Graph Convolutional Networks Modifications

Mikhail Salnikov [1] Anastasia Remizova [1] Aleksei Kalinov [1] Mark Griguletskii [1] Igor Markov [1]

Abstract

The project focuses on improving the Graph Convolutional Networks (GCNs) performance. These networks became a quite common method for graph representation learning and are widely used in a variety of real-world applications. However, GCNs are typically shallow, with the number of layers not more than two. Greater depth usually leads to network overfitting, oversmoothing or even an inability to learn, where the application of standard methods like dropout and weight penalizing does not help. In this work we explore certain techniques that allow the construction of deeper networks and preserve the ability of network to train effectively.

1. Introduction

Graph Convolutional Networks(GCNs) are becoming one of the most crucial tools for graph representation learning. These neural networks are widely used in a variety of real-world applications like social recommendation systems, link prediction, node classification, and many others. However, the ability of deep graph nets to fit a variety of complex real-world datasets becomes its potential weakness. Due to the huge amount of neurons and layers, deep graph neural networks become so closely fit to the training set that it becomes impossible to generalize and make predictions for new data. That is why existing GCNs are typically shallow. Nowadays, there is an intensive research work conducted to find the best way to overcome the issue of overfitting in deep GCNs. For instance, in the work (Rong et al., 2020) authors propose a novel technique to alleviate overfitting and over-smoothing of the GCN called DropEdge. The key idea of this method is to randomly remove a certain number of edges from the input graph at each training epoch, acting as a data augmenter and also a message-passing reducer. While the authors achieve state-of-the-art results, their networks are still quite shallow, most of the time restricted to 2 layers. This work is chosen as the backbone for our experiments as we explore other classic deep learning techniques aimed at better training and generalization at a higher number of layers, such as, skip connections and orthogonalization of weights, as well as a more numerically intensive approach of weight matrix singular values modification.

The main contributions of this report are as follows:

- We provide a detailed analysis of singular value characteristics of network weights, perform several experiments to decrease overfitting by modification of singular values and discuss achieved results.
- We demonstrate that skip-connection technique can be effective at preserving generalization ability of a graph convolutional network with many layers.
- The number of approaches that did not yield any gain in generalization ability are presented in order to demonstrate the peculiarities of the problem and discourage researchers from chasing unpromising hypotheses.

2. Related work

2.1. Graph Convolutional Networks

Finding their influence in image processing, Graph Convolutional Networks attempt to apply notion of convolution on the graphs with arbitrary topology. The work on this topic can be roughly categorized into three main research directions. Pioneered in (Bruna et al., 2013), the first approach uses spectral graph theory (Kipf & Welling, 2016) to analyze and make inference on graphs. The main drawback of such approach was their poor scalability. To address the issue spatial-base CGNs were developed (Hamilton et al., 2017). Due to the way these methods directly aggregate information from neighbours, they might be incapable of capturing information from more distant nodes. Sampling approaches aim to achieve fast graph structure learning by various node and edge sampling procedures. DropEdge (Rong et al., 2020) is a most recent example of such approach where a random number of edges is dropped at each epoch to promote better resilience to different graph topologies.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Mikhail Salnikov <m.salnikov@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

2.2. Orthogonalization of weights

A global convergence of neural networks is a complex theoretical and practical question, concerning weight dynamics, structures of networks and valleys of hyperparameters. One of directions that researchers pursue is the choice of initial values for weights. Orthogonal initialization (Saxe et al., 2013) has shown to be a crucial step to train deep networks (Xiao et al., 2018) as the weights with absolute magnitude of singular values equal to 1 discourage growing or fading of gradients and speed up convergence (Hu et al., 2020). Keeping weights orthogonal during training has shown contradictory results (Vorontsov et al., 2017), (Brock et al., 2016). In this report we explore the topic further.

3. Algorithms & Models

The algorithms and models in use could be found in the github repository by link `https://github.com/MihailSalnikov/svd4gcn` with dependency on a modified DropEdge model `http://github.com/mousebaiker/DropEdge`. All the algorithms used in the work are the variations of the DropEdge Graph convolutional neural network.

