Appendix—Graph Random Neural Networks for Semi-Supervised Learning on Graphs

Contents

A Reproducibility.............................................1
  A.1 Datasets Details........................................1
  A.2 Implementation Details...............................1
  A.3 Hyperparameter Details..............................2

B Theorem Proofs...........................................3
  B.1 Proof for Theorem 1..................................3
  B.2 Proof for Theorem 2..................................3

C Additional Experiments.................................4
  C.1 Results on Large Datasets............................4
  C.2 Efficiency Analysis................................4
  C.3 Parameter Sensitivity................................4
  C.4 DropNode vs Dropout..................................5
  C.5 GRAND vs. GRAND_GCN & GRAND_GAT..............6
  C.6 Performance of GRAND under different label rates........6

A Reproducibility

A.1 Datasets Details

Table 2 summarizes the statistics of the three benchmark datasets — Cora, Citeseer and Pubmed. Our preprocessing scripts for Cora, Citeseer and Pubmed is implemented with reference to the codes of Planetoid [10]. We use exactly the same experimental settings—such as features and data splits—on the three benchmark datasets as literature on semi-supervised graph mining [10, 5, 9] and run 100 trials with 100 random seeds for all results on Cora, Citeseer and Pubmed reported in Section 4. We also evaluate our method on six publicly available and large datasets, the statistics and results are summarized in Appendix C.1.

A.2 Implementation Details

We make use of PyTorch to implement GRAND and its variants. The random propagation procedure is efficiently implemented with sparse-dense matrix multiplication. The codes of GCN and
Table 2: Benchmark Dataset statistics.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Train/Valid/Test Nodes</th>
<th>Classes</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>2,708</td>
<td>5,429</td>
<td>140/500/1,000</td>
<td>7</td>
<td>1,433</td>
</tr>
<tr>
<td>Citeseer</td>
<td>3,327</td>
<td>4,732</td>
<td>120/500/1,000</td>
<td>6</td>
<td>3,703</td>
</tr>
<tr>
<td>Pubmed</td>
<td>19,717</td>
<td>44,338</td>
<td>60/500/1,000</td>
<td>3</td>
<td>500</td>
</tr>
</tbody>
</table>

GRAND_GCN are implemented referring to the PyTorch version of GCN. As for GRAND_GAT and GAT, we adopt the implementation of GAT layer from the PyTorch-Geometric library in our experiments. The weight matrices of classifier are initialized with Glorot normal initializer. We employ Adam to optimize parameters of the proposed methods and adopt early stopping to control the training epochs based on validation loss. Apart from DropNode (or dropout) used in random propagation, we also apply dropout on the input layer and hidden layer of the prediction module used in GRAND as a common practice of preventing overfitting in optimizing neural network. For the experiments on Pubmed, we also use batch normalization to stabilize the training procedure. All the experiments in this paper are conducted on a single NVIDIA GeForce RTX 2080 Ti with 11 GB memory size. Server operating system is Unbuntu 18.04. As for software versions, we use Python 3.7.3, PyTorch 1.2.0, NumPy 1.16.4, SciPy 1.3.0, CUDA 10.0.

A.3 Hyperparameter Details

Overall Results in Section 4.2. GRAND introduces five additional hyperparameters, that is the DropNode probability $\delta$ in random propagation, propagation step $K$, data augmentation times $S$ at each training epoch, sharpening temperature $T$ when calculating consistency regularization loss and the coefficient of consistency regularization loss $\lambda$ trading-off the balance between $L_{sup}$ and $L_{con}$. In practice, $\delta$ is always set to 0.5 across all experiments. As for other hyperparameters, we perform hyperparameter search for each dataset. Specifically, we first search $K$ from \{2,4,5,6,8\}. With the best selection of $K$, we then search $S$ from \{2,3,4\}. Finally, we fix $K$ and $S$ to the best values and take a grid search for $T$ and $\lambda$ from \{0.1, 0.2, 0.3, 0.5\} and \{0.5, 0.7, 1.0\} respectively. For each search of hyperparameter configuration, we run the experiments with 20 random seeds and select the best configuration of hyperparameters based on average accuracy on validation set. Other hyperparameters used in our experiments includes learning rate of Adam, early stopping patience, L2 weight decay rate, hidden layer size, dropout rates of input layer and hidden layer. We didn’t spend much effort to tune these hyperparameters in practice, as we observe that GRAND is not very sensitive with those. Table 3 reports the best hyperparameters of GRAND we used for the results reported in Table 1.

