting on the node embedding has never been deeply studied. Existing research set the value of γ in one line of code, without further study. In the next two sections of the paper, we conduct and report the empirical results of such a study, focusing on link prediction applications. Our objective is to assess the effect of setting:

$$\gamma = \alpha \times \frac{|V \times V \setminus E|}{|E|},$$

for different values of the scalar parameter $\alpha \in \mathbb{R}^+$. Setting $\alpha = 1$ leads to an artificially balanced reconstruction loss (as before), while setting $\alpha < 1$ will underweight edges with respect to non-connected node pairs and setting $\alpha > 1$ will overweight edges with respect to non-connected node pairs.

III. EXPERIMENTAL DESIGN

For evaluation, we follow the experimental procedure of previous papers [7, 8, 9, 10, 11, 18, 19] and do *link prediction*. The goal is to assess, for different values of α, our performance at predicting if two nodes u and v from the original graph are connected by an edge (u, v) or not, only by using the learnt node embedding vectors z_u and z_v and the associated reconstructed cell $\hat{A}_{uv}$. We will report results on the three citation networks *Cora, Citeseer* and *Pubmed* (we refer to [6] for description), whose statistics are available in Table 1. These three graphs are very relevant for our study, because they are commonly used, and because they are very sparse and thus will all require the tuning of an edge reweighting scalar parameter γ to return good results.

Table 1: Statistics of the Cora, Citeseer and Pubmed citation networks

Graph	Number of nodes	Number of edges
Cora	3 327	4 732
Citeseer	2 708	5 429
Pubmed	19 717	44 338

More precisely we will train several graph auto-encoders on *some masked versions of the original graph data*, with only 85% of edges. Among the missing edges, 5% are put in a validation set and 10% and put in a test set, together with the same number of non-connected pairs of nodes (selected randomly). These numbers are on par with previous papers. Are we able to retrieve the missing edges in the test set? This is actually a *classification* problem, that we will evaluate using the following two metrics:

- AUC: Area Under the receiver operating characteristic Curve [21].
- AP: Average Precision score [22].

We chose to train all the models using the ADAM algorithm [23] with a learning rate of 0.01, for 200 epochs of training and with $d_{hidden} = 32$ and $d = 16$ (as explained in II). For the Pubmed graph, which is larger than Cora and Citeseer, we used the FastGAE code from [19] for faster evaluations.

IV. RESULTS

Table 2 shows our results on the three graphs Cora, Citeseer and Pubmed. All the AUC and AP scores are in percentage and are averaged over 20 trainings of the graph auto-encoder model, and we also present the corresponding standard deviations over all these different trainings (to account for the volatility due to the randomness in edge masking). We tested a wide range of values for the parameter α. As explained in II, setting $\alpha = 1$ is equivalent to a balanced reconstruction loss, whereas setting $\alpha < 1$ will underweight edges with respect to non-connected node pairs and setting $\alpha > 1$ will overweight edges with respect to non-connected node pairs.

Table 2: Link prediction results (on test sets) for the Cora, Citeseer and Pubmed citation networks