3.1. Graph Convolutional Networks

Typical graph convolutional network (GCN) consists of multiple layers of convolutions on graph nodes feature matrix X. This matrix captures information about each node in a graph. The convolution is performed with respect to graph adjacency matrix A. The most typical formulation of a layer is written as a non-linear function:

$$H^{l+1} = f(H^l, A) = \sigma(AH^lW^l), \tag{1}$$

where W^l is a weight matrix for the l-th neural layer and σ is a non-linear function like the ReLU. However, this layer-wise propagation rule is usually modified because of two limitations. First, we add the identity matrix to A to enforce self loops in the graph. Second, we need to normalize A to ensure that multiplication with A does not change the scale of the feature vectors.

$$f(H^l, A) = \sigma(D^{-0.5}(A+I)D^{-0.5}H^lW^l), \tag{2}$$

where D is a diagonal node degree matrix of $A + I$.

In order to capture node information X, the convolution in the first layer is perfomed on it, i.e. $H^1 = X$. The final output H_k, where k is number of layers in the convolutional model can be treated directly as a set of logits for each node by appropriately setting the dimensions of W^k. They can also be treated as extracted features and passed further down the pipeline to any differentiable classifier. In our case, the additional classifier consists of two fully-connected layers.

3.2. DropEdge

DropEdge technique applies the graph edge subsampling on every epoch. Some edges are discarded and the training is performed on some subsample of the graph edges. The term "DropEdge" refers to randomly dropping out certain rate of edges of the input graph for each training time. There are several benefits in applying DropEdge for the GCN training. First, DropEdge can be considered as a data augmentation technique. By DropEdge, we are actually generating different random deformed copies of the original graph; as such, we augment the randomness and the diversity of the input data, thus better capable of preventing over-fitting. Second, DropEdge can also be treated as a message passing reducer. In GCNs, the message passing between adjacent nodes is conducted along edge paths. Removing certain edges is making node connections more sparse, and hence avoiding over-smoothing to some extent when GCN goes very deep. DropEdge either reduces the convergence speed of over-smoothing or relieves the information loss caused by it.

3.2.1. Truncated SVD

We try to apply SVD on all fully-connected layers of our model, except the output one due to its already small rank. This approach is also called low-rank approximation or truncated SVD. It takes a layer and decomposes it into several smaller layers, thus decreasing number of weights and total number of floating-point operations. SVD exists for any matrix, and it can be factorized as follows:

$$\underset{n\times m}{A} = \underset{n\times n}{U} \cdot \underset{n\times m}{S} \cdot \underset{m\times m}{V^\top}, \tag{3}$$

where S is a diagonal matrix with non-negative decreasing singular values on its diagonal. U and V are orthogonal matrices composed of singular vectors. If we take the r largest singular values and zero out the rest, we get the best low-rank approximation of A in terms of both Frobenius and spectral norms:

$$A_r = U_r S_r V_r^T, \tag{4}$$

The matrix A_r is the r-rank matrix, which has both the Frobenius and the spectral norm closest to those of A. The fully-connected layer does the matrix multiplication of its input by a matrix A and adds a bias b. We just take the SVD of A and keep only the first r singular values.

$$Ax + b = (U_r S_r V_r^T)x + b, \tag{5}$$

We supposed that truncated SVD could help to prevent over-fitting. For example, we know that application of Principal Component Analysis to the data can improve the quality of

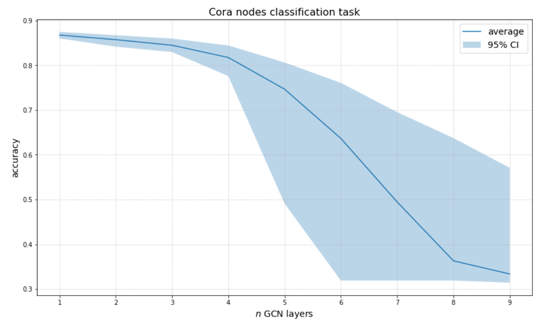

Figure 1. Accuracy of the GCN model on Cora dataset with increase of the number of layers

the model because it solves multicollinearity problem. Low-rank approximation on weights can do something similar — reduce variable collinearity between weights. Besides, after application of truncated SVD we have fewer parameters in the model which is often the core of overfitting problem.

