ABSTRACT
Graph neural networks (GNNs) have been widely used under semi-supervised settings. Prior studies have mainly focused on finding appropriate graph filters (e.g., aggregation schemes) to generalize well for both homophilic and heterophilic graphs. Even though these approaches are essential and effective, they still suffer from the sparsity in initial node features inherent in the bag-of-words representation. Common in semi-supervised learning where the training samples often fail to cover the entire dimensions of graph filters (hyperplanes), this can precipitate over-fitting of specific dimensions in the first projection matrix. To deal with this problem, we suggest a simple and novel strategy; create additional space by flipping the initial features and hyperplane simultaneously. Training in both the original and in the flip space can provide precise updates of learnable parameters. To the best of our knowledge, this is the first attempt that effectively moderates the overfitting problem in GNN. Extensive experiments on real-world datasets demonstrate that the proposed technique improves the node classification accuracy up to 40.2%.

CCS CONCEPTS
• Computing methodologies → Machine learning.

KEYWORDS
Graph neural networks, Semi-supervised learning, Flipping initial features

ACM Reference Format:
Yoonhyuk Choi¹, Jiho Choi¹, Taewook Ko¹, Chong-Kwon Kim². 2022. Flip Initial Features: Generalization of Neural Networks for Semi-supervised Node Classification. In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/

1 INTRODUCTION
With the flux of graphical data, graph neural networks (GNNs) accomplished prominent improvement in various fields including molecular [14] and social networks [11]. Integrating node features and network structures concurrently, they have shown the powerful ability for node and graph classification tasks [9, 18, 25, 41]. Message passing, which aggregates features from neighboring nodes recursively, is a key mechanism of GNNs [15].

Contrary to previous approaches, we concentrate on the training mechanism of weight matrices (hyperplanes). Our quest originates from an observation that if initial features contain a few non-zero elements (e.g., bag-of-words), insufficiency in training samples (semi-supervised settings) causes overfitting of specific dimensions in the first layer parameters. Figure 1 is a simple example that reveals that a zero feature vector fails to update the slope of a line. While a green node \((0, 0)\) only updates the bias term (please refer to Appendix A for details), an orange node \((0, 0.5)\) in Figure 1b changes slope and bias. Both initial features well classify nodes in this two dimension example, but if the dimensionality is large, the first hyperplane might be biased in certain dimensions. We emphasize that this can
We also considered to add a noise to inputs. But, this re-
contributions of this paper can be summarized as follows:
over their originals are 17.2 %, 24.2 %, and 17.8 %, respectively. The
of the three flipping methods, Flip-MLP, Flip-GCN, and Flip-GAT
lines on nine real-world datasets. We observe that the average gain
the performances of the three variants with state-of-the-art base-
is orthogonal to GNNs and we apply the flipping mechanism to
semantic learning of each dimension. The flipping mechanism
ent problem caused by input data and also facilitates accurate
and flipped spaces concurrently. Our method solves the zero gradi-
change of parameter can ensure our model operates in the original
features. We also focus on a dual path network which a slight
networks which can preserve the volume of gradients and ini-
ious techniques of shifting parameters and rotation of neural
simulated by previous studies. To better optimize the feature transformation matrix, we focus
on the perturbation of the initial features. Our initial choice was a
data augmentation mechanism like dimensional image shifting used
in computer vision. However, we concluded that shifting is not applicable to GNNs with bag-of-words features where a bit position itself has a specific meaning; positional shifting disarranges the semantic information. Also, compared to the convolutional neural network which ensures local invariance, GNNs employ multi-layer perceptron (MLP) which is not translation invariant. We also considered to add a noise to inputs. But, this requires an additional decoding process with a precise selection of hyperparameters and also implicates a normalization problem.

We devise a solution that flips the initial features and parameters simultaneously. The proposed flipping scheme is inspired by previous techniques of shifting parameters and rotation of neural networks which can preserve the volume of gradients and initial features. We also focus on a dual path network which a slight change of parameter can ensure our model operates in the original and flipped spaces concurrently. Our method solves the zero gradient problem caused by input data and also facilitates accurate semantic learning of each dimension. The flipping mechanism is orthogonal to GNNs and we apply the flipping mechanism to three representative methods (MLP, GCN, and GAT). We compare the performances of the three variants with state-of-the-art baselines on nine real-world datasets. We observe that the average gain of the three flipping methods, Flip-MLP, Flip-GCN, and Flip-GAT over their originals are 17.2 %, 24.2 %, and 17.8 %, respectively. The contributions of this paper can be summarized as follows:

• We show that GNNs are quite sensitive to initial features and their performance can be largely improved if zero elements are eliminated.
• To solve the problem, we propose a flipping mechanism that concurrently transposes initial features and hyperplane. Unlike previous methods that focus on aggregation schemes, our method scrutinizes the backpropagation and subsidizes precise component-wise guiding of the first hyperplane.
• The proposed flipping mechanism is orthogonal to GNNs. Applying flipping to MLP, GCN, and GAT, we develop three flipping variants. We conduct extensive experiments using real-world benchmark graphs. The flipping variants surpass all state-of-the-art baselines.

2 PRELIMINARIES
This section starts with the feature distribution of each benchmark dataset. Then, we define the widely used notations of the graph-structured dataset that will be used throughout this paper. Finally, we describe the basic mechanisms of several GNN algorithms.

2.1 Empirical Analysis
We conduct an empirical study to show the ratio of non-zero elements of each dataset. Firstly, let us define $z$ as below:

$$z = \frac{\text{# of non-zero feature dimension in training nodes}}{\text{# of entire feature dimension}}$$

The numerator contains the elements of all training nodes whose values are non-zero. In Figure 2, we describe the $z$-value of each dataset by varying the range of neighbors from ego to 2-hop adjacent nodes. We can see that the $z$-value increases in proportion to the range since more features are available during the training phase. We also observe that the $z$-values of datasets are quite different. As we will explain later, the experimental results show that the lower the $z$-value of a dataset is, the greater the performance improvement that flipping achieves. We insist that the $z$-value of ego is dominant for gradient updates (see Section 4.6 for details).

