# A Simple Training Strategy for Graph Autoencoder

Yingfeng Wang
Middle Georgia State University
100 University Parkway
Macon, Georgia, USA
1-478-471-2477
yingfeng.wang@mga.edu

Biyun Xu
Beijing Kubao Technology Company
1089 Huihe South Street
Chaoyang, Beijing, China
86-17601230404
xu.biyun@foxmail.com

Myungjae Kwak
Middle Georgia State University
100 University Parkway
Macon, Georgia, USA
1-478-757-6682
myungjae.kwak@mga.edu

Xiaoqin Zeng
Hohai University
1 Xikang Road
Nanjing, Jiangsu, China
xzeng@hhu.edu.cn

## ABSTRACT
Graph autoencoder can map graph data into a low-dimensional space. It is a powerful graph embedding method applied in graph analytics to reduce the computational cost. The training algorithm of a graph autoencoder searches the weight setting for preserving most graph information of the graph data with reduced dimensionality. This paper presents a simple training strategy, which can improve the training performance without significantly increasing time complexity. This strategy can flexibly fit many existing training algorithms. The experimental results confirm the effectiveness of this strategy.

## CCS Concepts
**• Computing methodologies ➝ Machine learning ➝ Machine learning approaches ➝ Neural networks**

## Keywords
Keywords: Autoencoder; Graph Convolutional Network; Perturbation; Training Algorithm.

## 1. INTRODUCTION
Autoencoder is a neural network composed of encoder and decoder. Encoder converts the input data into an abstract representation, while decoder reconstructs the original input data from the output of encoder. Graph autoencoder is based on graph neural network, whose input data is graph information. Because graph autoencoder has shown great potential in dimensionality reduction, it has been drawing more and more attention in graph embedding [1].

Graph autoencoder embeds graph data based on matrix factorization [1, 16]. It aims to preserve the graph structure of the input matrix, e.g., adjacency matrix, in a low-dimensional space by matrix factorization [25]. The effectiveness of an autoencoder depend on the effectiveness of its training algorithm, which minimizes the loss between the input data and the reconstructed data. The traditional autoencoders are based on traditional neural networks. Their training algorithms have been studied for decades [8]. However, graph neural network is relatively new [17]. It motivates researchers to invest substantial efforts in developing training algorithms of graph neural network for addressing different issues [3, 7, 13, 23, 25, 26]. These efforts also benefit the training algorithm design for graph autoencoder.

Most existing graph autoencoders use deep models or convolutional networks. The deep learning method was first implemented in a graph autoencoder named SDNE [21]. This autoencoder focuses on the proximity of a simple graph composed of vertexes and edges. It uses the first order and second order of proximity to describe pairwise vertex proximity and pairwise neighborhood structure proximity, respectively. The training algorithm of SDNE is semi-supervised. Tu et al. also applied deep learning in their graph autoencoder DRNE [18]. This autoencoder uses multilayer perceptron (MLP), a feedforward neural network, while its whole model is recursive. DRNE uses Adam training algorithm [9]. Cao et al. applied deep learning in a denoising graph autoencoder called DNGR [2]. A denoising autoencoder used corrupted input in the training, while the expected output of decoder is the original input [19]. This training algorithm aims to enable the trained autoencoder to automatically filter out noise. Yu et al. used deep learning to develop an adversarially regularized autoencoder named NetRA [24]. Its adversarial training platform is composed of generator and discriminator [6]. Generator generates faked encoded representations, while discriminator attempts to distinguish faked representations from real output of encoder. This algorithm is good at reducing autoencoder learning failures caused by too much capacity of encoder and decoder. Graph convolutional network (GCN) [11] is also very popular in graph autoencoders. Kipf and Welling introduced a variational graph autoencoder (VGAE) and its non-probabilistic variant, GAE, based on a two-layer GCN [12]. The encoder of a variational autoencoder is a generative model, which learns the distribution of training samples [10]. Wang et al. presented a marginalized graph autoencoder (MGAE), which is also based on GCN [20]. The training algorithm of MGAE marginalizes the corrupted input and optimizes the autoencoder to reconstruct the original input. Based on VGAE and GAE, Pan et


*ICMLC 2020,* February 15–17, 2020, Shenzhen, China

ACM ISBN 978-1-4503-7642-6/20/02…$15.00
DOI: https://doi.org/10.1145/3383972.3383985

al. proposed an adversarially regularized autoencoder ARVGE and its variant ARGE [16].