3.3. Orthogonalization of weights

Orthogonal matrices W are the ones that have their columns orthogonal to each other, i.e. $W^T W = I$. Their spectrum follow a nice property of being a unit circle in a complex plane. If we assume no non-linearity and bias between layers, than the product $\widehat{W} = \prod_{i=1}^{k} W^k$ is essentially computed. If matrices W^k are orthogonal, then the resulting $\widehat{W}$ is also orthogonal and norm preserving. While Gaussian initialization of matrices W^k that can also be done in a norm-preserving manner, the resulting spectrum of $\widehat{W}$ is much less uniform with many singular values tending to zero. Thus, multiplication by such matrix is essentially equal to projection on very low-dimensional manifold. A more detailed and formal analysis can be found in (Saxe et al., 2013). We try using orthogonal initialization of weights for DropEdge model and report our findinding below.

Instead of just initializing weights with orthogonal matrices, it is possible to constrain them to conform to property of orthogonality during the whole training procedure. Specifically, a regularizer of type $\|W^T W - I\|^2$ is added to the loss function for each weight in the network. We conduct experiments with such modification and state the results in the further section.

3.4. Skip connections

Deep neural networks often show the following behaviour. When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly, as can be seen in 1. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution by construction to the deeper model: the added layers are identity mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart, as in (He et al., 2015). Implementing these skip layer connections allows the network to preserve the information from hidden layer and, as we believe, lower the accuracy saturation.

3.5. Spectral normalization

Introduced in (Miyato et al., 2018) for Generative Adversarial Networks, spectral normalization is a weight normalization technique aimed at weight stabilization during training. In its essence the normalization of weight is done by its largest singular value $\tilde{W} = \dfrac{W}{\sigma(W)}$. As stated in the original paper, this normalization allows to get Lipschitz-continuity bounds on the layer transformations that guarantee stable training. Given that the truncated SVD procedure already computes the full spectrum of weight matrix, this normalization is effectively done for free in terms of computational efficiency.

4. Experiments and Results

4.1. Datasets

The Pubmed Diabetes dataset consists of 19717 scientific publications from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44338 links. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words. The README file in the dataset provides more details. It can be downloaded from `https://linqs-data.soe.ucsc.edu/public/Pubmed-Diabetes.tgz`. The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words. It can be downloaded from `https://relational.fit.cvut.cz/dataset/CORA`. The CiteSeer dataset consists of 3312 scientific publications classified into one of six classes. The citation network consists of 4732 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The

Table 1. Parameters for training GCN model. Fixed for all experiments as no substaintial gain has been found with hyper-parameter search.

Parameter	Value
dataset	{pubmed, cora, citeseer}
# hidden layers	variable
size of hidden features	128
# epochs	400
optimizer	Adam
learning rate	0.01
weight_decay	0.005
dropout	0.8

dictionary consists of 3703 unique words. It can be downloaded from `https://linqs-data.soe.ucsc.edu/public/lbc/citeseer.tgz`. The computations were performed on consumer grade GPU and CPU.

The datasets have public split into training and test batches which we strictly follow. Additionally, training set is split into training and validation categories with latter having 500 random nodes and the former being assigned the rest. Citeseer dataset contains some isolated nodes in the graph. We include them in both training and test sets as nodes without any neighbours.

The main metric agreed upon for these datasets and used throughout the experiment section is accuracy or a rate of correctly predicted classes in the whole set.

As described in the former section, adjacency matrix of each graph is modified to include self-loops and is normalized to preserve the norms of features.

4.2. Training parameters

DropEdge implementation was distributed with default sane hyper-parameters. We performed hyper-parameter search during several experiments described below and found no substantial gain in accuracy. Thus, we fix the parameters for the whole duration of work. Table (1) summarizes the resulting parameter set. For all the techniques we explored in our project, we ran the validation tests 3 times to eliminate the dependency on the random seed. We ran the validation tests 100 times for the techniques that involve applying SVD decomposition.