2.2 Notations
Let $G=(V, E)$ be an undirected graph that contains $|V|=n$ nodes and $|E|=m$ edges. Let $A \in \{0,1\}^{n \times n}$ be an adjacency matrix determined by the connectivity of a graph. In many cases, it is available to utilize a feature matrix $X \in \mathbb{R}^{n \times F}$ that comprise the properties of the entire nodes, where $F$ is the input dimension of initial features. Each node has its label that is represented as $Y \in \mathbb{R}^{n \times C}$. Here, $C$ is the number of labels (classes). We separate the first layer weight matrix of GNN into two parts $W_o$ and $W_f$, where the lower script $o$ and $f$ denote an original and flipped space, respectively. A symbol $\nabla$ represents a partial derivative. Our goal is to solve a node classification task under semi-supervised settings, where the labeled nodes $V_L \subset V$ are partially available as a training set. GNNs focus on how to better utilize the given information for the prediction of unlabeled nodes $V_U = V - V_L$.

Figure 2: Initial feature distribution of each graph dataset. The definition of $z$-value is described in Equation 1
We focus on spatial GNNs which can reduce the computational cost (e.g., ReLU). Applying softmax on the output \( \sigma \) node pairs, and integrate them with the original adjacency matrix. 

Layer-wise attention is used, but we omit it for brevity. Others will obtain Flip-MLP and Flip-GAT as demonstrated in Figure 3, the upper and lower panels describe the original and flipped spaces, respectively. Even though the proposed flipping scheme is orthogonal to GNN techniques, we take GCN as plane GCN [25]. The last row and column of \( A^{\text{plane}} \) represents a full-batch gradient of Equation 3. As can be seen in the above equation, the sparseness (or low value) of \( \hat{A} \) can obstruct the gradient flow between dissimilar nodes. Unfortunately, there arises a problem when updating the parameters of initial layer \( W^{(1)} \) as:

\[
\nabla_W W^{(1)} = \hat{A} H^{(1)\top} \nabla_{H^{(1)}} J, \quad l = 1, \ldots, L.
\]

The above equation implies that the gradients become zero for certain dimensions with zero inputs (please refer to Figure 1a). Thus, aggregation itself cannot handle the precise update of the first hyperplane \( W^{(1)} \), necessitating a sparseness removal mechanism for the initial features \( X \). To enable \( W^{(1)} \) to learn the precise meaning of each dimension, we suggest flipping which is simple but is effective for semi-supervised learning. In the upcoming subsection, we detail the training mechanism of GCN in the flipped space accompanied by parameter adjustment.

### 3.1 Flipped Graph Neural Networks

The inadequate gradient update problem described before can be more conspicuous in semi-supervised learning. A simple remedy to the problem is shifting which is popularly used in computer vision. To remove zero signals, shifting adds a vector \( X_{\text{shift}} \), which consists of elements of small values, to the input features \( X \) as below:

\[
X = X + X_{\text{shift}}.
\]

However, shifting may not be applicable to GNNs because MLP is not shift-invariant and can deteriorate robustness [39]. Further, it changes the magnitude of an input which is critical to the normalization of neural networks. For a more rigorous investigation on shifting, we devise a simple method that combines shifting with GCN and describes its performance in Appendix B.

We develop a scheme that flips both feature vectors and the hyperplane concurrently. If an original feature all of whose elements belong to \([0, 1]\), its symmetric transposition through \( p = (0.5, \ldots, 0.5) \) also resides in this range. The inadequate gradient update problem rooted in the sparseness of the initial features. However, as explained below, an aggregation scheme alone cannot avoid the improper learning problem rooted in the sparseness of the initial features. To explain the problem in greater detail, we first scrutinize the backpropagation of GNNs:

\[
\nabla_W W^{(1)} = \hat{A} H^{(1)\top} \nabla_{H^{(1)}} J, \quad l = 1, \ldots, L.
\]
We should also flip the first hyperplane $F$.

The loss $L_{GNN}$ can be computed using the final representation $H_f^{(L)}$ as below:

$$L_{GNN} = L_{nll}(Y, \tilde{Y}_f), \quad \tilde{Y}_f = \text{softmax}(H_f^{(L)}).$$

**Algorithm 1** The overall mechanism of Flip-GCN

Require: Adjacency matrix $A$, node features $X$, parameters $\theta_G$, epoch $K$, best valid and test score $\delta^*$, learning rate $\eta$

Ensure: Node classification accuracy $\delta^*$

1. for number of training epoch $K$ do
2. # GCN in original space
3. for training samples do
4. Using initial features $X$, compute the $L_{GNN}$ (Eq. 3)
5. Update $\theta_{k+1} = \theta_{k} - \eta \frac{\partial L_{GNN}}{\partial \theta_{k}}$
6. end for
7. Compute the validation $\gamma$ and test score $\delta$
8. if $\gamma > \gamma'$ then
9. $\delta^* = \delta$
10. # GCN in flipped space
11. for training samples do
12. Flip initial features using Eq. 10
13. Flip the first hyperplane through Eq. 11
14. Using these parameters, compute $L_{GNN}$ in Eq. 13
15. Update $\theta_{k+2} = \theta_{k+1} - \eta \frac{\partial L_{GNN}}{\partial \theta_{k+1}}$
16. end for
17. Compute the validation $\gamma$ and test score $\delta$
18. if $\gamma > \gamma'$ then
19. $\delta^* = \delta$
20. end for

Based on Eq. 13, $W^{(1)}$ can be trained in the flipped space. We describe the mechanism of our method in Algorithm 1.

