AdaGrad for Training Graph Neural Networks: A Comparative Analysis

Vishvam Patel
University of Alabama in Huntsville
vhp0001@uah.edu

ABSTRACT

Training Graph Neural networks poses unique challenges due to the complex and irregular structure of graphs. Despite the effectiveness of many stochastic optimization algorithms such as SGD, Adam, RMSProp, and AdaGrad for training deep neural networks, only Adam has been explored in training GNNs. This paper investigates the suitability of AdaGrad for optimizing the training of GNNs. This study examines the convergence behavior, optimization efficiency, and generalization ability of AdaGrad in contrast to Adam. The experiments are conducted on three benchmark graph datasets: *Cora, CiteSeer, and Protein-Protein Interaction (PPI)*. The findings of this study will provide valuable insights into the applicability of AdaGrad in training of GNNs.

1. INTRODUCTION

In recent years, Graph Neural Networks (GNNs) have gained popularity for analyzing and understanding data in the form of graphs. Their ability to capture both local and global dependencies within graph structures makes them well-suited for a wide range of tasks, including node classification, link prediction, and graph classification. However, training GNNs presents unique challenges compared to traditional feedforward neural networks because of the irregular and dynamic nature of graph data.

While many stochastic based algorithms have shown effectiveness in training deep neural networks, only Adam[2] has been applied frequently for training GNNs. This study focuses on the suitability of AdaGrad[1], an adaptive gradient algorithm, for optimizing the training of GNNs. AdaGrad adaptively adjusts learning rates of individual parameters based on their historical gradients, making it promising with sparse data and gradients[1]. AdaGrad has shown great potential for training Graph Isomorphic Networks for graph classification tasks[3]. In this paper, the experiments will perform node classification tasks using three benchmark datasets: Cora, CiteSeer, PPI, and two different GNN architectures: GraphSAGE[4] and Graph Convolutional Network (GCN)[5].

The objectives of this study are as follows:

- Provide an understanding of AdaGrad's mechanisms and its implication for GNN optimization.
- Analyze its performance for training GNNs through experimentation

The paper is organized as follows:

- **Section 2** presents the AdaGrad optimization algorithm in-depth while highlighting its key concepts, and internal mechanisms.
- **Section 3** outlines the experimental setup of the comparative analysis, including GNN architectures, datasets, and evaluation metrics used.
- **Section 4** presents the results and discussion of the study
- **Section 5** outlines the lessons learned
- **Section 6** summarizes the findings.

2. ADAGRAD: ADAPTIVE GRADIENT METHOD

AdaGrad was introduced by Duchi et al.[1] in 2011. AdaGrad gained popularity in the world of machine learning because of its adaptive nature based on the geometry of the data. Standard gradient methods use update rules with step sizes without observing the past gradients while AdaGrad adapts the learning rate for each parameter individually. Particularly, it assigns higher learning rates to infrequent features to allow the updates based on relevance. This approach enables AdaGrad to perform well in scenarios where data and gradients are sparse.

A standard gradient method update rule is as follows:

$$x_{t+1} = x_t - \alpha\, g_t$$

where, x is the set of parameters to be updated, t is the time step, g_t is the gradient of the object function, α is the learning rate.

Now, AdaGrad's update rule is as follows:

$$x_{t+1} = x_t - \alpha\, G_t^{-1/2} g_t$$

where, $G_t = g_t^T g_t$, outer product of all previous gradients. As mentioned earlier, for each parameter, AdaGrad adaptively updates the learning rate for each parameter. But, calculating the inverse of the square root of $G_t^{-1/2}$ is computationally impractical for higher dimensions[1]. Therefore, the variant of the above update rule is employed.

$$x_{t+1} = x_t - \alpha\, diag(G_t)^{-1/2} g_t$$

where the algorithm only considers the diagonal elements of the matrix. Both the root and inverse can be computed in linear time without any loss of generality.