4.3. SVD tricks

4.3.1. Evaluation of properties

To understand network properties more thoroughly, the following analysis was performed. The model of consideration is sequential multilayer GCN (without DropEdge technique) with input layer, several hidden layers and output layer, all

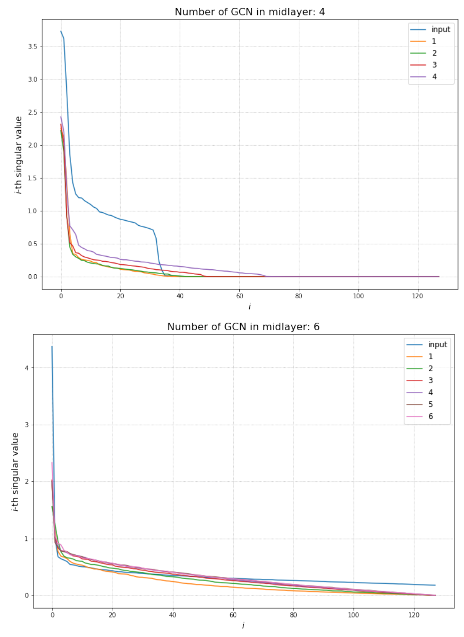

Figure 2. Singular values plots for cases of successfully trained and failed to train networks.

of them are GCLs.

Let l be an index of GCL. SVD decomposition was applied to the weight matrix $W^l \in R^{n \times m}$ of this layer. Then, for a fixed r the weight matrix W^l was replaced with truncated SVD approximation of rank r. After that, singular values, loss and accuracy of the modified model was estimated on the test dataset.

The described procedure was performed for each r from 1 to the $\min(n, m)$ and for each GCL including input layer, but excluding output layer.

This experiment was performed on Cora dataset.

Singular values can be seen on the Figure 2. In the first case, with 4 layers, the network was able to learn and achieved accuracy 0.791 on the test set. In the second, with 6 layers, it failed with accuracy of only 0.374.

The patterns are similar for other number of layers. For successfully trained networks, input layer singular values form ”ladder step” and after the ”step”, they are nearly zero. Other layers singular values are also close to zero from some point. Besides, singular values of the layer are generally higher with the relative number of layer. Therefore, rank can be reduced without significant changes of the network weights.

In the case of network which failed to learn, all plots of singular values have similar form. There are less values close to zero, especially in case of the input layer.

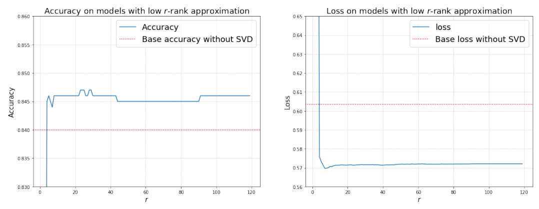

Figure 3. Accuracy and loss scores of the model with SVD.

4.3.2. Accuracy with SVD

Inception GCN model with default parameters was used.

Original loss and accuracy scores after 400 epochs are 0.6 and 0.84 respectively.

The choice of layers for approximation is a creative process and requires a large number of experiments. For the chosen model we found the best solution is truncated Singular Value Decomposition approximation with rank 3 (instead of 128) on internal GCN layers in the Inception module. Quality of model with the low-rank approximation on "ingc" layer (input layer) is very sensitive to r, where r – rank. Different results for different r are represented in figure **??**.

However, for great number of layers, SVD does not really help to fight overfitting, therefore, other techniques were used.

4.3.3. Fine tuning after low rank approximation

During the working on project was implemented Fine Tuning (FT) after low rank approximation in three ways. First - FT all network on training set after approximation some layers. Second - FT all network, but approximate layers layer by layer, firstly we approximate first hidden layer and FT it on a few epochs, after that approximate second layer and etc. Third - FT only approximated layers.

Results for the first way is a small improving accuracy on test set, similar to above. Accuracy for send way is similar to first, but we have interesting but not amazing FT history. After approximation each layer, loss dramatically increased and accuracy dramatically decreased (figure 4).

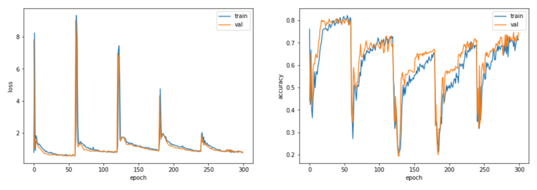

Figure 4. Second FT way accuracy and loss training history

Third way (figure 5) allow us to increase score for GCN network on Citeseer, CORA and pubmed datasets too, but not more that first and second way.

