RANKEDDROP: ENHANCING DEEP GRAPH CONVOLUTIONAL NETWORKS TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) are playing a more and more important role for analyzing unstructured data from the complex real world. Introducing random edge dropping from the input graph at training epochs could reduce over-fitting and over-smoothing phenomenon and increase the depth of GNNs. However, such method relies strongly on the chosen randomness. It makes the accuracy depend on the initialization of the randomness, which lets the selection of hyper-parameters be even more difficult. We propose in this paper RankedDrop a novel method with a spatial-aware dropping-edge selection. The selection takes into account the graph global information using PageRank, and graph local neighborhood information with node degree. RankedDrop provides a more stable training results comparing to the state-of-the-art solution, by maintaining the advantages of random edge dropping. Furthermore, RankedDrop is a general method that can be deployed on a deep learning framework for enhancing performance of GNNs.

1 INTRODUCTION & CONTEXT

Convolutional Neural Networks (CNNs) demonstrated a great success in our today's daily life for image classification and many other applications. However in the real world there are still many non-Euclidean (graph) data like social networks or reference systems that cannot be handled by CNNs. After Defferrard et al. (2016) introducing Graph Neural Networks (GNNs), Defferrard et al. (2016) generalized CNNs to graph to exploit their potential for classification problems on non-Euclidean data structure. The computation of Graph Convolutional Neural Networks (GCNs) can be summarized as iterative neighborhood aggregations with a message passing schema (Huan et al. (2021)).

He et al. (2016) showed that deeper CNN has higher potential to achieve better precision. However, modern GCNs (Kipf & Welling (2017); Pei et al. (2021); Hamilton et al. (2017)) can work with very limited number of layers, because training deep neural networks is a very complex task (Claesen & De Moor (2015)), the complexity of the computed function grows exponentially with depth (Raghu et al. (2017)), and the deeper the networks are, the more they are subject to over-smoothing (Li et al. (2018); Chen et al. (2020)). Meanwhile, deeper GCN and/or small graph datasets could lead to over-fitting, where a model could fit well the training data but poorly the testing data.

Dropout (Hinton et al. (2012); Srivastava et al. (2014)) is a promising regularization techniques to reduce over-fitting. In the field of GCN, DropEdge introduced by Rong et al. (2019), which randomly removes a certain proportion of edges from the input graph at each epoch, showed promising results to reduce the convergence speed of over-fitting and over-smoothing. Moreover, the random dropping happened on the message passing schema of most of GCNs. Therefore such method could be applied for many GCN backbone models like GCN (Kipf & Welling (2017)), ResGCN (Pei et al. (2021)), GraphSage (Hamilton et al. (2017)), IncepGCN (szegedy2016rethinking) and JKNet (Xu et al. (2018)).

However, the accuracy obtained by DropEdge depends on how the randomness of dropping is initialized. Moreover, the only parameter that can be adjusted in DropEdge is the percentage of edges that will be dropped. Missing of control on the way of how dropping edges be selected, may limit possibilities to optimize GCN training according to application domain and chosen backbone architecture. Furthermore, a graph structure includes a lot of useful information (Newman (2003)).

Random dropping may destroy graph structure information and again limits the potential of optimizing GCN training.

This paper proposes RankedDrop a novel method with a spatial-aware dropping-edge selection. The selection takes into account the graph global information using PageRank, and graph local neighborhood information with node degree. Graph structure information is extracted to reduce the impact of randomness in the selection and also to improve the final accuracy after training. RankedDrop provides a more stable training results comparing to the state-of-the-art solution, by maintaining the advantages of random edge dropping including over-fitting and over-smoothing reduction and being a general method for different GCN backbones. Shown by our experiments, the accuracies of deep GCNs on semi-supervised learning are significantly improved by using RankedDrop.

2 RANKEDDROP METHOD WITH DATA SELECTION

RankedDrop is a general method, and it is applied on the input graph of a GCN training before each epoch. It first extracts a score based on graph analysis for each node (Sec 2.2); after that the nodes are reordered to control the selection probability then selected according to the computed score (Sec 2.3); at the end the edges of selected nodes are selected and we drop the selected edges to create the new input graph (Sec 2.4).