Value of α	Link prediction results (in percentage)					
	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
0.001	52.2 ± 2.2	52.3 ± 2.2	51.9 ± 2.8	52.1 ± 2.8	55.3 ± 2.8	55.5 ± 3.2
0.01	84.1 ± 1.3	86.6 ± 1.2	78.0 ± 2.2	81.4 ± 2.3	73.8 ± 2.6	77.2 ± 3.2
0.1	84.5 ± 0.9	88.2 ± 0.9	78.1 ± 1.9	83.7 ± 1.5	82.4 ± 0.7	86.2 ± 0.5
0.25	84.8 ± 0.8	88.2 ± 0.8	78.1 ± 1.5	83.7 ± 1.5	83.3 ± 0.6	86.9 ± 0.4
0.5	84.8 ± 0.8	88.3 ± 0.8	78.2 ± 1.6	83.7 ± 1.3	83.7 ± 0.5	87.2 ± 0.3
0.75	84.9 ± 0.8	88.3 ± 0.8	78.2 ± 1.4	83.8 ± 1.2	83.7 ± 0.4	87.1 ± 0.3
0.90	84.8 ± 0.8	88.1 ± 0.6	78.3 ± 1.6	83.8 ± 1.2	83.8 ± 0.3	87.1 ± 0.3
0.95	84.8 ± 0.7	88.2 ± 0.6	78.3 ± 1.2	83.8 ± 1.0	83.8 ± 0.5	87.1 ± 0.3
1	84.9 ± 0.7	**88.4 ± 0.6**	78.4 ± 1.2	83.8 ± 1.1	**83.9 ± 0.4**	**87.4 ± 0.3**
1.05	84.9 ± 0.7	88.3 ± 0.6	**78.8 ± 1.2**	**83.9 ± 1.2**	83.8 ± 0.5	87.2 ± 0.3
1.10	84.8 ± 0.8	**88.4 ± 0.5**	78.2 ± 1.5	**83.9 ± 1.2**	83.7 ± 0.4	86.8 ± 0.3
1.25	**85.0 ± 0.8**	**88.4 ± 0.7**	78.1 ± 1.3	83.8 ± 1.6	83.6 ± 0.5	86.8 ± 0.3
1.50	84.9 ± 0.9	88.3 ± 0.8	77.5 ± 1.2	83.2 ± 1.5	83.4 ± 0.4	86.7 ± 0.3
1.75	84.7 ± 1.2	88.3 ± 0.9	77.5 ± 1.4	82.7 ± 1.3	83.4 ± 0.4	86.7 ± 0.3
2.0	84.1 ± 1.3	87.6 ± 1.3	77.5 ± 1.6	82.5 ± 1.3	83.0 ± 0.4	86.5 ± 0.3
5.0	83.6 ± 1.4	86.8 ± 1.2	77.4 ± 2.0	82.5 ± 1.6	82.9 ± 0.4	86.2 ± 0.3
10.	81.4 ± 1.4	85.0 ± 1.3	77.1 ± 2.2	81.9 ± 2.1	81.9 ± 0.5	85.2 ± 0.4
100	75.1 ± 2.1	77.4 ± 2.2	70.4 ± 2.3	74.1 ± 2.1	71.1 ± 0.6	75.6 ± 0.6
1000	66.5 ± 2.2	70.2 ± 2.1	65.5 ± 2.3	70.0 ± 2.0	70.1 ± 0.9	75.5 ± 0.7

Foremost, we see in Table 2 that the Area Under the ROC Curve and the Average Precision link prediction scores on the test sets are quite *insensitive* to the choice of α in the graph auto-encoder reconstruction loss, with the exception of *extreme values* ($\alpha = 0.001, 0.01, 100$ or $1\,000$). For all other values of α we reach scores that are quite close to the scores of the balanced reconstruction loss with $\alpha = 1$. Another result from our experimental analysis is that: fine-tuning α (particularly to oversample the edges with respect to the non-connected node pairs) can sometimes *very slightly* improve the results. In Table 2, for the Cora graph and for the Citeseer graph, choosing $\alpha = 1.25$ and $\alpha = 1.05$, respectively, is optimal. Ultimately, we know that the standard deviations of our studies are quite large and that differences are not necessarily expressive ; yet, we see that selecting α around 1 decreases the scores volatilities with respect to very unbalanced reconstruction losses. Future works on larger graphs could be necessary to really confirm our results on the advantage of $\alpha > 1$.