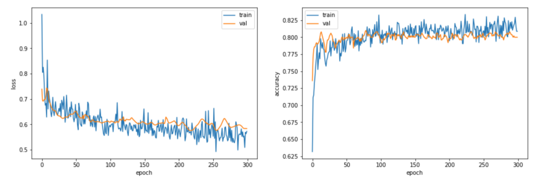

Figure 5. Third FT way accuracy and loss training history

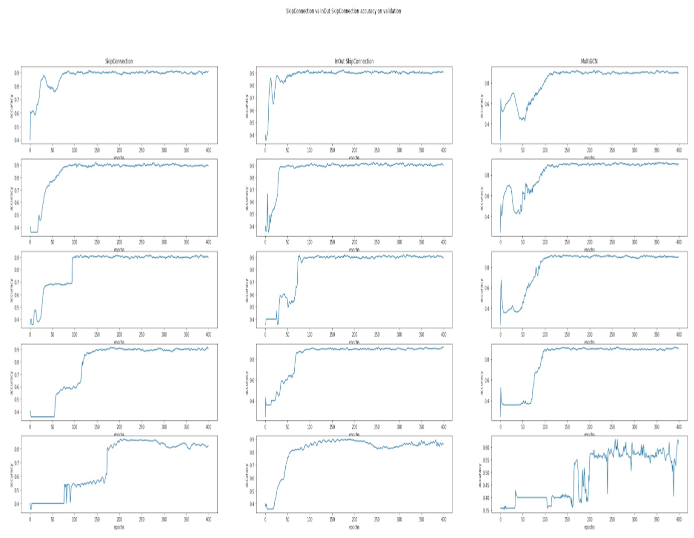

Figure 6. Validation accuracy for SC, InoutSC and vanilla Multi-GCN

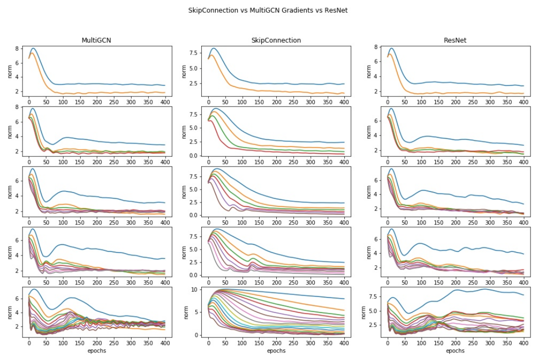

Figure 7. Gradient norms for 2, 4, 6, 8, 16, layers from top to bottom for MultiGCN, SC and ResNet

4.4. Orthogonalization of weights

In order to preserve the rank of weight matrices on each layer, we apply the weight orthogonalization procedure. We tried two main approaches: initializing the weights as an orthogonal matrix, and modifying the loss in order to force the orthogonality.

To conduct the experiment we initialized each weight W^l with random orthogonal matrix, trained the network and compared the accuracy with the deafault implementation which uses uniform distribution to initialize each element of matrix independently. As for vectors the notion of orthogonality does not make any sense, biases in the network are initialized with element-wise uniform distribution scaled by the number of features in the current layer.

The accuracy was compared on three main datasets described above. The results are presented in table 2. It is immediately obvious that initialization does not help train better shallow networks. While it may seem that orthogonal initialization helps in deeper networks, the fact that the standard deviation of these results is $\approx 15\%$ for networks with 10 and 16 layers makes the results insignificant.

Table presents result when orthogonal weight regularizer has been used. It seems that on shallow networks it can be useful as the results are better and the resulting variance is small. Note that adding orthogonal initialization in conjunction with weight forcing does not improve results.

Table 2. Ablation study for orthogonal initialization. *init* indicates results for model with orthogonal initialization. *no_init* results with uniform initialization. Accuracy on test set is reported in percent. Standard deviation of results for 10 and 16 layers is $\approx 15\%$.