Table 3: Hyperparameters of GRAND for results in Table 1.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>DropNode probability $\delta$</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Propagation step $K$</td>
<td>8</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Data augmentation times $S$</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>CR loss coefficient $\lambda$</td>
<td>1.0</td>
<td>0.7</td>
<td>1.0</td>
</tr>
<tr>
<td>Sharpening temperature $T$</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.01</td>
<td>0.01</td>
<td>0.2</td>
</tr>
<tr>
<td>Early stopping patience</td>
<td>200</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>Hidden layer size</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>L2 weight decay rate</td>
<td>5e-4</td>
<td>5e-4</td>
<td>5e-4</td>
</tr>
<tr>
<td>Dropout rate in input layer</td>
<td>0.5</td>
<td>0.0</td>
<td>0.6</td>
</tr>
<tr>
<td>Dropout rate in hidden layer</td>
<td>0.5</td>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Robustness Analysis in Section 4.5. For random attack, we implement the attack method with Python and NumPy library. The propagation step $K$ of GRAND (with or without CR) is set to 5. And the other hyperparameters are set to the values in Table 3. As for Metattack, we use the publicly available implementation published by the authors with the same hyperparameters used in the experiments. 

https://github.com/tkipf/pygcn
https://pytorch-geometric.readthedocs.io
https://github.com/danielzuegner/gnn-meta-attack
the original paper. We observe GRAND (with or without CR) is sensitive to the propagation step \( K \) under different perturbation rates. Thus we search \( K \) from \( \{5, 6, 7, 8\} \) for each perturbation rate. The other hyperparameters are fixed to the values reported in Table 3.

Other Experiments. For the other results reported in Section 4.2–4.6, the hyperparameters used in GRAND are set to the values reported in Table 3 with one or two changed for the corresponding analysis.

Baseline Methods. For the results of GCN or GAT reported in Section 4.5–4.6, the learning rate is set to 0.01, early stopping patience is 100, L2 weight decay rate is 5e-4, dropout rate is 0.5. The hidden layer size of GCN is 32. For GAT, the hidden layer consists 8 attention heads and each head consists 8 hidden units.

B Theorem Proofs

B.1 Proof for Theorem 1

Proof. The expectation of \( \mathcal{L}_{con} \) is:

\[
\frac{1}{2} \sum_{i=0}^{n-1} \mathbb{E} \left[ (z_i^{(1)} - \tilde{z}_i^{(2)})^2 \right] = \frac{1}{2} \sum_{i=0}^{n-1} \mathbb{E} \left[ (z_i^{(1)} - (z_i^{(2)} - z_i))^2 \right].
\]

(7)

Here \( z_i = \text{sigmoid}(\overline{A}_i \cdot \overline{X} \cdot \mathbf{W}) \), \( \tilde{z}_i = \text{sigmoid}(\overline{A}_i \cdot \tilde{X} \cdot \mathbf{W}) \). For the term of \( \tilde{z}_i - z_i \), we can approximate it with its first-order Taylor expansion around \( \overline{A}_i \cdot \overline{X} \cdot \mathbf{W} \), i.e., \( \tilde{z}_i - z_i \approx z_i (1 - z_i) (\overline{A}_i (\overline{X} - \overline{X}) \cdot \mathbf{W}) \). Applying this rule to the above equation, we have:

\[
\frac{1}{2} \sum_{i=0}^{n-1} \mathbb{E} \left[ (\tilde{z}_i^{(1)} - \tilde{z}_i^{(2)})^2 \right] \approx \frac{1}{2} \sum_{i=0}^{n-1} z_i^2 (1 - z_i)^2 \mathbb{E} \left[ (\overline{A}_i (\overline{X}^{(1)} - \overline{X}^{(2)}) \cdot \mathbf{W})^2 \right] = \sum_{i=0}^{n-1} z_i^2 (1 - z_i)^2 \text{Var}_e \left( \overline{A}_i \overline{X} \cdot \mathbf{W} \right).
\]

\( \blacksquare \)

B.2 Proof for Theorem 2

Proof. Expanding the logistic function, \( \mathcal{L}_{org} \) is rewritten as:

\[
\mathcal{L}_{org} = \sum_{i=0}^{m-1} \left[ -y_i \overline{A}_i \cdot \overline{X} \cdot \mathbf{W} + \mathcal{A}(\overline{A}_i, \overline{X}) \right],
\]

(9)

where \( \mathcal{A}(\overline{A}_i, \overline{X}) = -\log \left( \frac{\exp(-\overline{X} \cdot \mathbf{W})}{\exp(-\overline{X} \cdot \mathbf{W})} \right) \). Then the expectation of perturbed classification loss can be rewritten as:

\[
\mathbb{E}_e (\mathcal{L}_{sup}) = \mathcal{L}_{org} + \mathcal{R}(\mathbf{W}),
\]