### 3.2 Optimization Strategy

We define two loss functions in Equation 3 and 13. For gradient analyses, it is worth noting the following equation:

$$H_f^{(2)} = \sigma(\tilde{A}XW_o^{(1)}) = H_f^{(2)} = -\sigma(\tilde{A}XfW_f^{(1)})$$

which implies that the outputs (or gradients) of two different spaces are equivalent after two layers as below:

$$\nabla W_0^{(i)}J_f = \nabla W_f^{(i)}J_f, \quad (i \geq 2).$$

Similar to $J_f = L_{GNN}$ is a full-batch gradient in the flipped space (Eq. 13). Even though Sigmoid or Tanh guarantees a perfect symmetry $J_f = J_f$, we employ ReLU for better performance. Now, referring to Eq. 7, we define the gradients of the first hyperplane $W^{(1)}$ on both spaces as below.

In the original space, update $W_o^{(1)}$:

$$\nabla W_0^{(i)}J_f = (\tilde{A}X)^T \nabla H_0^{(i)}J_f, \quad H_0^{(2)} = \tilde{A}XW_o^{(1)}.$$ (16)

In the flipped space, update $W_f^{(1)}$:

$$\nabla W_f^{(i)}J_f = (\tilde{A}X_f)^T \nabla H_o^{(i)}J_f, \quad H_f^{(2)} = \tilde{A}X_fW_f^{(1)}.$$ (17)

Through the above equations, we can guide the first hyperplane $W^{(1)}$ concurrently.
To analyze the computational complexity, we divide our model into two parts. As demonstrated in Figure 3, the proposed scheme consists of a vanilla GCN [25] and a flipped GCN. The computational cost of vanilla GCN is known to be \( O(|E|P_{GCN}) \), where \( P_{GCN} \) and \( d_{max} \) is the number of learnable weights. More specifically, \( P_{GCN} \) can be decomposed [51] into \( O(nz(X)F' + F'C) \), where \( nz(X) \) is the number of non-zero elements in input \( X \) and \( F' \) is the dimension after projection. \( F'C \) is the parameter of the second convolution layer. Also, \( O(|E|) \approx O(|E|d_{max}) \) and \( d_{max} \) is the maximum degree. Similar to the vanilla GCN, the complexity of the flipped GCN can be defined as \( O(|E|P'_{GCN}) \). In detail, \( P'_{GCN} = O(nz(X_f)F' + F'C) \) where \( nz(X_f) \) is the number of non-zero elements in flipped features. Summarizing the above two equations, the complexity of our model is \( O(|E|(P_{GCN} + P'_{GCN})) \), which is proportional to the number of edges \( |E| \) and the size of learnable matrices \( P_{GCN} + P'_{GCN} \).

4 EXPERIMENTS

This section describes the experiments for the performance analysis. We focused our efforts to find answers to the following research questions:

- **RQ1**: Does flipping efficiently solve the problem occurred by multiple zero values in initial features?
- **RQ2**: Does flipping preserve the convergence?
- **RQ3**: How does the performance of the flipping method change as the number of training samples vary?
- **RQ4**: How much do the gradients from the original and flipped spaces differ?

### 4.1 Datasets and Baselines

Details of datasets. Our experiments are conducted on nine datasets whose statistical details are described in Table 1. We also measure the assortativity of each dataset as below:

\[
h = \frac{\# \text{ of edges that connect nodes with the same label}}{\# \text{ of entire edges}}
\]  

| Datasets      | Cora | Citeseer | Pubmed | Actor | Chameleon | Squirrel | Cornell | Texas | Wisconsin |
|---------------|------|----------|--------|-------|-----------|----------|---------|-------|-----------|
| # Nodes       | 2,708| 3,327    | 19,717 | 7,600 | 2,277     | 5,201    | 183     | 183   | 251       |
| # Edges       | 10,558| 9,104    | 88,648 | 25,944| 33,824    | 38,421   | 211,872 | 295   | 309       |
| # Features    | 1,433| 3,703    | 500    | 931   | 2,325     | 2,089    | 1,703   | 1,703 | 1,703     |
| # Classes     | 7     | 6        | 3      | 5     | 5         | 5        | 5       | 5     | 5         |
| # Training Nodes| 140   | 120      | 60     | 100   | 100       | 100      | 25      | 25    | 25        |
| # Validation Nodes| 1,568 | 2,207    | 18,657 | 3,750 | 1,088     | 2,550    | 79      | 79    | 113       |
| # Test Nodes  | 1,000 | 1,000    | 1,000  | 3,750 | 1,089     | 2,551    | 79      | 79    | 113       |

### 3.3 Design Choice

Though plane GNN guarantees the convergence to a local optimum, the initial features might incur overfitting in the first hyperplane. Consequently, GNNs may fail to learn the meaning of entire dimensions when bag-of-words are adopted as features. As described in Eq. 8, shifting might be one solution but it raises a difficult regularization issue. Therefore, we devise flipping which preserves the scale of initial features before and after transposition. We further pick \( p_1 \) as an anchor point such that the first hyperplane can be trained concurrently from both spaces.