The training algorithm is a key component of a graph autoencoder. Researchers have designed different training algorithms for above graph autoencoders. This paper proposes a simple training strategy, which can be applied in many existing training algorithms. Here, we focus on using this strategy in GCN based autoencoders.

The rest of this paper is organized as follows. Section 2 introduces the basic architecture of a GCN based autoencoder model. The details of the proposed training strategy are provided in section 3. Experimental results are presented in section 4. Section 5 concluded the paper.

## 2. THE GCN BASED AUTOENCODER MODEL

A graph autoencoder is composed of an encoder and a decoder. The upper part of Figure 1 is a diagram of a general graph autoencoder. The input graph data is encoded by the encoder. The output of encoder is the input of decoder. Decoder can reconstruct the original input graph data. Kipf and Welling proposed a GCN-based autoencoder model [12]. This diagram of this model is given in the lower part of Figure 1. The encoder in this model is a GCN. The input graph data can be represented by ($A$, $X$), where $A$ is the adjacency matrix, while $X$ is the node feature matrix. If the number of nodes is $n$, $A$ is a $n \times n$ matrix. It is worth noting that all diagonal elements of $A$ are set to one. If the number of features is $m$, $X$ is a $n \times m$ matrix. We can build the $n \times n$ degree matrix $D$ based on $A$ by the following expression.

$$D_{ij} = \begin{cases} \sum_{k=1}^{n} A_{ik}, if\ i = j \\ 0, if\ i \neq j \end{cases}, \quad (1)$$

where $1 \leq i, j \leq n$. Assume the GCN used in the encoder has $k$ layers. The output of the $l$th layer of GCN, represented by $Z^{(l)}$, can be computed by the following formula.

$$Z^{(l)} = f^{(l)}(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}Z^{(l-1)}W^{(l)}), \quad (2)$$

where $1 \leq l \leq k$, $Z^0 = X$, $f^{(l)}$ is the activation function of the $l$th layer, $W^{(l)}$ is the weight matrix of the $l$th layer, and $D^{-\frac{1}{2}}$ can be calculated by the following expression.

$$D_{ij}^{-\frac{1}{2}} = \begin{cases} D_{ii}^{-\frac{1}{2}}, if\ i = j \\ 0, if\ i \neq j \end{cases} \quad (3)$$

The training algorithm adjusts the weight parameters of all layers. In GAE VGAE, ARGE, and ARVGE, the two-layer GCN of the encoder uses the following Relu activation function for the first layer.

$$f_{Relu}(t) = \max(0, t) \quad (4)$$

And the following linear function is used for the second layer.

$$f_{linear}(t) = t \quad (5)$$

We represent the output of the encoder by $Z$. In GAE and ARGE,

$$Z = Z^{(2)}, \quad (6)$$

while in VGAE and ARVGE, the output of the encoder is calculated by the following expression.

$$Z = Z^{(2)} + N_{n \times r}(0, Exp(Z^{(2)})), \quad (7)$$

where $r$ is the number of units in hidden layer 2, $Exp(\bullet)$ element-wisely computes the natural exponential of the input matrix, and $N_{n \times r}(0, \bullet)$ returns an $n \times r$ matrix filled with random values with $(0, \bullet)$ normal distribution.

The input of the decoder is $Z$, the output of the encoder. Here, we focus on reconstructing the adjacency matrix, $A$. The reconstructed adjacency matrix is denoted by $A'$ and can be computed by the following formula.

$$A' = sigmoid(ZZ^T), \quad (8)$$

where $Z^T$ is the transpose *matrix* of $Z$ and the details of $sigmoid(\bullet)$ are given as follows.