	Citeseer		Cora		Pubmed	
layers	init	no_init	init	no_init	init	no_init
2	56.3	77.6	86.3	85.8	89.9	75.2
4	52.3	56.2	80	79.3	79.8	89.5
8	28.7	26.9	31.9	31.9	86.1	83.1
10	26.8	27.3	31.9	31.9	88.1	81.1
16	30.1	29	31.9	31.9	75.9	68.6

Table 3. Orthogonal weight forcing. *force* indicates model with weight forcing. *force + init* — results with weight forcing and orthogonal initilaization. Accuracy on test set is reported in percent.

layers	force	force + init
3	85.5	85.5
4	84.0	83.9

4.5. Skip connection

Skip GCN network effectively is a modification of MultiGCN network. For the regular Skip Connection (SC) the output layer is changed into the combination of two linear

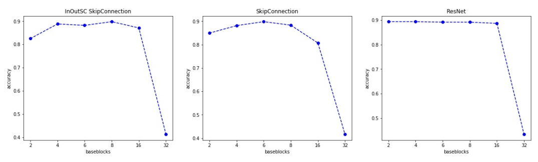

Figure 8. Comparison of accuracy for InOut SC, SC and ResNet

layers with ReLU. The outputs of each hidden layer are concatenated into the one tensor which is later provided to the output layer which has the aforementioned structure. For the InOut SC the structure is the same, but the output layer is only provided the original input and the output of the last hidden layer. The results of the implementation of skip connections were promising, as we were able to achieve accuracy not lower than the original architecture, but also allowed the construction of the deeper architectures with much lesser weight saturation. The accuracies are illustrated in 6 One can observe that the InOut skip connection achievs the top accuracy already on 50th-60th epoch, while the vanilla MultiGCN and SC require at least a 100 epochs. Skip connections also allow for the stabler gradients which vanish much slower (7). Final comparison in accuracy between the two implementations of SC and pre-implemented ResNet can be seen in 8. This figure illustrates that, although our implementations of SC start to degrade a little faster, the top accuracy is a bit better than the ResNet.

4.6. Skip connection and orthogonal initialization

4.6.1. Truncated SVD application

The experiment was conducted on SkipGCN with default parameters. Only number of layers was varied.

For each r_{inp} from 1 to rank of the input layer size and r_{mid} from 1 to rank of the middle layer sizes, truncated SVD was applied to corresponding layers weights and their self-loop weights. Then, accuracy was evaluated on the test set.

While more experiments should be conducted, results on Figure 9 are promising. Each plot corresponds to each r_1; the points of it are accuracies with chosen ranks of r_{inp} and r_{mid}.

Often the better accuracy is achieved with lower rank the original, even much lower such as rank $r_{mid} = 3$ instead of original 128 for all hidden layers (e.g in case of 10 layers). Application of truncated SVD with small rank also reduces used memory and inference time.

4.7. Spectral normalization

To conduct experiments with spectral normalization we modified the training procedure of DropEdge to divide the weight matrix by the largest singular value during each iter-

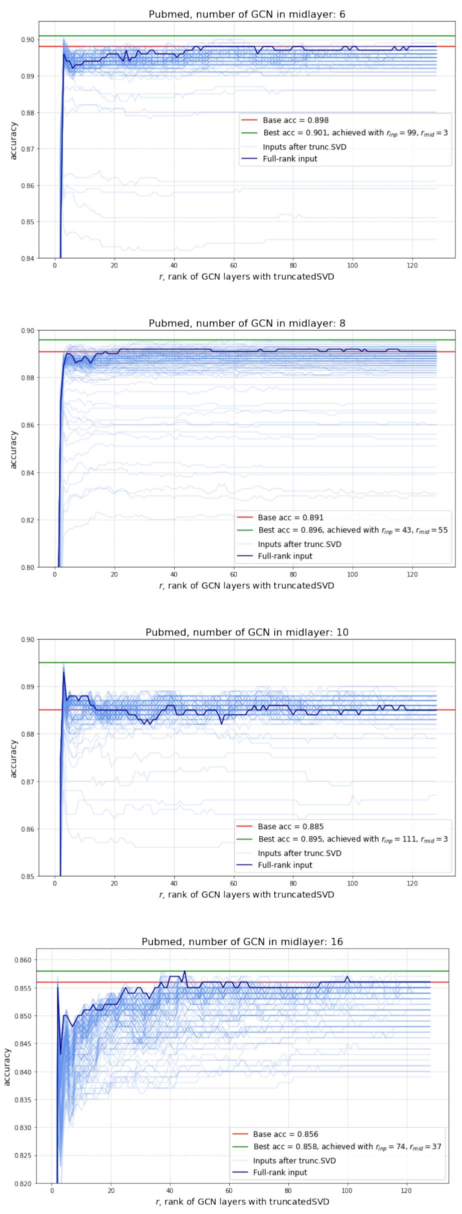

Figure 9. Accuracy of SkipGCN with TruncatedSVD applied to weights.