### 3.4 Time Complexity

To analyze the computational complexity, we divide our model into two parts. As demonstrated in Figure 3, the proposed scheme consists of a vanilla GCN [25] and a flipped GCN. The computational cost of vanilla GCN is known to be \( O(|E|P_{GCN}) \), where \( P_{GCN} \) and \( d_{max} \) is the number of learnable weights. More specifically, \( P_{GCN} \) can be decomposed [51] into \( O(nz(X)F' + F'C) \), where \( nz(X) \) is the number of non-zero elements in input \( X \) and \( F' \) is the dimension after projection. \( F'C \) is the parameter of the second convolution layer. Also, \( O(|E|) \approx O(|E|d_{max}) \) and \( d_{max} \) is the maximum degree. Similar to the vanilla GCN, the complexity of the flipped GCN can be defined as \( O(|E|P'_{GCN}) \). In detail, \( P'_{GCN} = O(nz(X_f)F' + F'C) \) where \( nz(X_f) \) is the number of non-zero elements in flipped features. Summarizing the above two equations, the complexity of our model is \( O(|E|(P_{GCN} + P'_{GCN})) \), which is proportional to the number of edges \( |E| \) and the size of learnable matrices \( P_{GCN} + P'_{GCN} \).
Table 2: Node classification accuracy (%) on benchmark datasets. Bold with an asterisk (*) symbol indicates the best performance, and methods with † are built upon GCN. We show α, β that achieves the best accuracy which is defined in Eq. 19

| Datasets | Cora | Citeeseer | Pubmed | Actor | Chameleon | Squirrel | Cornell | Texas | Wisconsin |
|----------|------|-----------|--------|-------|-----------|----------|---------|-------|-----------|
| z-value (Eq. 1) | 0.59 | 0.41 | 0.96 | 0.21 | 1.0 | 1.0 | 0.62 | 0.41 | 0.53 |
| Homophily (Eq. 20) | 0.81 | 0.74 | 0.8 | 0.22 | 0.23 | 0.22 | 0.11 | 0.06 | 0.16 |

| Method | α ± | β ± | MLP | GCN† | DropEdge† | Ortho-GCN† | GIN | GAT | GATv2 | APPNP | GCNII | H2-GCN | P-reg† | FAGCN |
|--------|------|------|-----|------|-----------|------------|-----|-----|-------|-------|-------|--------|--------|--------|
| Flip-MLP | +0.8% | 0.3% | 53.2 ±0.5% | 53.7 ±1.7% | 69.7 ±0.4% | 27.9 ±1.1% | 41.2 ±1.8% | 26.5 ±0.6% | 60.1 ±1.2% | 65.8 ±5.8% | 73.5 ±5.4% |
| vs MLP (± %) | +1.4% | +13.1% | +0.3% | +28.7% | +2.3% | +2.1% | +17.3% | +20.4% | +9.5% |
| α, β | 1.01 | 1.1 | 1.01 | 1.01 | 1.1 | 1.1 | 1.1 | 1.1 | 1.1 |
| Flip-GCN† | +0.6% | 0.4% | 82.7 ±0.5% | 72.4 ±0.9% | 79.2 ±0.2% | 28.6 ±0.3% | 50.4 ±0.5% | 32.4 ±0.3% | 52.4 ±0.6% | 62.2 ±1.8% | 52.3 ±2.4% |
| vs GCN (± %) | +4.6% | +7.3% | +1.8% | +40.2% | +2.0% | +19.3% | +33.0% | +30.7% | +29.1% |
| α, β | 0.1, 0.01 | 0.001, 0.001 | 1.001 | 0.1, 0.1 | 1.0001 | 1.1 | 0.01, 0.01 | 0.1, 0.01 | 0.0, 0.01 |
| Flip-GAT | +0.5% | 0.4% | 83.1 ±0.6% | 72.8 ±0.4% | 78.5 ±0.2% | 30.3 ±0.8% | 48.3 ±0.3% | 31.9 ±0.5% | 50.6 ±1.4% | 61.0 ±1.1% | 54.5 ±2.1% |
| vs GAT (± %) | +3.7% | +7.1% | +0.6% | +34.7% | +3.0% | +3.6% | +20.2% | +22.0% | +19.0% |
| α, β | 0.1, 0.01 | 0.001, 0.001 | 1.0001 | 0.1, 0.1 | 1.001 | 1.1 | 0.01, 0.01 | 0.1, 0.01 | 0.0, 0.01 |

Baselines We compare our proposed method with several state-of-the-art baselines below.

- **MLP** [35] adopts a feed-forward neural network without using neighboring nodes.
- **GCN** [25] is a first-order approximation of Chebyshev polynomials [9] that utilizes localized filters.
- **DropEdge** [36] randomly deletes edges based with a given probability in order to alleviate over-fitting.
- **Ortho-GCN** [17] maintains the orthogonality of feature transformation matrices through three constraints.
- **GIN** [45] adopts an injective mapping function to enhance the discriminative power of GCN and ensures isomorphism.
- **GAT** [41] employs node features to assign different weights for each edge without graph structural property.
- **GATv2** [2] improves GAT to generate a more dynamic graph attention that is more expressive.
- **APPNP** [26] combines personalized PageRank with GCN to improve accuracy while reducing a computational cost.
- **GCNII** [6] further utilizes weighted identity mapping function to redeem the deficiency of APPNP.
- **H2-GCN** [51] separates ego from neighbors and apply hop-based aggregation for heterophilic networks.
- **P-reg** [47] improves the traditional graph Laplacian to provide extra information that GNNs might not capture.
- **FAGCN** [1] adaptively controls the propagation of low and high values during signals, data, course, and project.

Hyperlinks between web pages. Web pages are classified into five types (labels) based on the monthly traffic.

- **Cornell, Texas, Wisconsin** The WebKB datasets include web pages that are collected from computer science departments of multiple universities. Here, the nodes are web pages and the edges are hyperlinks between them. Similar to citation networks and actor graphs, the node features are binary values (bag-of-words) and their labels are manually classified as the student, faculty, staff, course, and project.