$$sigmoid(t) = \frac{1}{1+e^{-t}} \quad (9)$$

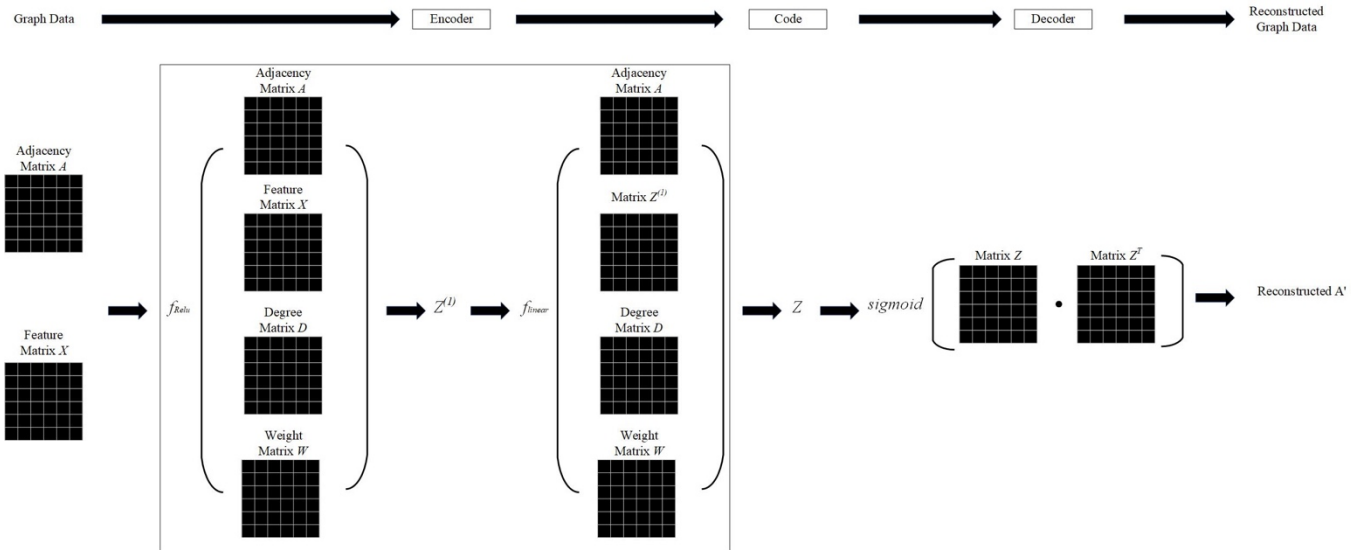


**Figure 1. The upper part is the diagram of a general graph autoencoder. The lower part is the diagram of a GCN based graph autoencoder proposed by Kipf and Welling** [12]**.**

## 3. TRAINING STRATEGY

One goal of autoencoder training is to minimize the generalization error, which reflects the generalization ability of an autoencoder. It is reasonable to assume that the training set can represent the whole input space. We propose a simple training strategy to create new training samples based on the existing training set by adding perturbation to original training samples.

As for a given training sample, its adjacency matrix is filled with one and zero. If an element is one, it suggests the corresponding pair of nodes are connected by an edge. Similarly, zero indicates the corresponding pair of nodes are not connected. Note that we only focus on the edges between different nodes, so the elements of the diagonal of the adjacency matrix are ignored. Here, we design a training strategy for sparse adjacency matrixes, while this strategy can also be adjusted for dense adjacency matrix. We set a small perturbation rate denoted by p. If there are totally u edges in an original training sample, $\lfloor pu \rfloor$ elements of the adjacency matrix will be randomly picked for value change from one to zero, where $\lfloor \bullet \rfloor$ is the floor function. Similarly, $\lfloor pu \rfloor$ elements with original value zero will also be randomly selected for changing values to one. In the training, we feed perturbed training samples instead of the original samples to the training algorithm. This strategy is summarized as follows.

---

Training Strategy (perturbation rate $p, 0 < p \ll 1$):

For each iteration:

randomly remove edges with rate $p$;

randomly add new edges with rate $p$;

feed the perturbed sample to the original training algorithm.

End For

---

The working flow of this strategy is provided in Figure 2.