ation. The model is firstly trained on pubmed dataset. Initial results showed that the network is unable to train completely after this modification. The division procedure was changed to be performed once every epoch with no success as initial division brought many singular values of weights close to zero, so that most dimensions became useless after normalization. We do not report any score as it is no better than a random baseline.

5. Conclusion

This report has reviewed several techniques to improve generalization ability of deep graph convolutional networks. We have found that skip connections showed great potential in enabling the graph convolutional networks to be built deeper and stay stable for increased number of layers. A specific implementation of SC, mainly InOut SC also showed an increase in the training time, as it was able to achieve the top results for the architecture on a smaller number of epochs. Truncated SVD also showed promising results and allowed to increase accuracy for the shallow network, but it failed to produce sustainable results for deeper networks. Additionally, our results suggest that orthogonal initialization of weights and spectral normalization are not applicable to the current problem as they did not yield as promising results as were expected.

Overall, the obtained results are a solid ground for future research, as they provide a stable starting point for the further exploration of the problem. They do not, however, solve it fully, but rather show a few small but noticeable improvements.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Mikhail Salnikov (20% of work)

- Reviewing literate on the main topics.
- Experiments with truncated SVD and fine tuning after truncated SVD.
- Implementation and experiments with orthogonal forcing.
- Implementation and experiments with orthogonal weights initialization.
- Unsuccessfully experiments with Singular Normalization.

Anastasia Remizova(20% of work)

- Analyzed literature on graph convolutional networks.
- Implemented TruncatedSVD layer.
- Evaluated properties of TruncatedSVD application to networks.
- Applied TruncatedSVD techniques to GCN with skip connection.

Aleksei Kalinov(20% of work)

- Analyzed literature on initialization, weight orthogonalization and spectral normalization in networks.
- Implemented orthogonal initialization for DropEdge.
- Implemented an automated system to run experiments for orthogonal initialization and skip connections on three datasets, to extract the results and to present them in a human-readable format.
- Conducting experiments for orthogonal initialization with different hyperparameters.
- Writing “Related work”, “Weight orthogonalizaion” and “Spectral normalization” parts of the report.
- Unsuccessfully attempted to run experiments with Lanczos Network from authors' Github repository.

Mark Griguletskii(20% of work)

- Implemented Skip Connection algorithm with vanilla GCN.
- Implemented InOut Skip Connection algorithm.
- Made all plots dedicated to Skip Connection part of project.
- Reviewed literature, got lots of new knowledge.

Igor Markov(20% of work)

- Implemented Skip Connection with GCN.
- Conducted experiments on comparing Skip Connection and InOut SC performance with vanilla GCN with various parameters
- Analyzed existing literature and implementations of Skip Connection in Graph Convolutional Networks
- Unsuccessfully tried to reproduce the method of building deep GCNs found online in `https://github.com/lightaime/deep_gcns_torch`
- Analyzed articles on GCNs.
- Recorded the video of presentation.
- Worked on the report.

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment:

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: We base our research on an existing architecture DropEdge, which we referenced throughout the report.

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: See Models & Algorithms section.

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: See Dataset section.

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: See Datasets section.

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: As we base our research on existing experiments on popular datasets, the training, validation and testing samples were allocated by original authors.

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: See Training Parameters section.

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: See Experiments section.

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: Accuracy metric is self-explanatory, for gradient norm see Skip Connection section.

12. Clearly defined error bars are included in the report.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: See Orthogonalization section.

13. A description of the computing infrastructure used is included in the report.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: See Datasets section.

References

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 1024–1034. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Hu, W., Xiao, L., and Pennington, J. Provable benefit of orthogonal initialization in optimizing deep linear networks. *arXiv preprint arXiv:2001.05992*, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3570–3578. JMLR. org, 2017.

Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S. S., and Pennington, J. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. *arXiv preprint arXiv:1806.05393*, 2018.