Random and Guided Edge Perturbations for Improved Learning on Graphs

Alex Derhacobian
Department of Computer Science
Stanford University
alexder@stanford.edu

Edward Vendrow
Department of Computer Science
Stanford University
evendrow@stanford.edu

Abstract

Deep graph neural networks have enabled state-of-the-art performance in graph prediction tasks such as node property prediction. However, these models tend to have many parameters which risks over-fitting, especially when the dataset size is small relative to the parameter size. Graph data may also be noisy with missing or incorrect edges, which may further degrade performance. In this paper we present an exploration of DropEdge, a proposed technique for addressing these issues. Our results confirm the effectiveness of dropping random edges for training deep learning models on graphs. DropEdge applied to GraphSage outperforms the current Open Graph Dataset (OGB) Leaderboard benchmark for GraphSage on the *ogbn-proteins* dataset. Additionally, we propose a novel variation to DropEdge. Our variation consists of using a link predictor to generate edge probabilities and strategically adding and dropping edges from the input graph. This augmented input graph is then used as a noisy input for a downstream node classification task.

1 Introduction

Introduced by Kipf & Welling [1], the deep Graph Convolutional Network (GCN) and related works have achieved success in applying convolutional methods to graph-based data[2][3][4][5]. Building on the success of convolutional architectures in computer vision, GCNs similarly perform convolutions on graph data by repeatedly transforming and aggregating information from adjacent nodes.

Deep graph neural networks have enabled state-of-the-art performance in graph prediction tasks such as node property prediction. However, these models tend to have many parameters which risks over-fitting, especially when the dataset size is small relative to the parameter size. Additionally, graph data may be noisy with missing or incorrect edges. Fitting a model to these incorrect edges may further degrade performance. Deep graph convolutional networks further suffer from over-smoothing, a pheonmenon during which nodes will gather representations of an increasingly large number of nodes, leading to indistinct node embeddings and poor performance. Preventing over-smoothing in graph-based learning has recently been an area of focus for researchers [6][7][8].

One proposal to address these problems is DropEdge [6], which aims to solve the problems of over-fitting and over-smoothing by randomly removing edges from the input graph during training. This idea is inspired by the dropout regularization method in classic neural networks. DropEdge can be applied to entire graphs or to single layers, and both variations have been shown decrease over-fitting and over-smooothing. DropEdge is particularly notable because the method is simple to implement and can be used with nearly any existing GCN.

Final Project, CS224w (Winter 2021)

2 Background

2.1 OGB Datasets

We will be using the OGB *ogbn-proteins* dataset [9]. This dataset represents biologically meaningful associations between proteins, including physical interactions, co-expression, or homology. This dataset is a good candidate for random edge dropping techniques because protein interaction graphs tend to be generally fairly noisy with many false positives [10]. Previous work has been done in de-noising protein-protein interactions, so we believe that this will provide the optimal conditions to test our method's ability to make a model more robust to noise.[11][12] The results presented in this paper will be based on performance on this dataset.

2.2 GCNs and Deep GCNs

Although residual connections have enabled CNNs to achieve state-of-the-art performance in computer vision tasks, creating deep GCNs has proven more challenging. Kipf & Welling attempt to build a deep (i.e. > 3 layer) GCN using such residual connections, but find that the performance decreases rather than improves.

Deep graph convolutional networks further suffer from over-smoothing, where many-layered networks will pool representations of an increasingly large k-hop neighborhood. In the extreme, this will push the representations of all nodes to a single fixed value, resulting in poor performance. Oono & Suzuki [13] extended this idea by proving theoretically that in the limit of infinite layers, GCNs suffer from information loss resulting from node features exponentially converging to a small subspace. More formally, over-smoothing occurs when after some hidden layer L, the distance between the input matrix and subspace $\mathcal{M}$ is bounded below by some ϵ such that namely, $d_{\mathcal{M}}(\mathbf{H}^{(\ell)}) <, \forall \ell \geq L$

3 Motivation

The motivation behind perturbing the edge set of the input graph is the well known dropout method developed by Hinton et. al.[14]. Dropout aims to reduce over-fitting and increase model robustness by randomly zero-ing out feature dimensions. Edge perturbation methods like DropEdge are similarly motivated. These methods act as a data augmentation method as well as a method to prevent over-fitting. Unlike dropout, edge perturbations also reduces over-smoothing[6].