V. CONCLUSION

As many graphs are sparse, researchers often positively reweight the edges of their reconstruction losses when training graph auto-encoders models, with graph neural network encoders and inner product decoders. However, a deep experimental analysis of the effect of this reweighing on the model was missing. In this paper, we reported and commented the results of such an analysis. We focused on the usage of graph auto-encoders for link prediction on three popular citation networks. We showed that the link prediction performances are quite insensitive to unbalanced reconstruction losses, with the exception of extreme values. We also explained the potential interest of keeping a quite balanced loss as well as slightly overweighting edges with respect to non-connected node pairs, in terms of optimal scores and of reduced standard deviations. Future studies will try to confirm our results on variants of graph auto-encoders, such as graph variational auto-encoders (preliminary experiments are conclusive) as well as on different graph datasets.

VI. REFERENCES

[1]. Hamilton, W. L. (2020). Graph representation learning. Synthesis Lectures on A.I. and Machine Learning, 14(3), 1-159.
[2]. Hamilton, W. L., Ying R., Jure Leskovec. (2017). Representation learning on graphs: Methods and applications. IEEE Data Engineering Bulletin.
[3]. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems.
[4]. Zhang, Z., Cui, P., & Zhu, W. (2020). Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering.
[5]. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE Transactions on Neural Networks, 20(1), 61-80.
[6]. Kipf, T.N., Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. International Conference on Learning Representations.
[7]. Kipf, T.N., Welling, M. (2016) Variational Graph Auto-Encoders. NeurIPS Bayesian Deep Learning workshop.
[8]. Salha, G., Limnios, S., Hennequin, R., Tran, V. A., & Vazirgiannis, M. (2019). Gravity-inspired graph autoencoders for directed link prediction. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (pp. 589-598).
[9]. Shin, J., Kim, K., Park, D., Kim, S., & Kang, J. (2020). Bipartite Link Prediction by Intra-Class Connection Based Triadic Closure. IEEE Access, 8, 140194-140204.
[10]. Grover, A., Zweig, A., Ermon, S. (2019) Graphite: Iterative generative modeling of graphs. International Conf. on Machine Learning.
[11]. Salha, G., Hennequin, R., & Vazirgiannis, M. (2019). Keep it simple: Graph autoencoders without graph convolutional networks. arXiv preprint arXiv:1910.00942.
[12]. Wang, C., Pan, S., Long, G., Zhu, X., Jiang, J. (2017). Mgae: Marginalized graph autoencoder for graph clustering. ACM Conference on Information and Knowledge Management.
[13]. Liu, J. J. C. X., & Murata, T. (2020). Optimizing Variational Graph Autoencoder for Community Detection with Dual Optimization. Entropy, 22(2), 197.
[14]. Salha, G., Hennequin, R., & Vazirgiannis, M. (2020). Simple and effective graph autoencoders with one-hop linear models. arXiv preprint arXiv:2001.07614.
[15]. Jin, W., Barzilay, R., & Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. arXiv preprint arXiv:1802.04364.
[16]. Ma, T., Chen, J., & Xiao, C. (2018). Constrained generation of semantically valid graphs via regularizing variational autoencoders. In Advances in Neural Information Processing Systems 7113-7124.
[17]. Simonovsky, M., & Komodakis, N. (2018). Graphvae: Towards generation of small graphs using variational autoencoders. In International Conference on Artificial Neural Networks pp. 412-422.
[18]. Salha, G., Hennequin, R., Tran, V. A., & Vazirgiannis, M. (2019). A degeneracy framework for scalable graph autoencoders. arXiv preprint arXiv:1902.08813.
[19]. Salha, G., Hennequin, R., Remy, J. B., Moussallam, M., & Vazirgiannis, M. (2020). FastGAE: Fast, Scalable and Effective Graph Autoencoders with Stochastic Subgraph Decoding. arXiv preprint arXiv:2002.01910.
[20]. Liben-Nowell, D., & Kleinberg, J. (2007). The link-prediction problem for social networks. Journal of the American society for information science and technology, 58(7), 1019-1031.
[21]. AUC on Scikit Learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
[22]. AP on Scikit Learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html
[23]. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.