In the original paper by Duchi et al.[1], AdaGrad's convergence rate wasn't proven, but it has a regret bound of order $O(\sqrt{T})$ for a sparse sequence with a perfect predictor[1].

AdaGrad was proposed for a convex objective function. Its effectiveness was proven through experiments for tasks such as text classification, image ranking, multiclass optical character recognition, and regression tasks like income predictor. In this paper, we apply AdaGrad for both convex and non-convex objective functions as graph data has varying structure.

3. EXPERIMENTAL SETUP

In this section, we provide a detailed description of the datasets, GNN model architecture, hyper-parameters, and evaluation metrics used. Performs node classification task in a semi-supervised and supervised setting to evaluate AdaGrad's convergence behavior and generalizability.

All the experiments were performed on Intel i7-10750H CPU @2.60Ghz 6 cores 12 processors and 32GB memory. GNN models were defined using PyTorch Geometric[6] library and the random seed of 1234567 for uniformity across all experiments.

3.1 Dataset

The experiments were conducted using three dataset: Cora, CiteSeer, and PPI where Cora and CiteSeer are citation networks for semi-supervised tasks and PPI is the protein-protein interaction graphs corresponding to different human tissues for supervised and generalization task. These dataset are relatively large and provide different interactions between nodes, structure properties, and sparsity. A summary of the datasets used is provided in Table 1. Pytorch Geometric was used to access the datasets.

The Cora dataset contains one graph with 2708 nodes, 5429 edges, 7 different classes for classification, and 1433 node features. The CiteSeer dataset contains one graph with 3327 nodes, 4732 edges, 6 classes and 3703 node features. Each node is labeled. For both the datasets, 20 nodes per class were used for training, 1000 nodes for testing, and 500 nodes for validation. Node features correspond to bad-of-words representation of a document, making it extremely sparse.

The PPI dataset contains 24 graphs with 50 node features. 20 graphs for training, 2 graphs for validation, and 2 graphs for testing. Testing graphs remains unseen during training to observe the generalization ability of AdaGrad. The node features correspond to gene sets and it is not sparse.

Table 1: Summary of the datasets

	Cora	CiteSeer	PPI
# Graphs	1	1	24
# Nodes	2708	3327	2372 (per graph)
# Edges	5429	4732	34,113 (per graph)
# Features	1433	3703	50
# Classes	7	6	121

3.2 Model Architecture

For Cora and CiteSeer, a two layer GCN model with 32 hidden units was employed. Relu activation and dropout with probability of 0.5 was applied between layers. Cross Entropy Loss, a non-convex objective function, was used for multiclass classification.

For PPI, a two layer GraphSAGE model with 256 hidden units followed by a fully connected layer was employed and Relu activation between the layers. GraphSAGE was selected because of its ability to perform inductive learning[4]. Binary Cross Entropy with Logits Loss (aka Logistic Loss), a convex objective function, was used for multilabel classification.

Learning rates for Adam and AdaGrad were set to 0.01 and 0.1 respectively for all experiments. Early stopping techniques were employed to ensure best results and the total number of epochs were set to 50.

4. RESULTS

For Cora and CiteSeer datasets, the results are summarized in Table 2. AdaGrad performs better or as competitive as Adam, but with fewer iterations needed to achieve the results. Because of the sparsity of the data, it was expected AdaGrad would outperform Adam. Furthermore, despite a higher learning rate for AdaGrad, it adapts the learning rate effectively and reaches a solution quicker than Adam. APPENDIX Figure A & B reports the learning and accuracy curves for both datasets.