2.1 NOTATIONS AND PRELIMINARIES

We use an adjacency matrix $\mathbf{A}$ to represent the original input graph G, and nnz the number of non-zero value of $\mathbf{A}$, a.k.a. the number of edges of the graph G. We denote p the proportion of edges from G that will be dropped. Therefore, after dropping, the new input graph G_{drop} has $(1-p) \times nnz$ edges. We denote the resulting adjacency matrix $\mathbf{A}_{drop}$ for G_{drop}, and we use $\mathbf{A}'$ to denote the matrix of $p \times nnz$ dropped edges. The relation between the above three matrices is:

$$\mathbf{A}_{drop} \quad = \quad \mathbf{A} - \mathbf{A}' \tag{1}$$

The theorem 1 introduced in the paper (Rong et al. (2019)) proved that training GCN on G_{drop} instead of G allows to reduce the speed of convergence of the over-smoothing and to reduce the loss of information. The idea is based on the concept of mixing time in the random walk theory (Lovász (1993)), and the proof is based on the work of Oono & Suzuki (2019). Luo et al. (2021) demonstrated again the effectiveness of such dropping method.

In the following parts of the paper, we focus on the methods of selection of edges to drop, which are the main contributions of RankedDrop. RankedDrop ranks the nodes in order to assign a weight during the selection. Different from Sparsification (Eppstein et al. (1997)) or DropEdge (Rong et al. (2019)), the goal of RankedDrop is to control the randomness with several parameters, but not to completely control the choice of the drop edges, to create at each iteration a sub-graph in a more intelligent way. It means that we bias the probability (greater or lesser) of being selected of each edge according to our graph analysis, to create dropping strategy by reducing the dependency on full randomness.

2.2 GRAPH INFORMATION EXTRACTION

Most GCN architectures are mainly oriented on inter-neighbor communication (Huan et al. (2021)). The information propagates through edges w.r.t. GCN layers. The shorter the path between two nodes, the more they will influence each other. Removing the most impactful neighbors limits such over-influence and reserves space for taking into account the information from other neighbors for each epoch and among the epochs in a training. With the above idea, we propose here a node ranking strategy in order to prepare a better dropping selection for the next steps. Two kinds of graph structure information are extracted and used in the node selection step:

Local structure information The degrees of each node, which reflects the local impact of the node on its neighborhood. Higher degree reflects stronger influence from a local point of view in the

graph. If a node has a lot of neighbors, it will have an impact at each layer on them and therefore the information it contains will be strongly taken into account at the local level. The degrees are extracted from the adjacency matrix $\mathbf{A}$. Consider that $\mathbf{A}$ is an $n \times n$ matrix and that its number of nonzero elements is nnz. A vector of size n is used to store the degrees of the nodes of the graph.

Global structure information Different graph node ranking algorithms (Agarwal & Chakrabarti (2007)) could be used here to judge importance of each node on the global graph. We use in the paper the PageRank algorithm (Page et al. (1999)) to generate the score of importance, because (1) PageRank is the most studied algorithms of the last decades, by our knowledge it can be easily implemented in a distributed way to accelerate its computation; (2) It was already used in GNNs to reduce the over-smoothing (Bojchevski et al. (2019)). From the adjacency matrix $\mathbf{A}$, a vector of size n will be returned and will contain the score of each node. The algorithm 1 represents the implementation of PageRank used in RankedDrop. It shows that the main operation of each iteration of the PageRank is a matrix-vector multiplication where the output vector is used to perform the next iteration multiplication. By considering $\mathbf{A}$ as sparse, the cost of this sequence of sparse matrix-vector multiplications is reduced and can be executed efficiently in a distributed way (Hugues & Petiton (2010)), which allows to optimize the extra computations that PageRank requires. This iterative method stops when the convergence has reached the expected precision. The result vector of the last iteration contains then the scores of each node of the graph and all elements are between 0 and 1. The higher the score, the more important the node is in the global graph. A β coefficient is also introduced during the PageRank. It is an optimization allowing to redistribute a part of the scores of each node among all the other nodes. In this way, the convergence of the result vector is faster and avoids that all the score is distributed only within the strongly connected component. Conventionally, the β coefficient is fixed around 0.85, this value was used for the experiments in the section 3.