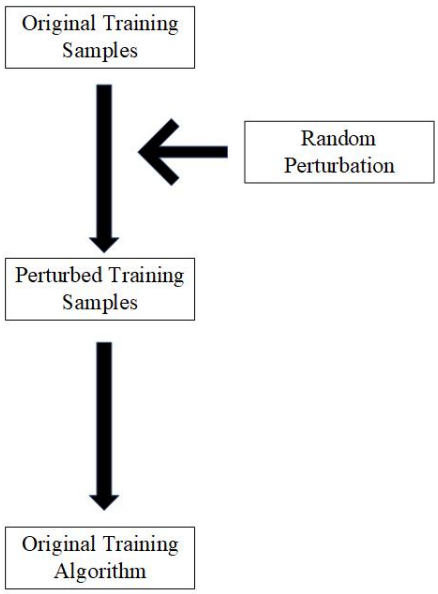


**Figure 2. The working flow of the proposed training strategy.**

In the unsupervised learning framework of autoencoder, the training algorithm can use the input as expected output. Based on this property, our training strategy generates "new" samples to avoid the overfitting problem. This strategy only updates the training samples, so it can flexibly fit many existing training algorithms. Different from denoising training algorithms, in which the expected output is the original input, our strategy also uses the perturbed samples as expected output of decoder. Based on our strategy, original samples are replaced by perturbed samples in the training. The number of samples processed by the training algorithm is the same. Therefore, the computational cost of training algorithm does not significantly increase. The original training algorithm is the special case of our strategy with perturbation rate zero.

## 4. EXPERIMENTS

**Table 1. The information of data sets used in experiments**

| | Cora | CiteSeer |
|---|---|---|
| The number of Nodes | 2708 | 3327 |
| The number of edges | 5429 | 4732 |
| The number of Features | 1433 | 3703 |

**Table 2. Results for Link Prediction on Cora. GAE, VGAE, ARGE, and ARVGE are original methods, while GAE+, VGAE+, ARGE+, and ARVGE+ are the corresponding methods with the proposed strategy.**

| Method | AUC | AP |
|---|---|---|
| GAE | 91.0 ± 0.01 | 92.1 ± 0.01 |
| GAE+ | 91.2 ± 0.01 | 92.4 ± 0.01 |
| GVAE | 91.8 ± 0.01 | 92.8 ± 0.01 |
| GVAE+ | 92.1 ± 0.01 | 92.9 ± 0.01 |
| ARGE | 88.6 ± 0.01 | 90.5 ± 0.01 |
| ARGE+ | 90.4 ± 0.01 | 91.9 ± 0.01 |
| ARVGE | 90.6 ± 0.01 | 92.2 ± 0.01 |
| ARVGE+ | 90.9 ± 0.01 | 92.4 ± 0.01 |

To verify the effectiveness of our strategy, we conduct experiments of link prediction on two data sets: Cora [15] and CiteSeer [5]. The information of these two data sets are given in Table 1 [12, 16]. Both data sets were used for testing [12, 16]. Our experiments test GAE, VGAE, ARGE, and ARVGE, four GCN based autoencoder, with and without using our strategy. We follow the same setting used in experiments of [12]. In each data set, 5% connected node pairs and 5% non-connected node pairs are randomly picked for the validation set. Similarly, 10% connected node pairs and 10% non-connected node pairs are randomly picked for the test set. The rest is used for the training set. The learning rate is 0.01. Each time, the training uses Adam algorithm [9] and takes 200 iterations. In practice, the proposed strategy uses the original training sample in the last iteration and newly generated samples in other iterations. The dimensions of the first and second layer are 32 and 16, respectively. As for other parameters, we use the default setting of each method.

**Table 3. Results for Link Prediction on CiteSeer. GAE, VGAE, ARGE, and ARVGE are original methods, while GAE+, VGAE+, ARGE+, and ARVGE+ are the corresponding methods with the proposed strategy.**

| Method | AUC | AP |
|---|---|---|
| GAE | 89.2 ± 0.02 | 89.9 ± 0.01 |
| GAE+ | 90.1 ± 0.01 | 90.6 ± 0.01 |
| GVAE | 90.7 ± 0.01 | 92.0 ± 0.01 |
| GVAE+ | 91.0 ± 0.01 | 92.4 ± 0.01 |
| ARGE | 84.8 ± 0.02 | 87.1 ± 0.01 |
| ARGE+ | 87.2 ± 0.01 | 88.9 ± 0.01 |
| ARVGE | 88.3 ± 0.01 | 90.0 ± 0.01 |
| ARVGE+ | 90.3 ± 0.01 | 91.6 ± 0.01 |

The experiments apply area under the ROC curve (AUC) and average precision (AP) scores [4, 14, 22] to measure performance. As for each method, we repeatedly train ten times on either data set. In order to easily reproduce training results, random seed of the *i*th time is set to *i*. We report average scores and related standard deviations.

The experimental results on Cora are given in Table 2. These results show the proposed strategy consistently improves the performance of each method on Cora. The experimental results on CiteSeer are given in Table 3. The proposed strategy also consistently improves the performance of each method on CiteSeer. The consistent advantage of our strategy on all methods and both data set confirms the effectiveness of this strategy.