4.2 Experiment Details

We implemented the baselines and our method using *PyTorch Geometric* [13]. In detail, two layers of GNNs are used for all baselines except for APPNP and GCNII. The hidden dimension is set to 64 and we adopted ReLU as an activation function with dropout. The log Softmax is applied for classification. A learning ratio of (1e<sup>-3</sup>) and Adam optimizer with weight decay (5e<sup>-4</sup>) are used for training. As illustrated in Table 1, we randomly chose 20 nodes per class as training samples except for the WebKB datasets whose size is smaller than the others. The remaining nodes are randomly and equally divided into validation and testing nodes. The test accuracy is measured at each iteration, and we selected one that achieves the best validation score. Our source code is available here<sup>3</sup>. In addition to flip-GCN, we describe the details of Flip-MLP and Flip-GAT in Appendix C.

---

<sup>1</sup>http://www.cs.cmu.edu/~afs/cs/project/theo-20/www/data/

<sup>2</sup>https://pytorch-geometric.readthedocs.io/en/latest/modules/mn.html

<sup>3</sup>https://anonymous.4open.science/r/Flip-GCN-9870
4.3 Results and Discussion (RQ1)

Most importantly, in Table 2, we notice that the three flipping variants (Flip-MLP, Flip-GCN, and Flip-GAT) perform significantly better than their bases (MLP, GCN, and GAT). Now, we analyze these results from two perspectives.

Performance of flipping is sensitive to the $z$-value. Because flipping is designed to reduce overfitting caused by the sparsity in initial features, we can easily presume that $z$-value, the non-zero element ratio, is the key factor that determines the performance gains of flipping. Indeed, flipping attains larger performance gains on low $z$-value datasets than on high $z$-value datasets. For three datasets with high $z$-values (Pubmed, Chameleon, and Squirrel), the advancement of flipping over their vanilla models (e.g., Flip-MLP vs MLP) is relatively small. However, all three flipping methods perform better than the original methods achieving average gains of 3 %, 1.9 %, and 2.4 %, respectively, indicating the effectiveness of flipping even for non-zero initials. On another dataset with low $z$-values, the three flipping variants obtain remarkable advancements over their bases achieving 17.2 %, 24.2 %, and 17.8 % on average. Notably, flipping methods perform best except for Squirrel (with high $z$ and low homophily). This may imply that a slight perturbation to the input features can have a greater impact than aggregation scheme modifications under semi-supervised settings.

The accuracy of GNNs are highly related to the homophily ratio. Message passing based GNNs exploit the homophily (Eq. 20) property commonly observed in graphs [28, 46]. Three citation graphs have higher homophily ratios ($\geq 0.7$) than others. Here, GNNs overwhelm MLP on these homophilic graphs. However, in other datasets like Actor and three WebKB networks, Flip-MLP achieves the best accuracy among the baselines. This implies that message passing fails to generalize well under the high heterophily. Further, the performance gain of Flip-MLP is higher than Flip-GNNs on homophilic graphs, but on heterophilic graphs, GNNs benefit from the advantages of flipping. Among baselines, APPNP performs best on homophilic graphs and some methods (H2GCN, FAGCN) outperform GCN on heterophilic networks. However, our flipping methods surpass these algorithms regardless of homophilic ratios.

4.4 Convergence Analysis (RQ2)

One may argue that flipping may hurt the stability of training because it operates on two spaces. Figure 5 plots the performance of two baselines (GCN and GAT) and two flipping methods (Flip-GCN and Flip-GAT) as a function of the number of iterations (one dataset Texas is excluded because of a space limitation). The performance of plain GCN (orange), plain GAT (green), flip-GCN (blue), and flip-GAT (red) are plotted with different colors. Here, the x-axis is the training epoch and the y-axis is node classification accuracy.

In Figure 5, we can see that the flipped methods outperform the plain algorithms. Also, compared to the plane GCN and GAT, we notice that flip-based methods experience higher oscillations. We contemplate that this is caused by the gradient updates in the flipped space, where most dimensions are updated at each iteration. Some may argue that the instability of convergence might be risky for neural networks. However, acquiring stable but lower performance with high over-fitting is not the ultimate goal of the problems and we can handle them through the measurement of validation score, which is a widely used strategy for performance evaluation. To summarize, flipping methods achieve better performance than the plain models on datasets with multiple zero elements but we also admit that further investigation has to be conducted for better convergence.

4.5 Varying the Size of Training Samples (RQ3)

As we pointed out before, the root cause that deteriorates the performance of semi-supervised learning is the lack of labeled samples. Thus, we scrutinize the effect of labeled sample size on performance by adjusting the scale of training samples. In Table 3, we show the $z$-value of central nodes based on the number of labeled nodes per class (L/C). Due to the space limitation, we exclude Wisconsin here.

From the above result, GCN and GAT outperform MLP since the above two datasets are quite assortative (please see Homophily in Table 2). At the same time, we can observe that the performance gain of flipping decreases as the L/C increases. The reason is that as
Table 3: Node classification accuracy (%) w.r.t. the different number of training samples. The symbol (+F) means that flipping is applied on a base method.

| Dataset | Cora | Actor | Citeseer | Chameleon | Squirrel | Cornell | Texas |
|---------|------|-------|----------|-----------|----------|---------|-------|
| L/C     | 0.59 | 0.76  | 0.91     | 0.41      | 0.59     | 0.78    | 0.91  |
| z-value | 20   | 40    | 80       | 20        | 40       | 80      | 5     |
| MLP     | 53.2 | 56.9  | 62.1     | 53.7      | 57.3     | 63.8    | 41.2  |
| MLP+F   | 61.4 | 65.5  | 70.4     | 60.3      | 63.1     | 67.7    | 43.5  |
| GCN     | 79.1 | 82.8  | 83.4     | 76.7      | 69.3     | 71.5    | 20.4  |
| GCN+F   | 82.7 | 84.5  | 85.5     | 72.4      | 73.3     | 75.6    | 28.6  |
| GAT     | 80.1 | 82.4  | 83.0     | 68.0      | 68.9     | 71.3    | 22.5  |
| GAT+F   | 83.1 | 84.0  | 84.6     | 72.8      | 73.4     | 74.3    | 30.3  |

Figure 6: Using the Cora dataset, we plot the magnitude of the first projection matrix gradients during the training phase as a function of the number of iterations.