4 Methods

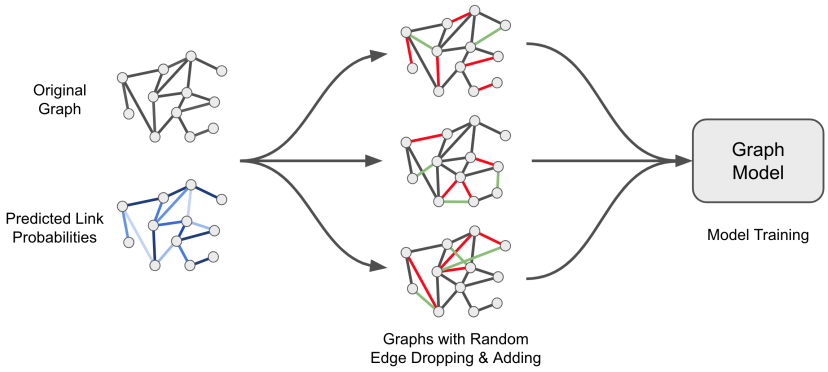

Figure 1: This diagram illustrates our variations on the DropEdge method of edge perturbation. Inspired by Zhao et. al. [15], we train a link predictor on the original graph. Then, using the edge probabilities of input edges that are generated by the link predictor, we strategically add and drop edges from the input graph. This method acts as a data augmentation method for graph-structured data by producing realistic synthetic input graphs for the node classification task.

Model	Standard	Single Drop	DropEdge
GCN, 3-layer	77.16	77.29	77.91 *
GCN, 8-layer	78.47	77.63	79.40*
GraphSage, 3-layer	83.45	83.46	83.38*
GraphSage, 8-layer	84.05	84.16	84.32*
MLP, 3-layer	76.82	77.73	78.01*
MLP, 8-layer	77.75	77.68	78.21*

Table 1: Validation accuracies across different model architectures for the regular model, random edge dropping before training, and random edge dropping every epoch (DropEdge). Drop probabilityes are $p = 0.2$. *: Outperforms OGB leaderboard benchmark for same model architecture

The methods we explore are based on the idea of random edge-dropping presented by DropEdge. Additionally, we propose variations to DropEdge inspired by Zhao et. al.[15]. In our variation, we use a link predictor to generate a sparse predictive adjacency matrix A^{pred} that guides edge perturbations of the input graph.

4.1 DropEdge

At each iteration, DropEdge randomly selects some V_p number of edges to drop. Mathematically, this means that beginning with adjacency matrix A we obtain matrix A_{drop} such that

$$A_{drop} = A - A' \,,$$

where A' is a sparse matrix containing exactly V_p nonzero elements selected from A. The number of nonzero elements selected is determined by a new hyperparameter defined as the drop rate, p. As suggested by Kipf & Welling (2017), the re-normalization trick is applied to the modified adjacency matrix to obtain $\hat{A}_{drop}$, which is then used in training [1]. Here is an example of DropEdge in action. Consider some adjacency matrix A of a directed graph $\mathcal{G}$ as defined below and a drop-rate $p = 1/3$:

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad A' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A_{\text{drop}} = A - A' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

DropEdge can also be performed at each layer of the graph, so that at each layer (ℓ) in the GCN, there is a unique adjacency matrix $\hat{A}^{(\ell)}_{\text{drop}}$. The layer-wise DropEdge will add more randomness than the graph-wise approach. By producing various perturbations of the input graph, DropEdge acts as a form of data augmentation. DropEdge also minimizes over-smoothing. DropEdge increases the ϵ-smoothing layer L, thus reducing over-smoothing. In experiments, DropEdge improves performance on popular GCNs like GCN, ResGCN, JKNet, IncepGCN, and GraphSAGE.