(10)

where \( \mathcal{R}(\mathbf{W}) = \sum_{i=0}^{m-1} \mathbb{E}_e \left[ \mathcal{A}(\overline{A}_i, \overline{X}) - \mathcal{A}(\overline{A}_i, \overline{X}) \right] \). Here \( \mathcal{R}(\mathbf{W}) \) acts as a regularization term for \( \mathbf{W} \). To demonstrate that, we can take a second-order Taylor expansion of \( \mathcal{A}(\overline{A}_i, \overline{X}) \) around \( \overline{A}_i \cdot \overline{X} \cdot \mathbf{W} \):

\[
\mathbb{E}_e \left[ \mathcal{A}(\overline{A}_i, \overline{X}) - \mathcal{A}(\overline{A}_i, \overline{X}) \right] \approx \frac{1}{2} \mathcal{A}''(\overline{A}_i, \overline{X}) \text{Var}_e \left( \overline{A}_i \overline{X} \cdot \mathbf{W} \right).
\]

(11)

Note that the first-order term \( \mathbb{E}_e \left[ \mathcal{A}'(\overline{A}_i, \overline{X})(\overline{X} - \overline{X}) \right] \) vanishes since \( \mathbb{E}_e(\overline{X}) = \overline{X} \). We can easily check that \( \mathcal{A}''(\overline{A}_i, \overline{X}) = z_i (1 - z_i) \). Applying this quadratic approximation to \( \mathcal{R}(\mathbf{W}) \), we get the quadratic approximation form of \( \mathcal{R}(\mathbf{W}) \):

\[
\mathcal{R}(\mathbf{W}) \approx \mathcal{R}^q(\mathbf{W}) = \frac{1}{2} \sum_{i=0}^{m-1} z_i (1 - z_i) \text{Var}_e(\overline{A}_i \overline{X} \cdot \mathbf{W}).
\]

(12)

\( \blacksquare \)
C Additional Experiments

C.1 Results on Large Datasets

Table 4: Statistics of Large Datasets.

<table>
<thead>
<tr>
<th></th>
<th>Classes</th>
<th>Features</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora-Full</td>
<td>67</td>
<td>8,710</td>
<td>18,703</td>
<td>62,421</td>
</tr>
<tr>
<td>Coauthor CS</td>
<td>15</td>
<td>6,805</td>
<td>18,333</td>
<td>81,894</td>
</tr>
<tr>
<td>Coauthor Physics</td>
<td>5</td>
<td>8,415</td>
<td>34,493</td>
<td>247,962</td>
</tr>
<tr>
<td>Aminer CS</td>
<td>18</td>
<td>100</td>
<td>593,486</td>
<td>6,217,004</td>
</tr>
<tr>
<td>Amazon Computers</td>
<td>10</td>
<td>767</td>
<td>13,381</td>
<td>245,778</td>
</tr>
<tr>
<td>Amazon Photo</td>
<td>8</td>
<td>745</td>
<td>7,487</td>
<td>119,043</td>
</tr>
</tbody>
</table>

We also evaluate our methods on six relatively large datasets, i.e., Cora-Full, Coauthor CS, Coauthor Physics, Amazon Computers, Amazon Photo and Aminer CS. The statistics of these datasets are given in Table 4. Cora-Full is proposed in [1]. Coauthor CS, Coauthor Physics, Amazon Computers and Amazon Photo are proposed in [7]. We download the processed versions of the five datasets here. Aminer CS is extracted from the DBLP data downloaded from https://www.aminer.cn/citation. In Aminer CS, each node corresponds to a paper in computer science, and edges represent citation relations between papers. These papers are manually categorized into 18 topics based on their publication venues. We use averaged GLOVE-100 word vector of paper abstract as the node feature vector. Our goal is to predict the corresponding topic of each paper based on feature matrix and citation graph structure.

Following the evaluation protocol used in [7], we run each model on 100 random train/validation/test splits and 20 random initializations for each split (with 2000 runs on each dataset in total). For each trial, we choose 20 samples for training, 30 samples for validation and the remaining samples for test. We ignore 3 classes with less than 50 nodes in Cora-Full dataset as done in [7]. The results are presented in Table 5. The results of GCN and GAT on the first five datasets are taken from [7]. We can observe that GRAND significantly outperforms GCN and GAT on all these datasets.