The first one established a mathematical foundation of conducting graph convolution operation in a spectral domain using the Laplacian matrix [3, 9, 10]. Spatial-based GNNs aggregate the information of local neighborhoods from spatial perspectives, and ignited many aggregation scheme developments for the handling of noisy connections [1, 8, 34, 41, 51]. Compared to the noisy neighbor problem, the problem incurred by the sparse initial features has not received much research attention.

Generalization of neural networks. Many approaches have been proposed for the generalization of parameters in neural networks [5, 12, 43]. Several suggested the normalization of deep neural networks [19] while others applied regularization to all adjacent nodes [47] or integrated label propagation to give further information [42]. More recently, the orthogonal GCN [17] attacks the gradient vanishing problem at the initial few layers of GNNs. Rawls-GCN [22] claims the unfairness of gradient update which is biased to nodes with a large degree. Though these methods show notable improvements under the semi-supervised scenario, they fail to solve the problem that is inherently occurred by a characteristic of initial features. In this paper, we aim to solve this limitation through the generalization of the first hyperplane.

6 CONCLUSION

Existing GNNs mainly focus on the tuning of aggregation strategy, while the type of initial features is unconsidered. In this work, we investigate the theoretical relationship between the zero elements of input vectors and their impact on the first layer of neural networks. We then propose a co-training in both the original and
flipped spaces, which leverages the gradient flows from both channels and adaptively adjusts the parameters. We provide an analysis of the backpropagation and empirical studies on nine real-world benchmark datasets. Equipping three base methods with flipping improves the node classification accuracy, which indicates that our suggestion is highly scalable and competitive. In future work, we expect the application of flipping with many other variations of GNNs to enhance their quality.

REFERENCES

[1] Deyu Bo, Xiao Wang, Chuan Shi, and Huaweishen. 2021. Beyond low-frequency information in graph convolutional networks. arXiv preprint arXiv:2103.08797 (2021).
[2] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491 (2021).
[3] Jian Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013).
[4] Tianqi Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-Yan Liu, and Liwei Wang. 2021. Graphorm: A principled approach to accelerating graph neural network training. In International Conference on Machine Learning, PMLR, 1204–1215.
[5] Jianfei Chen, Jun Zhu, and Le Song. 2017. Stochastic training of graph convolutional networks with variance reduction. arXiv preprint arXiv:1710.10568 (2017).
[6] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In International Conference on Machine Learning, PMLR, 1725–1735.
[7] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shucheng Yan, and Jiaxi Feng. 2017. Dual path networks. Advances in neural information processing systems 30 (2017).
[8] Li Chien, Jianhao Peng, Pan Li, and Olga Milenkovic. 2020. Adaptive universal generalized pagerank graph neural network. arXiv preprint arXiv:2006.07988 (2020).
[9] Michael Deferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional networks on graphs with fast localized spectral filtering. Advances in neural information processing systems 29 (2016).
[10] Yushun Dong, Kaize Ding, Brian Jalaian, Shuiwang Ji, and Jundong Li. 2021. Graphnorm: A principled approach to accelerating graph neural network training. In Proceedings of the ACM Web Conference 2022, 1214–1225.
[11] Huayun Gu, Xiaofei Liu, and Yizhou Sun. 2021. Scalable graph convolutional networks with only inductive bias. arXiv preprint arXiv:2010.10229 (2020).
[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).
[13] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997 (2018).
[14] Yuanzhi Li and Yang Yuan. 2017. Convergence analysis of two-layer neural networks with relu activation. Advances in neural information processing systems 30 (2017).
[15] Derek Lim, Felix Hohne, Xiuyu Li, Siija Linda Huang, Vaibhavi Gupta, Omkar Bhalerao, and Ser Nam Lim. 2021. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. Advances in Neural Information Processing Systems 34 (2021), 20887–20902.
[16] Mingbao Lin, Ronrong Ji, Bo Zhang, Jiawen Wang, Yingjiao Fu, Yu Hyung Chang, and Chia-Wen Lin. 2020. Rotated binary neural network. Advances in neural information processing systems 33 (2020), 7474–7485.
[17] Hongrui Liu, Binbin Hu, Xiao Wang, Chuan Shi, Zhiqiang Zhang, and Jun Zhou. 2022. Confidence May Cheat: Self-Training on Graph Neural Networks under Distribution Shift. In Proceedings of the ACM Web Conference 2022, 1248–1258.
[18] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jinchao Hu, Haifeng C. and Xiang Zhang. 2021. Learning to drop: Robust graph neural network via topological denoising. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining, 779–787.
[19] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. Annual review of sociology 27, 1 (2001), 415–444.
[20] Shashank Pantid, Dun Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In Proceedings of the 16th international conference on World Wide Web, 201–210.
[21] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geo-gen: Geometric graph convolutional networks. arXiv preprint arXiv:2002.05287 (2020).
[22] Mario-Cristian Constantinescu, Valentina E Balas, Liliana Persuc-Popescu, and Nikos Mastorakis. 2009. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems 6, 7 (2009), 579–588.
[23] Yu Rong, Wenhong Huang, Tingyang Xu, and Junzhou Huang. 2021. Dropedge: Towards deep graph convolutional networks on node classification. arXiv preprint arXiv:2010.10469 (2021).
[24] Benedek Rózemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. Gemsec: Graph embedding with self-clustering. In Proceedings of the 2019 IEEE/ACM international conference on advances in social networks analysis and mining, 65–72.
[25] Jia Shijie, Wang Peng, Jia Peiyi, and Hu Siping. 2017. Research on data augmentation techniques in training dnns: Methodology, analysis and application. Advances in Neural Information Processing Systems 30 (2017).
[26] Jia Shijie, Wang Ping, Jia Peiyi, and Hu Siping. 2017. Research on data augmentation techniques in training dnns: Methodology, analysis and application. Advances in Neural Information Processing Systems 30 (2017).
[27] Yuanzhi Li and Yang Yuan. 2017. Convergence analysis of two-layer neural networks. In Proceedings of the 33rd International Conference on Machine Learning, PMLR, 1725–1735.
[28] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 807–816.
[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 7715–7751.
[30] Ser Nam Lim. 2021. Large scale learning on non-homophilous graphs. In Proceedings of the 16th international conference on World Wide Web, 201–210.
[31] Hongyuan Zeng, Zhongyi Chen, and Changzheng Yang. 2021. Learning how to propagate messages in graph neural networks. In Proceedings of the 29th ACM SIGKDD international conference on Knowledge Discovery & Data Mining, 1894–1903.
[32] Keyulu Xu, Weiwei Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? arXiv preprint arXiv:1810.08260 (2018).
[33] Binbin Hu, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond low-frequency information in graph convolutional networks. arXiv preprint arXiv:2012.04836 (2019).
[34] Wei Jin, Tyler Deng, Yu Wang, Yao Ma, Ziruo Lin, and Jiliang Tang. 2021. Node similarity preserving graph convolutional networks. In Proceedings of the 14th ACM international conference on web search and data mining, 148–156.
[35] Rongrong Ji, Yan Zhou, and Chuan Shi. 2019. How to train graph neural networks? In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 36. 3996–4004.
[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. Advances in neural information processing systems 30 (2017).
[37] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. 2020. Normalization techniques in training dnn: Methodology, analysis and application. arXiv preprint arXiv:2009.12836 (2020).
[38] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning. PMLR, 448–456.
[39] Wei Jin, Tyler Deng, Yu Wang, Yao Ma, Ziruo Lin, and Jiliang Tang. 2021. Node similarity preserving graph convolutional networks. In Proceedings of the 14th ACM international conference on web search and data mining, 148–156.
[40] Jian Kang, Yan Zhu, Yinglong Xia, Jiebo Luo, and Hanghang Tong. 2022. Rawgen. Towards rawlisan difference principle on graph convolutional network.
[49] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems 32 (2019).