The local and global structure information is used to rank the nodes of the graph to determinate the overall importance of each node in the graph. The importance of the nodes in the global and/or local structure of the graph gives a score to each node so that the nodes with a higher score are more often included in the matrix $\mathbf{A}_{drop}$. Thus, the structure of the graphs G_{drop} that will be generated at each iteration will be closer to the structure of the graph G than when the dropping is done randomly. We note s the vector of size n which stores the final score of the associated nodes. The computation of this score is flexible. There are many possibilities to compute the values of s by taking the information of the local and/or global structure, and potentially other information. At the end, the goal is to have a vector such that $\forall i \in [1, n]$, $0 \leq S_i \leq 1$ and $\sum_{i=1}^{n} S_i = 1$.

2.3 Node selection with probability control

After getting the score vector s, we sort the nodes according to their scores in a decreasing order. The permutations performed during the sorting are stored in memory in order to keep the association information between the nodes and the scores.

After the sorting, we create a probability scale from the sorted score vector by applying a Scan-With-Add (SWA) algorithm (a.k.a prefix sum, Blelloch (1990)). SWA will generate an interval between 0 and 1 for each node. Therefore, the node selection is no more in a fully random way but the randomness is limited in the interval. The resulting vector is of size n where the values are more and more ordered. The resulting vector is such that $SWA_s[n] = 1$. In addition, SWA could help visualize the inequalities of score between the nodes in the graph, like the Lorenz curve used in economics (Lorenz (1905)).

The node selection is performed with the SWA vector. The SWA value of each node corresponds to the probability of the node being selected. Formatting the score vector as a SWA accelerates the selection of nodes. For each node selection, we take a random number between 0 and 1 and find the node associated with this value in the vector SWA_s. A binary search (Knuth (1998)) on the SWA vector can find the node with a $\mathcal{O}(\log n)$ complexity, where it is necessary to browse element by element the vector of s scores at each node selection. We will discuss the selection of nodes from the SWA vector in more detail in the next section.

Algorithm 1 Algorithm to get the PageRank score from adjacency matrix

Input: $\mathbf{A}$ the adjacency (sparse) matrix, δ precision, β coefficient
Output: v vector of PageRank score of size n
Initialisation :
$sum \leftarrow 0$
$err \leftarrow$ INF
new vector tmp of size n
assign $\frac{1}{n}$ to each element in v
START LOOP
while $err > \delta$ **do**
  reset all element of tmp to 0
  $tmp \leftarrow$ SpMV between $\mathbf{A}$ and v
  for each $elem$ in tmp **do**
    $elem \leftarrow \beta * elem + (1 - \beta) * \frac{1}{n}$
  end for
  $err \leftarrow$ norm between tpm and v
  $v \leftarrow tmp$
end while
return v

Algorithm 2 Node selection from the score vector

Input: SWA_s the Scan-With-Add final score vector, $sumScore$ the sum of the remaining nodes' scores, $malus$ the vector of malus applied to each node.
Output: ind the index of the node to perform the drop edge
Initialisation :
$r \leftarrow randomin]0, 1[$
$r \leftarrow r * sumScore$
$m \leftarrow 0$
$a \leftarrow 0$
$b \leftarrow$ size of SWA_s
while b-a != 1 **do**
  $c \leftarrow (a + b)/2$
  $m \leftarrow m+$ malus on c node
  if SWA_S of $c - m - sB < r$ **then**
    $a \leftarrow c$
  else
    $b \leftarrow c$
    add $(c + b)/2$ on the potential malus node list
    add c in the explored nodes list
  end if
end while
return b

2.4 DROPPING EDGE SELECTION

The last step is to select the exact edges to drop. Different from the previous steps that can be performed only once in the beginning of training, the edge selection is performed for each epoch to generate a different subgraph. At each epoch, $p \times nnz$ edges are chosen from the selected nodes and are removed.

Different edge selection algorithms could be applied here. For example, the selection could be based on the tail, on the head, or directly removing all edges of a selected node (a.k.a DropNode). For the experiments presented in the section 3, we randomly select edges from the selected node. This adds randomness to the selection process. To select a node, we took our inspiration from the bisection method (Burden & Faires (1985)). By randomly pulling in a uniform way a number r between 0 and 1, we obtain the index of the node i that checks $SWA_s[i] < r < SWA_s[i + 1]$ by performing a dichotomy. The selection of edges and nodes is based on the SWA des scores SWA_s, , the importance of the node in the graph will influence its probability to be selected at each epoch. Thus, the randomness is controlled but the selection probabilities are different for each node so that the randomness takes into account the global structure of the graph. It is possible from the PageRank results and/or degrees vector to create subgraphs that keep the key nodes of the graph so that at each epoch the graph generated is consistent with the structure of the initial graph.