Table 5: Results on large datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>Cora Full</th>
<th>Coauthor CS</th>
<th>Coauthor Physics</th>
<th>Amazon Computers</th>
<th>Amazon Photo</th>
<th>Aminer CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>62.2 ± 0.6</td>
<td>91.1 ± 0.5</td>
<td>92.8 ± 1.0</td>
<td>82.6 ± 2.4</td>
<td>91.2 ± 1.2</td>
<td>49.9 ± 2.0</td>
</tr>
<tr>
<td>GAT</td>
<td>51.9 ± 1.5</td>
<td>90.5 ± 0.6</td>
<td>92.5 ± 0.9</td>
<td>78.0 ± 19.0</td>
<td>85.7 ± 20.3</td>
<td>49.6 ± 1.7</td>
</tr>
<tr>
<td>GRAND</td>
<td>63.5 ± 0.6</td>
<td>92.9 ± 0.5</td>
<td>94.6 ± 0.5</td>
<td>85.7 ± 1.8</td>
<td>92.5 ± 1.7</td>
<td>52.8 ± 1.2</td>
</tr>
</tbody>
</table>

C.2 Efficiency Analysis

The efficiency of GRAND is mainly influenced by two hyperparameters: the propagation step $K$ and augmentation times $S$. Figure reports the average per-epoch training time and classification accuracy of GRAND on Cora under different values of $K$ and $S$ with #training epochs fixed to 1000. It also includes the results of the two-layer GCN and two-layer GAT with the same learning rate, #training epochs and hidden layer size as GRAND.

From Figure we can see that when $K = 2, S = 1$, GRAND outperforms GCN and GAT in terms of both efficiency and effectiveness. In addition, we observe that increasing $K$ or $S$ can significantly improve the model’s classification accuracy at the cost of its training efficiency. In practice, we can adjust the values of $K$ and $S$ to balance the trade-off between performance and efficiency.

C.3 Parameter Sensitivity

We investigate the sensitivity of consistency regularization (CR) loss coefficient $\lambda$ and DropNode probability $\delta$ in GRAND and its variants on Cora. The results are shown in Figure 6. We observe that

https://github.com/shchur/gnn-benchmark
their performance increase when enlarging the value of $\lambda$. As for DropNode probability, GRAND, GRAND\_GCN and GRAND\_GAT reach their peak performance at $\delta = 0.5$. This is because the augmentations produced by random propagation in that case are more stochastic and thus make GRAND generalize better with the help of consistency regularization.

### C.4 DropNode vs Dropout

![Figure 7: GRAND vs. GRAND\_dropout.](image)

We compare GRAND and GRAND\_dropout under different values of propagation step $K$. The results on Cora, Citeseer and Pubmed are illustrated in Figure 7. We observe GRAND always achieve better
performance than GRAND_dropout, suggesting DropNode is much more suitable for graph data augmentation.

Figure 8: Over-smoothing: GRAND vs. GRAND_GCN & GRAND_GAT on Cora.

C.5 GRAND vs. GRAND_GCN & GRAND_GAT

As shown in Table 1, GRAND_GCN and GRAND_GAT get worse performances than GRAND, indicating GCN and GAT perform worse than MLP under the framework of GRAND. Here we conduct a series of experiments to analyze the underlying reasons. Specifically, we compare the MADGap values and accuracies GRAND, GRAND_GCN and GRAND_GAT under different values of propagation step $K$ with other parameters fixed. The results are shown in Figure 8. We find that the MADGap and classification accuracy of GRAND increase significantly when enlarging the value of $K$. However, both the metrics of GRAND_GCN and GRAND_GAT have little improvements or even decrease. This indicates that GCN and GAT have higher over-smoothing risk than MLP.

C.6 Performance of GRAND under different label rates

We have conducted experiments to evaluate GRAND under different label rates. For each label rate setting, we randomly create 10 data splits, and run 10 trials with random initialization for each split. We compare GRAND with GCN and GAT. The results are shown in Table 6. We observe that GRAND consistently outperforms GCN and GAT across all label rates on three benchmarks.

Table 6: Classification Accuracy under different label rates (%).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label Rate</td>
<td>1%</td>
<td>3%</td>
<td>5%</td>
</tr>
<tr>
<td>GCN</td>
<td>62.8±5.3</td>
<td>76.1±1.9</td>
<td>79.6±2.1</td>
</tr>
<tr>
<td>GAT</td>
<td>64.3±5.8</td>
<td>77.2±2.4</td>
<td>80.8±2.1</td>
</tr>
<tr>
<td>GRAND</td>
<td>69.1±4.0</td>
<td>79.5±2.2</td>
<td>83.0±1.6</td>
</tr>
</tbody>
</table>

References