4.2 Guided Edge Dropping

DropEdge can be seen as a data augmentation technique, where the random edge dropping generates many noisy or deformed versions of the original graph. While this augmentation is useful, it is done at random and without consideration for the graph architecture. This may be important in graphs where some types of connections may be noisy, but others are not noisy and are important for proper message passing. Based on this intuition, we propose an edge dropping scheme which uses a previously trained link prediction model to drop edges based on edge probabilities (ie. the probability that there exists an edge between some nodes i and j). Formally, given an adjacency matrix A and link prediction model $\mathcal{M}$, we define the predicted matrix A as:

$$A^{pred}_{ij} = \begin{cases} 0 & \text{if } A_{ij} = 0 \\ \text{P}(\text{A}_{ij} = 1 \mid \mathcal{M}) & \text{if } A_{ij} = 1 \end{cases}$$

Intuitively this matrix wil give high scores to links that are essential to the graph, and low scores to noisy and incorrect links. Since we only need to compute the probability matrix A^{pred} for already

existing edges, this matrix is not prohibitively expensive to compute, especially for sparse graphs. In the case where it is computationally feasible to calculate the edge probabilities for every possible edge in the graph, then we could calculate $P(A_{ij} = 1)$ for every edge (i, j) in the matrix and then strategically add or drop edges from the graph. An example of this is implemented by Zhao et. al. [15], where randomized and guided edge perturbations are added to create noise during training. In this training scheme, a pretrained model is used to find the edge probabilities for the entire graph or some subgraph. Then, the bottom $i|\mathcal{E}|$ edges with lowest edge probabilities are dropped from the graph, and the top $j|\mathcal{E}|$ non-edges with the highest edge probabilities are added to the graph. i and j are hyperparameters that control the number of edges that are added and dropped from the graph, and $|\mathcal{E}|$ is the total number of edges in the graph or subgraph.

Using this graph and an edge drop probability p, we compute the final adjacency matrix used for training:

$$A'_{ij} = \mathbb{1}\left\{\alpha A_{ij} + (1-\alpha)A_{ij}^{pred} > p\right\}$$

Note that we also include a parameter α which allows controllability over the level of randomness used to determine the final dropped matrix.

4.3 Implementation Details

Since DropEdge acts as a general improvement over existing models, we base our code off of the benchmark models provided by the OGB team. In particular, we implement DropEdge and Guided Edge Dropping functionality into the GCN, GraphSage, and MLP networks for node property prediction on *ogbn-proteins*.

5 Experiments

We contribute to the understanding of the performance of DropEdge and related methods across various metrics and modifications. We perform experiments with three model architectures: GraphSage, Graph Convolutional Network (GCN), and a standard Multilayer Perception (MLP) with 3 hidden layers and 256 weights in each hidden layer. For all the models, we used a standard binary cross-entropy loss. The task of interest is node classification. All nodes in the *ogbn-proteins* have feature labels of length 112, where each feature label is a binary label. Since this is a multilabel binary classification problem, we used a binary cross-entropy loss for multiple categories.

$$\mathcal{L}(x, y) = \{\ell_1, \ldots, \ell_N\}^T, \qquad \ell_n = -w_n[y_n \cdot \log(\sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))] \tag{1}$$

where N is the batch size. Since there were 112 binary classification tasks for each node, we calculated this loss for all 112 classes.

Our experiments confirm that DropEdge is an improvement over baseline models. Most importantly, GraphSage with DropEdge shows performace improvements over the current OGB Leaderboard baseline for GraphSage applied to the *ogbn-proteins* dataset. We also investigate variations on the DropEdge method and its improvements to training. We propose improvements to the DropEdge method using a trained link predictor to guide the removal and addition of edges to the input graph fed to the node classifier.

Due to the effectiveness of random edge dropping, we propose a contribution to the DeepSnap library. Our contribution is an implementation of random edge dropping on a sparse adjacency matrix. The user can specify the drop probability in our implementation.