It is useful to keep an efficient selection method because it is performed a large number of times. This is why we have tried to optimize the implementation of this selection (see the algorithm 2) by adding a malus system when exploring the SWA vector to avoid selecting a node from which all edges have already been selected. When all the edges associated to a node have been dropped, there is no more interest to select this node again. Our optimization is based on the fact that the bisection method can be represented as a tree. At each step of the dichotomy, there is the possibility to move either the lower or the upper bound. To be sure not to select a node i, it is possible to apply a malus (equal to the score of the node i) to the explored branches when the upper bound is moved in the path to access the $SWA_s[i]$ (because all these paths lead to nodes further in the SWA_s vector). Thus,

it is enough to modify at most $\log n$ malus value instead of modifying a large number of values in SWA_s.

3 EXPERIMENTS AND DISCUSSIONS

RankedDrop is a general method that could equip different algorithms for the node selection and edge selection, and can be applied for different GNN architectures. In the experiments of this paper, we used the classic PageRank to compute scores for the node selection; we used basic random selection for the edge selection; and we used mostly the standard GCN as the training architecture, since Rong et al. (2019) and Luo et al. (2021) have already demonstrated the genericity of these Dropping methods for other GNN architecture. We believe such standard configuration could provide a clear and general idea of the potential offered by RankedDrop.

3.1 DATASETS AND ENVIRONMENT

Three standard citation datasets were used in our experiments: Cora, Citeseer and Pubmed. These datasets represent collections of scientific articles that are classified according to the paper's main research topic (Sen et al. (2008)). More information of these datasets can be found in the table x. We notice that these graphs are very sparse, since their number of edges per node is very low: in average about two edges per node. However, if we see closely, the highest degree nodes of those datasets have more than 100 edges (99 for Citeseer and up to 171 for Pubmed). This means that a very large number of nodes have a very low degree (≤ 2). Therefore, only few important nodes propagate their information very widely; the information of other low degree notes are quickly drowned. For example in Cora, the node with the highest degree is directly connected to more than 6% of the nodes in the graph.

Table 1: Datasets global informations

	Cora	***Citeseer***	***Pubmed***
Number of nodes	2 707	3 327	19 717
Number of edges	5 429	4 732	44 338
edges by node (mean)	2.01	1.42	2.25
Max node degree	168	99	171
Min node degree	1	1	1
Oriented	Yes	Yes	Yes
Max score PageRank	$4,92 * 10^{-02}$	$4,01 * 10^{-02}$	$6,10 * 10^{-03}$

The extraction of graph data, the score computation until edge selection and dropping were done before the GCN training on Intel Xeon Processor E5-2690 with 8 cores. The result was used to build G_{drop}. The trainings with G_{drop} were done on Nvidia Tesla V100 PCIe 16GB GPUs. The original GCN, the state of the art DropEdge and our RankedDrop were compared in this section to validate our solution.

3.2 HOW SCAN-WITH-ADD HELPS NODE SELECTION

The values of the SWA of PageRank and Degree are represented graphically in figure 1. We can see that PageRank allows to put forward a small number of nodes; these curves increase very quickly. The distribution of the scores is very unequal: for all the datasets, 10% of the best ranked nodes share more than 80% of the total score, because PageRank highlights the most important nodes in an exponential way. Therefore if the nodes are randomly selected, there is an 80% chance to choose one of the 10% high ranked nodes. By using SWA-PageRank, the best rated nodes of the G graph will be very often integrated to the G' graph, because they are the nodes that have the most impact in the global structure of the graph. On the other hand, the scores of SWA-Degree rise more slowly. The inequality between the nodes is thus less important. The highest degree nodes are privileged in terms of scores but they are not too much highlighted compared to SWA-PageRank. Therefore, we can choose the most adapted SWA and decide to select either the most or the less important nodes to prepare the edge selection.