# 5. CONCLUSION

This paper proposes a simple training strategy for graph autoencoder. It generates new training samples by adding random noise to original sample. This strategy can be applied in most existing training algorithms. The experimental results show this strategy improves the performance. Furthermore, our strategy also provides a new way to avoid the overfitting problem.

# 6. ACKNOWLEGEMENTS

This work was partially supported by the National Science Foundation under grant number 1813252.

# 7. REFERENCES

[1] Cai, H. et al. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*. 30, 9 (2018), 1616–1637. DOI:https://doi.org/10.1109/TKDE.2018.2807452.

[2] Cao, S. et al. 2016. Deep neural networks for learning graph representations. *Proceedings of 30th AAAI Conference on Artificial Intelligence* (2016), 1145–1152.

[3] Duvenaud, D. et al. 2015. Convolutional networks on graphs for learning molecular fingerprints. *Proceedings of the 28th International Conference on Neural Information Processing Systems* (2015), 2224–2232.

[4] Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters*. 27, 8 (2006), 861–874. DOI:https://doi.org/https://doi.org/10.1016/j.patrec.2005.10.010.

[5] Giles, C.L. et al. 1998. CiteSeer: An automatic citation indexing system. *Proceedings of the ACM International Conference on Digital Libraries*. (1998), 89–98.

[6] Goodfellow, I. et al. 2014. Generative adversarial nets. *Advances in Neural Information Processing Systems 27* (2014), 2672–2680.

[7] Goyal, P. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*. 151, (2018), 78–94. DOI:https://doi.org/https://doi.org/10.1016/j.knosys.2018.03.022.

[8] Haykin, S. 2008. *Neural networks and learning machines*. Pearson.

[9] Kingma, D.P. and Ba, J.L. 2015. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations* (2015).

[10] Kingma, D.P. and Welling, M. 2014. Auto-encoding variational bayes. *Proceedings of the 2nd International Conference on Learning Representations* (2014), 1–14.

[11] Kipf, T.N. and Welling, M. 2017. Semi-supervised classification with graph convolutional Networks. *Proceedings of International Conference on Learning Representations* (2017), 1–13.

[12] Kipf, T.N. and Welling, M. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[13] Li, Y. et al. 2016. Gated Graph Sequence Neural Networks. *Proceedings of the International Conference on Learning Representations* (2016).

[14] McClish, D.K. 1989. Analyzing a portion of the ROC curve. *Medical Decision Making*. 9, (1989), 190–195. DOI:https://doi.org/https://doi.org/10.1177/0272989X8900900307.

[15] McDowell, L.K. et al. 2009. Cautious collective classification. *Journal of Machine Learning Research*. 10, (2009), 2777–2836.

[16] Pan, S. et al. 2018. Adversarially regularized graph autoencoder for graph embedding. *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. (2018), 2609–2615. DOI:https://doi.org/10.1523/JNEUROSCI.1317-08.2008.

[17] Scarselli, F. et al. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*. 20, 1 (2009), 61–80. DOI:https://doi.org/10.1109/TNN.2008.2005605.

[18] Tu, K. et al. 2018. Deep recursive network embedding with regular equivalence. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. (2018), 2357–2366. DOI:https://doi.org/10.1145/3219819.3220068.

[19] Vincent, P. et al. 2008. Extracting and composing robust features with denoising autoencoders. *Proceedings of the International Conference on Machine Learning* (2008), 1096–1103.

[20] Wang, C. et al. 2017. MGAE: Marginalized graph autoencoder for graph clustering. *Proceedings of the International Conference on Information and Knowledge Management, Proceedings* (2017), 889–898.

[21] Wang, D. et al. 2016. Structural deep network embedding. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), 1225–1234.

[22] Wikipedia entry for the Receiver operating characteristic: *https://en.wikipedia.org/wiki/Receiver_operating_characteristic*.

[23] Wu, Z. et al. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv preprint arXiv:1901.00596v2*. (2019).

[24] Yu, W. et al. 2018. Learning deep network representations with adversarially. *Proceedings of the International Conference on Knowledge Discovery and Data Mining* (2018), 2663–2671.

[25] Zhang, D. et al. 2018. Network Representation Learning: A Survey. *IEEE Transactions on Big Data*. (2018). DOI:https://doi.org/10.1109/tbdata.2018.2850013.

[26] Zhou, J. et al. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434v3*. (2018).