5.1 Deep Model Training

One important advantage of random edge dropping is that it enables the training of deeper models which may otherwise overfit on the training data. We test this empirically. First, we trained baseline models without any edge dropping or adding. Next, we trained all three models with a single round of edge dropping at the beginning of training. Since drop probabilities of $p = 0.2$ proved to be the most effective, we used this drop probability for this round of experiments. Finally, we trained all three models with a round of edge dropping at the beginning of every epoch of training. Our results

Model	$p = 0.1$	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$	$p = 0.6$	$p = 0.7$	$p = 0.8$
GraphSage, 3-layer	83.29	83.46	83.53	83.15	83.45	83.40	83.31	83.32
MLP, 3-layer	76.89	77.95	77.84	77.69	78.21	77.66	76.52	76.27

Table 2: Highest validation accuracies for GraphSage 3-layer, and MLP 3-layer architectures across different dropout rates.

(see Table 1) showed performance improvements when edges were dropped with probability $p = 0.2$ at every epoch of training.

Notably, the result achieved by GraphSage with DropEdge is not only better than the baseline model, but it also outperforms the GraphSage model on the OGB Leaderboard for the *ogbn-proteins* dataset (see Table 1).

5.2 Early Training Stability

We find that using DropEdge significantly stabilizes and improves the early stages of training. We found that DropEdge leads to higher validation, test, and train accuracy earlier on in training. In our experiments, we log the validation, test, and training accuracies of an MLP model for node classification with DropEdge and without DropEdge. Figure 2 shows that during the initial training epochs, DropEdge accelerates training.

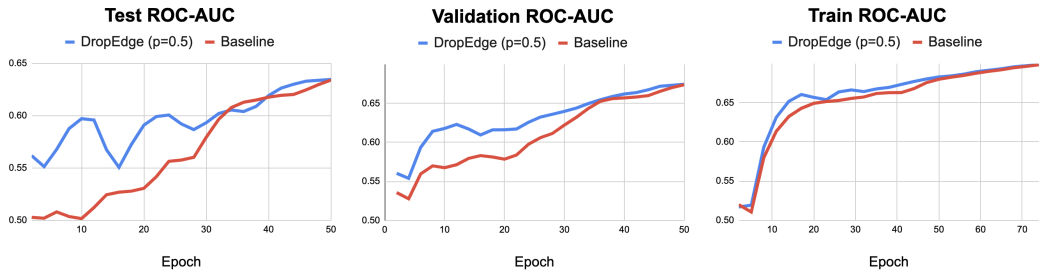

Figure 2: With a drop rate of 0.5, DropEdge shows to significantly speed up training during initial epochs. The blue plot logs accuracy with a drop rate of 0.5, and the red plot logs accuracy with no DropEdge. In both experiments, no dropout was applied to the model and the learning rate was 0.01. These experiments were performed on the MLP model and the *ogbn-proteins* dataset

5.3 Dependence of Drop Probability

We experiment with the effect that the drop probability hyperparameter has on model training. We evaluated training, validation, and test accuracy of GraphSage and MLP model architectures with edge dropping rates from 0.1 to 0.8. We found that a change in the drop rate did not have a significant impact on the performance of our models. This is because the *ogbn-proteins* dataset has node features that are generated based on edge-level features. So even with no edges in the graph, the model will still be able to learn node labels. Alternatively, in cases where node labels are not directly dependent on incident edge labels, DropEdge has varying results for different dropping rates.[6] In this case, with a drop rate of $p = 0.8$, the model will still be able to learn node labels, but this leads to slower convergence and more distance between node embeddings in the latent embedding space.

While DropEdge does show improvements in validation accuracy (as seen in Table 2), we notice that if node features contain edge-level information, variations in drop rate will not significantly affect model performance.

Model	Drop Rate	Highest Train	Highest Valid	Final Test
GraphSage, 3-layer	$p = 0.2$	88.10 ± 0.22	83.33 ± 0.18	77.35 ± 0.44
GraphSage, 8-layer	$p = 0.2$	89.106 ±0.27	83.922 ±0.292	77.9725 ±0.87

Table 3: These are the results from guided edge perturbations with GraphSage. We chose to test guided edge perturbations on GraphSage since it was the best performing model from the models we tested. The 8-layer GraphSage model with guided edge dropping yields better results than the OGB Leaderboard.