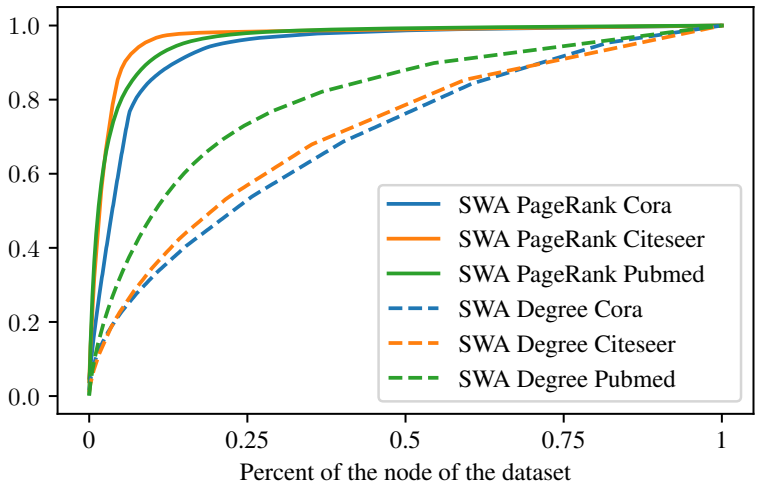
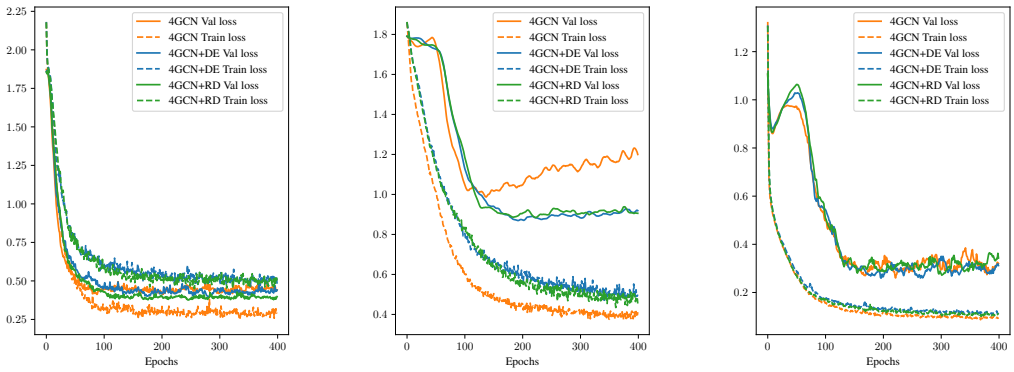


Figure 1: Variation of the Scan-With-Add vectors for Cora, Citeseer and Pubmed datasets for local structure with degrees and global structure with PageRank.

3.3 Impact of RankedDrop on over-fitting



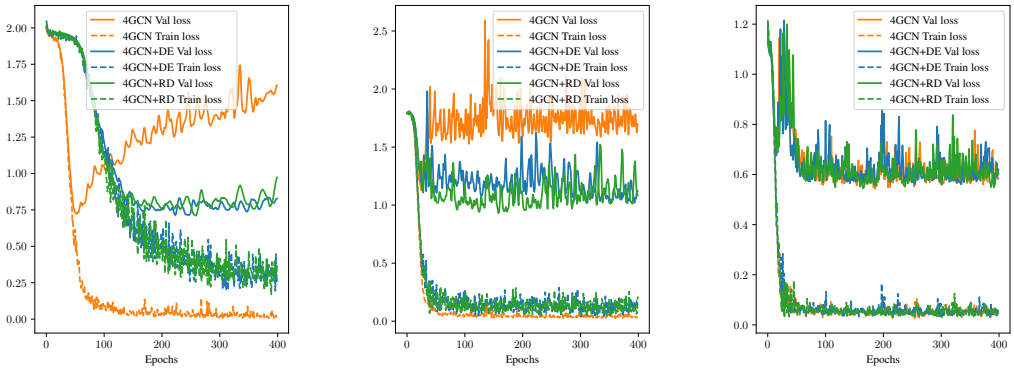
(a) Cora (b) Citeseer (c) Pubmed

Figure 2: Loss payout curve for datasets with the GCN 4 layers architecture in full-supervised learning. Comparaison between the original methode, DropEdge (DE) and RankedDrop (RD).