[50] Xin Zhang, Li Liu, Yuxiang Xue, Jie Chen, Lingda Wu, and Matti Pietikainen. 2017. Rotation invariant local binary convolution neural networks. In Proceedings of the IEEE International Conference on Computer Vision Workshops. 1210–1219.

[51] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in Neural Information Processing Systems 33 (2020), 7793–7804.

[52] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In Proceedings of the Web Conference 2021. 2069–2080.
SUPPLEMENTARY MATERIAL

A FEATURES AND PARAMETER UPDATE

In Figure 1, we take two examples using 2-dimensional input data X and a single hyperplane (line) W, respectively. Here, we exclude the activation layer since it is a single layer. Let us assume the true label y of this node is -1. Firstly, we should define the objective function that measures the difference between predicted and true labels as below:

\[
\bar{y} = W^T X = \sum_{i=1}^{2} W_i X_i + b, \\
\nabla_{W_j} L = \eta (y - \bar{y}) X_j.
\]

Simply, we assume the learning ratio \( \eta \) as 1. Thus, W can be updated as below:

\[
\begin{pmatrix}
W_1' \\
W_2'
\end{pmatrix} = \begin{pmatrix}
W_1 - X_1 (1 - \bar{y}) \\
W_2 - X_2 (1 - \bar{y})
\end{pmatrix}
\]

, and also bias term as \( b' = b - (-1 - \bar{y}) \).

Back into Figure 7a, let us assume the initial line is \( y = x - 1 \). Since the value of input features \( X = (0, 0) \), the weights \( W \) are not updated. Instead, the bias becomes \( 1 \) \( b' = -1 - (-1 - 1) = 1 \) and successfully predict its label as -1.

Likewise, in Figure 7b, let us consider a line and initial features as \( y = x \) and \( (0, 0.5) \), respectively. Then, the weights are updated as below:

\[
\begin{pmatrix}
W_1' \\
W_2'
\end{pmatrix} = \begin{pmatrix}
-1 - 0 \ast (1 - (0.5)) \\
1 - 0.5 \ast (1 - (0.5))
\end{pmatrix} = \begin{pmatrix}
-1 \\
0.75
\end{pmatrix}
\]

, and the bias as \( 0.5 \) \( b' = 0 - (-1 - (0.5)) = 1.5 \). Consequently, \( W \) becomes \( y = \frac{3}{2} x + 1.5 \) which also classifies this node \( (0,0.5) \) as -1.

The main observation is that the update of a hyperplane is quite sensitive to the element of input vectors, even though both of them are well-optimized in the two examples.