5.4 Guided Edge Perturbations

We experiment with guided edge perturbations on the input graph for the node classification task. Our training setup consists of first training a link predictor. This link predictor produces a predictive sparse adjacency matrix A^{pred}, where $A^{pred}_{ij} = p, p \in [0, 1]$, where p is the probability that there exists an edge between nodes i and j for all edges (i, j) in the sparse adjacency matrix of the input graph. As we described above, we use A^{pred} to generate a new input graph for the node classication task determined by an updated adjacency matrix, which we denote as A'.

$$A'_{ij} = \mathbb{1}\{\alpha A_{ij} + (1 - \alpha)A^{pred}_{ij} > p\}$$

We define α as a hyperparameter which controls the level of randomness in the input graph. p is the drop probability as defined by the DropEdge method.

Figure 1 illustrates the full pipeline of the guided edge perturbation method. The variations of the training graph to the graph model act as augmentations to the input data. Table 3 shows the results from training the guided edge perturbations pipeline. We report average validation accuracy, average test accuracy, and confidence intervals for our guided edge dropping (see Table 3). The validation and test accuracy for the GraphSage model with 8-layers and guided edge dropping shows improvements over the GraphSage model on the OGB Leaderboard. Improved model performance from graph data augmentation methods via edge manipulation in previous works [15] lead us to believe that our link prediction pipeline could produce even better performance improvements with further investigation.

6 DeepSnap Contribution

Because of the effectiveness of random edge dropping methods across many types of models, we believe that this method should be a tool available for anyone to use in graph training. To that end, we submit a contribution to the Stanford DeepSnap library. Our contribution consists of an implementation of the random edge dropping on a sparse adjacency matrix. This method supports an edge dropping probability parameter which can be specified by the user.

The implementation of the method in DeepSnap is available at https://github.com/evendrow/deepsnap. This addition has been tested to work, and we plan to submit a pull request as a contribution to the master repository.

7 Conclusion and Future Work

In this paper we have explored DropEdge as a generalized technique to improve training, and proposed an improvement to drop edges in a more guided way to aid graph training. Using the OGB dataset *ogbn-proteins* as a benchmark, we showed that implementing these methods gave statistically significant performance increases across a range of model architectures and outperformed the baseline models, especially with deeper models. We expect that future research will explore further related methods to randomly drop or add edges during training. In particular, we believe that guided random methods, where learned graph features are used to intelligently modify the graph, will continue to see success. Such methods will allow for graph models to perform better on noisy data, opening GCNs to new potential applications.

8 Contributions

The authors contributed equally to this work.

9 Code

See github.com/evendrow/cs224w_project

References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[2] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *arXiv preprint arXiv:1403.6652*, 2014.

[3] Zhou Jie, Cui Ganqu, Zhang Zhengyan, Yang Cheng, Liu Zhiyuan, Lifeng Wang, Li Changcheng, and Sun Maosong. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1403.6652*, 2014.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216v4*, 2018.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *arXiv preprint arXiv:1607.00653*, 2016.

[6] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[7] Zhichao Han Qimai Li and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *n Thirty-Second AAAI Conference on Artificial Intelligence*, 2018a.

[8] Fengwen Chen Guodong Long Chengqi Zhang Zonghan Wu, Shirui Pan and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[9] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[10] Oleksii Kuchaiev, Marija Rašajski, Desmond J Higham, and Nataša Pržulj. Geometric de-noising of protein-protein interaction networks. *PLoS computational biology*, 5(8):e1000454, 2009.

[11] Quaid D Morris, Brendan J Frey, and Christopher J Paige. Denoising and untangling graphs using degree priors. *Conference on Neural Information Processing Systems*, 2003.

[12] H.-L. Ng D.W. Rice T.O. Yeates E.M. Marcotte, M. Pellegrini and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285.

[13] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification, 2021.

[14] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[15] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830*, 2020.