The figures 2 and 3 show the training and validation loss curves in full-supervised and semi-supervised learning, respectively. All curves for the same dataset were obtained with the same hyper-parameters, only the percentage of dropping edges is different between DropEdge and RankedDrop. We can observe that, the two dropping methods in general have the similar behavior, both have better loss convergence than the original GCN, and in some cases, the validation loss of RankedDrop converges again better than the one of DropEdge. These experiments show that RankedDrop is the best method to reduce the over-fitting phenomenon and to stabilize the loss. RankedDrop has also the same behavior on over-smoothing reduction as DropEdge, so we will not discuss here.

3.4 Impact of dropping control

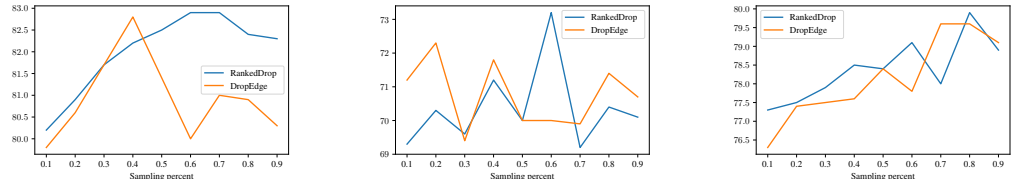
The accuracies after training with different proportion of non-drop edges are shown in figure 4. The accuracies were obtained with the best hyper-parameters for each cases with GCN in the semi-



(a) Cora (b) Citeseer (c) Pubmed

Figure 3: Loss payout curve for datasets with the GCN 4 layers architecture in semi-supervised learning. Comparaison between the original methode, DropEdge (DE) and RankedDrop (RD).

supervised learning. We can observe that for all three datasets the best accuracies are all from RankedDrop. Moreover, the best accuracy obtained by RankedDrop preserve more edges than the best one by DropEdge. For example, for the Citeseer dataset, the best accuracy subgraphs by DropEdge using 20% edges of the original graph, whereas the best accuracy subgraphs by RankedDrop maintain 60% of the edges. We believe the fewer edges was dropped, the more information of the original graph is kept, and we have more chance to achieve a better accuracy.



(a) Cora (b) Citeseer (c) Pubmed

Figure 4: Graphical representation of accuracy results for DropEdge and RankedDrop methods according to the percentage of preserved edges each datasets.

3.5 Overall performance results

3.5.1 Semi-supervised

Table 2: Accuracy comparaison for full-supervised learning methods for GCN architecure

Dataset	Method	2 Layers	4 Layers	8 Layers
Cora	Original	81.10	78.5	31.10
	DropEdge	82.80	78.8	53.1
	RankedDrop	82.90	82.00	63.90
Citeseer	Original	70.80	61.2	30.20
	DropEdge	72.30	68.8	33.20
	RankedDrop	73.20	71.30	45.50
Pubmed	Original	79.00	78.30	61.20
	DropEdge	79.60	77.7	54.50
	RankedDrop	79.90	79.40	77.1

We first compare the accuracy between original GCN, GCN with DropEdge and GCN with RankedDrop in the semi-supervised learning, with 2, 4 and 8 layers (Table 2). The hyper-parameters for 2 layers are from the paper of DropEdge, and the one for 4 and 8 layers are the best one that we found. The parameters used for the selection of the nodes with RankedDrop are available in the appendix A. The accuracies obtained with RankedDrop are all higher than with DropEdge. Moreover, the deeper the GCN is, the better accuracy improvement RankedDrop offers comparing to DropEdge. The accuracy obtained with RankedDrop for the 8-layer GCN is 20% better than the one with DropEdge. Even for the 2-layer GCN, the accuracies of RankedDrop are equivalent or superior to those well-tuned by DropEdge. This is particularly true with the Citeseer dataset where the 2-layer GCN with RankedDrop obtained 1% higher accuracy than with DropEdge.