Figure 7: Analysis on gradient update. The dotted line is the line before the update, and the solid one can be derived after backpropagation

Figure 8: Variation of node classification accuracy w.r.t. the shifted value \( X_s \) on four benchmark datasets

Table 4: Node classification accuracy (%) on six benchmark datasets. We describe \( X_s \) with the best performance

| Methods | MLP | MLP+S | \( X_s \) | GCN | GCN+S | \( X_s \) |
|---------|-----|-------|---------|-----|-------|---------|
| Cora    | 53.2| 53.5  | 0.01    | 79.1| 82.8  | 0.2     |
| Citeseer| 53.7| 54.6  | 0.05    | 67.5| 70.5  | 0.1     |
| Actor   | 27.9| 27.9  | 0       | 20.4| 25.9  | 0.9     |
| Cornell | 60.1| 60.1  | 0       | 39.4| 49.4  | 0.5     |
| Texas   | 65.8| 71.4  | 0.2     | 47.6| 53.0  | 0.1     |
| Wisconsin| 73.5| 73.5  | 0       | 40.5| 50.2  | 0.1     |

B ANALYSIS OF SHIFTING FEATURES

In Equation 8, we suggest that shifting can be one solution to solve the gradient vanishing problem. Specifically, adding a small value \( X_s \) to the initial feature \( X \) can remove the zero elements as below:

\[
X = X + X_s.
\]

In Figure 8 and Table 4, we measure node classification accuracy of two baselines (MLP, GCN) using original inputs \( X \) and their another version (MLP+S, GCN+S) with shifted inputs \( X + X_s \). We didn’t renormalize \( X + X_s \) here. The symbol (+S) means shifting is applied to a baseline model. Here, three datasets (Pubmed, Chameleon, Squirrel) are excluded since they contain a majority of non-zero elements. The \( X_s \) of each dataset is selected through a grid search that shows the best validation score.

In Figure 8, we can see that shifting for GCN has a greater impact, where the performance has improved for all datasets. Conversely, shifting for MLP generally downgrades the overall quality except for one dataset (Texas). We think that as the number of non-zero elements increases, a model requires more training samples. Since MLP only utilizes a central node compared to GNNs, shifting itself fails to guide the multiple parameters precisely.
whose value is origin symmetry propagation is defined as follows:

\[ \frac{d}{\theta} \text{attention values} \]

Eq. 12 but we should multiply the negative constant (-1) by the first multi-head projection matrix are defined as follows:

During backpropagation, the gradients of \( \frac{d}{\theta} \) attention layer which is defined in Equation 4, we should consider show the overall architecture of Flip-GAT. Since GAT employs an adjacency matrix, we do not treat them precisely. In Figure 9, we compared to Flip-GCN, they further employ attention layer \( a^{(1)} \) whose value is origin symmetry

\[ \text{Figure 9: The overview of the first layer in Flip-GAT. Compared to Flip-GCN, they further employ attention layer } a^{(1)} \text{ whose value is origin symmetry} \]

C DETAILS OF FLIP-GAT

Since Flip-MLP is a simple modification of Flip-GCN that removes an adjacency matrix, we do not treat them precisely. In Figure 9, we show the overall architecture of Flip-GAT. Since GAT employs an attention layer which is defined in Equation 4, we should consider this for both spaces. Firstly, we discuss the forward propagation in an original space as below:

\[
A^{(l)}_{ij} = \frac{\exp(\sigma(a^{(l)}[h_i^{(l)}|h_j^{(l)}]))}{\sum_{k \in N_i} \exp(\sigma(a^{(l)}[h_i^{(l)}|h_k^{(l)}]))}, \quad (l \geq 1)
\]

\[
H^{(l+1)} = \sigma(H^{(l+1)}), \quad H^{(l+1)} = A^{(l)}H^{(l)}W^{(l)}_o
\]

\[ \tilde{Y} = \text{softmax}(\tilde{H}^{(l)}) \]

During backpropagation, the gradients of \( \theta \) that is considered a multi-head projection matrix are defined as follows:

\[ \nabla_{\theta} W_l J = (A^{(l)}H^{(l)})^T \nabla_{\tilde{H}^{(l+1)}} J, \quad (l \geq 1). \]

Also, referring to Eq. 6, we can retrieve the gradients of the attention layer as below:

\[ \nabla_{A^{(l)}} J = (H^{(l)}W^{(l)}_o)^T \nabla_{\tilde{H}^{(l+1)}} J, \quad (l \geq 1). \]

In a flipped space, the forward propagation is quite similar to Eq. 12 but we should multiply the negative constant (-1) by the first layer attention vector. As described in the middle of Figure 9, the attention values \( d_1, d_2 \) are \( p_2 \) (origin) symmetry. Thus, the forward propagation is defined as follows:

\[
\tilde{A}^{(l)}_{ij} = \frac{\exp(\sigma(-a^{(1)}[h_i^{(1)}|h_j^{(1)}]))}{\sum_{k \in N_i} \exp(\sigma(-a^{(1)}[h_i^{(1)}|h_k^{(1)}]))}
\]

\[
H^{(2)}_f = -\sigma(\tilde{A}^{(1)}X_fW^{(1)}_f), \quad (l = 1)
\]

\[
H^{(l+1)} = \sigma(H^{(l+1)}), \quad \tilde{H}^{(l+1)} = A^{(l)}H^{(l)}W^{(l)}_f, \quad (l \geq 2)
\]

Likewise, the gradients of \( \theta \) are as follows:

\[
\nabla_{\theta} W^{(1)}_f J_f = (A^{(1)}X_f)^T \nabla_{\tilde{H}^{(2)}} J_f, \quad (l = 1)
\]

\[
\nabla_{\theta} W^{(l)}_f J_f = (A^{(l)}H^{(l)}_f)^T \nabla_{\tilde{H}^{(l+1)}} J_f, \quad (l \geq 2).
\]

Similarly, the attention vectors are updated as below:

\[
\nabla_{A^{(l)}} J = -(H^{(l)}W^{(l)}_f)^T \nabla_{\tilde{H}^{(l+1)}} J_f, \quad (l = 1)
\]

\[
\nabla_{A^{(l)}} J_f = (H^{(l)}W^{(l)}_f)^T \nabla_{\tilde{H}^{(l+1)}} J_f, \quad (l \geq 2).
\]