3.5.2 FULL-SUPERVISED

Table 3: Accuracy comparaison for full-supervised learning methods

Dataset	Type	Original method	DropEdge	RankedDrop
Cora	4 GCN	85.50	85.70	87.60
	8 IncepGCN	86.70	87.70	87.90
	16 JKNet	86.20	87.30	88.40
Citeseer	4 GCN	76.70	79.20	79.90
	8 IncepGCN	79.20	80.5	80.30
	8 JKNet	79.60	80.20	79.8
Pubmed	4 GCN	88.70	90.5	90.70
	4 IncepGCN	89.90	91.10	91.80
	16 JKNet	90.40	91.40	88.30

The accuracies of full-supervised learning are presented in the table 3. For each of the datasets, we evaluated with three different backbones: GCN, IncepGCN and JKNet. The number of layers for each backbone was chosen from the best accuracy declared by DropEdge. We used the same hyper-parameters given by Rong et al. (2019), only the edge dropping percentage is modified for RankedDrop. The accuracies are globally equivalent between RankedDrop and DropEdge; and RankedDrop achieved better accuracies than DropEdge for Cora the smallest dataset. It again show that RankedDrop reduce better the over-fitting phenomenon. Moreover, the hyper-parameters used here are not specifically adapted to RankedDrop, but RankedDrop can still achieve good accuracies. We believe there are still space to increase accuracies with RankedDrop by optimizing the hyper-parameters.

4 CONCLUSION & PERSPECTIVE

The RankedDrop method that we proposed in this paper provided more control on the selection of dropping edges and allows to customize the dropping step for various neural network architectures. Thanks to a personalized score system and the addition of several parameters, the control of the edges to drop is personalized. RankedDrop keeps the advantages of DropEdge concerning the reduction of over-smoothing and over-fitting as well as the possibility to use it on different architectures, while allowing to take into account information on the graph structure. RankedDrop add more control on randomness and a new degree of freedom for dropping selection. We have shown that the results given by RankedDrop are very encouraging and are more stable. The degree of freedom brought by taking into account the structure of the graph allows to project the construction of deeper GNNs.

It is also possible to imagine using this method to better control the training of neural networks on denser graphs. The computations that are performed to extract the data from the graph representation matrix can be executed in distributed computing. The choice of edges to drop at each epoch is more complex to do in distributed computing and will be the subject of future work.

REFERENCES

Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *Proceedings of the 24th international conference on Machine learning*, pp. 9–16, 2007.

Guy E Blelloch. *Vector models for data-parallel computing*, volume 2. MIT press Cambridge, 1990.

Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. Is pagerank all you need for scalable graph neural networks? In *ACM KDD, MLG Workshop*, 2019.

Richard L Burden and J Douglas Faires. 2.1 the bisection algorithm. *Numerical analysis*, pp. 46–52, 1985.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pp. 3438–3445, 2020.

Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. In *Proc. of the 11th Metaheuristics International Conference*, pp. 1–5, 2015.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29:3844–3852, 2016.

David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)*, 44(5):669–696, 1997.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *NIPS 2012 Workshop: Perturbations, Optimization, and Statistics*, 2012.

ZHAO Huan, YAO Quanming, and TU Weiwei. Search to aggregate neighborhood for graph neural network. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 552–563. IEEE, 2021.

Maxime R Hugues and Serge G Petiton. Sparse matrix formats evaluation and optimization on a gpu. In *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, pp. 122–129. IEEE, 2010.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Donald Knuth. Sorting and searching. *The art of computer programming*, 3:513, 1998.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Max O Lorenz. Methods of measuring the concentration of wealth. *Publications of the American statistical association*, 9(70):209–219, 1905.

László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2(1-46):4, 1993.

Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. Learning to drop: Robust graph neural network via topological denoising. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp. 779–787, 2021.

Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

Kenta Oono and Taiji Suzuki. On asymptotic behaviors of graph cnns from dynamical systems perspective. 2019.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Yulong Pei, Tianjin Huang, Werner van Ipenburg, and Mykola Pechenizkiy. Resgcn: attention-based deep residual modeling for anomaly detection on attributed networks. *Machine Learning*, pp. 1–23, 2021.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pp. 2847–2854. PMLR, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

A Appendix: Hyperparameters in experiments

In the table 4 are gathered the parameters used to generate the accuracys that have been presented in the paper. There are both the hyperparameters of the models that are used for the execution of the backbones, and also the few parameters that we used to control the selection of the edges to drop. We have implemented three ways to take into account the information from the structure of the graph. This is the parameter which is named `score`. Either we have used only the degree information or the PageRank information, which is respectively indicated by `Deg` and `PR`, or we have used both at the same time to build the score vector and it is noted `PRxD`. In addition to this parameter, we have influenced the choice of edges to remove from the graph with the following parameters:

- `dd`: It is a boolean that removes the edge in the opposite direction of the selected edge when the dataset is symmetric. Vertices are removed in pairs, and this allows to keep a undirected graph.
- `reverse`: It is a boolean that allows to reverse the adjacency matrix. By doing this, each edge is no longer associated with the tail node but with the head node, and if the scores of the two nodes associated with that edge are not the same, it changes the probability of selecting that particular edge.
- `lowest`: It is a boolean that reverses the ranking of the nodes of the graph using the reciprocal of the score associated to each node.

Ref	Backbone	Dataset	nlayers	Hyper-parameters
Table 2	GCN	Cora	2	lr:0.001, weight-decay:1e-4, sampling-percent:0.7, score:PRxD, dd:false, reverse:true, lowest:true, niter:400
Table 2	GCN	Citeseer	2	lr:0.007, weight-decay:1e-4, sampling-percent:0.6, score:PR, dd:false, reverse:false, lowest:true, niter:400
Table 2	GCN	Pubmed	2	lr:0.009, weight-decay:1e-2, sampling-percent:0.8, score:PR, dd:true, reverse:false, lowest:true, niter:400
Table 2	GCN	Cora	4	lr:0.004, weight-decay:1e-4, sampling-percent:0.3, score:PRxD, dd:false, reverse:true, lowest:true, niter:400
Table 2	GCN	Citeseer	4	lr:0.008, weight-decay:1e-3, sampling-percent:0.1, score:Deg, dd:false, reverse:true, lowest:false, niter:400
Table 2	GCN	Pubmed	4	lr:0.008, weight-decay:1e-2, sampling-percent:0.9, score:Deg, dd:false, reverse:true, lowest:false, niter:400
Table 2	GCN	Cora	8	lr:0.003, weight-decay:1e-5, sampling-percent:0.7, score:PR, dd:true, reverse:true, lowest:false, niter:1000
Table 2	GCN	Citeseer	8	lr:0.001, weight-decay:1e-5, sampling-percent:0.5, score:PRxD, dd:false, reverse:true, lowest:false, niter:1000
Table 2	GCN	Pubmed	8	lr:0.006, weight-decay:1e-4, sampling-percent:0.5, score:PRxD, dd:true, reverse:true, lowest:true, niter:1000
Table 3	GCN	Cora	4	lr:0.01, weight-decay:0.005, sampling-percent:0.6, score:Deg, dd:true, reverse:true, lowest:true, niter:400
Table 3	GCN	Citeseer	4	lr:0.009, weight-decay:1e-3, sampling-percent:0.1, score:Deg, dd:true, reverse:true, lowest:false, niter:400
Table 3	GCN	Pubmed	4	lr:0.01, weight-decay:1e-3, sampling-percent:0.2, score:PRxD, dd:true, reverse:true, lowest:false, niter:400
Table 3	IncepGCN	Cora	8	lr:0.01, weight-decay:1e-3, sampling-percent:0.1, score:PR, dd:true, reverse:false, lowest:true, niter:400
Table 3	IncepGCN	Citeseer	8	lr:0.002, weight-decay:0.005, sampling-percent:0.1, score:Deg, dd:true, reverse:true, lowest:false, niter:400
Table 3	IncepGCN	Pubmed	4	lr:0.002, weight-decay:1e-5, sampling-percent:0.3, score:PRxD, dd:false, reverse:true, lowest:true, niter:400
Table 3	JKNet	Cora	16	lr:0.008, weight-decay:5e-4, sampling-percent:0.1, score:PR, dd:true, reverse:true, lowest:true, niter:400
Table 3	JKNet	Citeseer	8	lr:0.004, weight-decay:5e-5, sampling-percent:0.8, score:PR, dd:true, reverse:true, lowest:true, niter:400
Table 3	JKNet	Pubmed	64	lr:0.005, weight-decay:1e-4, sampling-percent:0.9, score:Deg, dd:false, reverse:false, lowest:false, niter:400

Table 4: Hyper-parameters used to obtain the accuracy presented in this paper with the RankedDrop method.