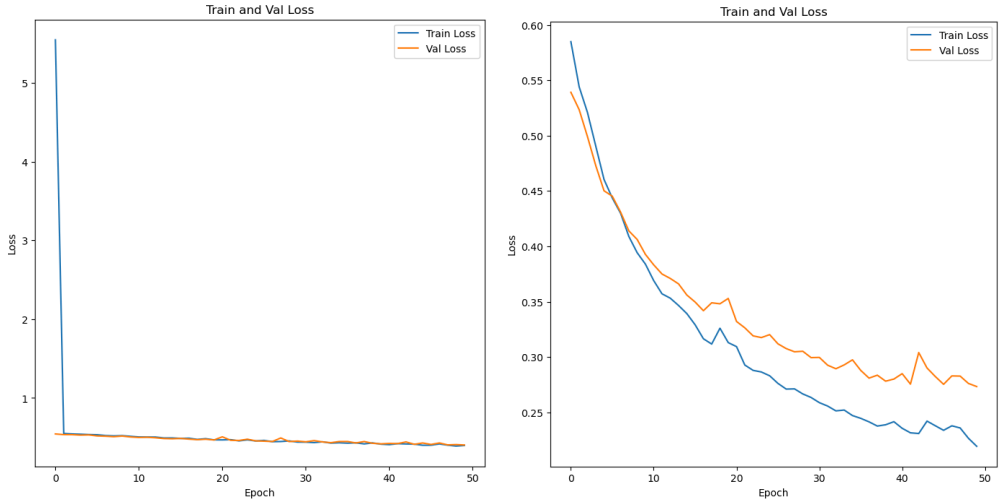
Table 2: Results in terms of accuracies (number of epoch taken) obtained by GCN model, for Cora and CiteSeer.

Optimizer	Cora	CiteSeer
Adam	79.6% (50 epoch)	**70.8% (50 epoch)**
AdaGrad	**81.5% (30 epoch)**	69.0% (30 epoch)

Table 3: Results in terms of micro F1-Score (number of epoch taken) obtained by GraphSAGE model for PPI dataset.

Optimizer	PPI
Adam	0.53
AdaGrad	**0.79**

Figure 1: Learning Curves of AdaGrad (left) and Adam (right) for PPI dataset.



For the PPI dataset, the results are summarized in Table 3. Adam outperforms AdaGrad by a huge margin. From the analysis of the learning curve in figure 1, AdaGrad makes a huge jump and stops learning, while Adam makes incremental progress. This observation shows that AdaGrad is sensitive to the initial conditions of parameters and learning rate. AdaGrad's adaptive learning rate mechanism is a virtue and a vice. The accumulation of gradients in G_t happens from the beginning of the training which could cause the learning rate to approach zero after a few iterations without reaching an optimal solution. Additionally, it is evident from the results that AdaGrad does not perform well for dense data. Further tuning of the model and optimizers could be performed to achieve better results.

5. WHAT I LEARNED/CONCLUSION

This study investigated the suitability of AdaGrad for optimizing the training of Graph Neural Networks (GNNs). The key findings are:

- AdaGrad performs competitively with Adam on sparse data scenarios, such as Cora and CiteSeer. Effectively utilizing the previous gradient information to adapt learning rates accordingly.

- AdaGrad is sensitive to initial conditions and learning rates. Raises a need for hyperparameter tuning.
- AdaGrad's Adaptive mechanism has drawbacks when the data is dense, as observed in PPI datasets.
- Accumulation of gradients overtime can cause the learning rate to approach zero.

In conclusion, this study provides valuable insights into the applicability of AdaGrad for training GNNs, highlighting its strengths in sparse data scenarios and limitations in dense data scenarios. However, AdaGrad is a strong candidate for training GNNs. In the future, more in-depth study could be conducted to further the applicability of AdaGrad for training GNNs.

REFERENCES

[1] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121-2159, Jul. 2011.

[2] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980, Dec. 2014.

[3] Y. Li, T. Zhang, and A. W. Lim, "Graph Isomorphism Networks for Graph Classification," arXiv:1810.00826, Jan. 2019.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," arXiv:1706.02216, Jun. 2017.

[5] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," arXiv:1609.02907, Sep. 2016.

[6] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," arXiv:2008.09020, Aug. 2020.

APPENDIX

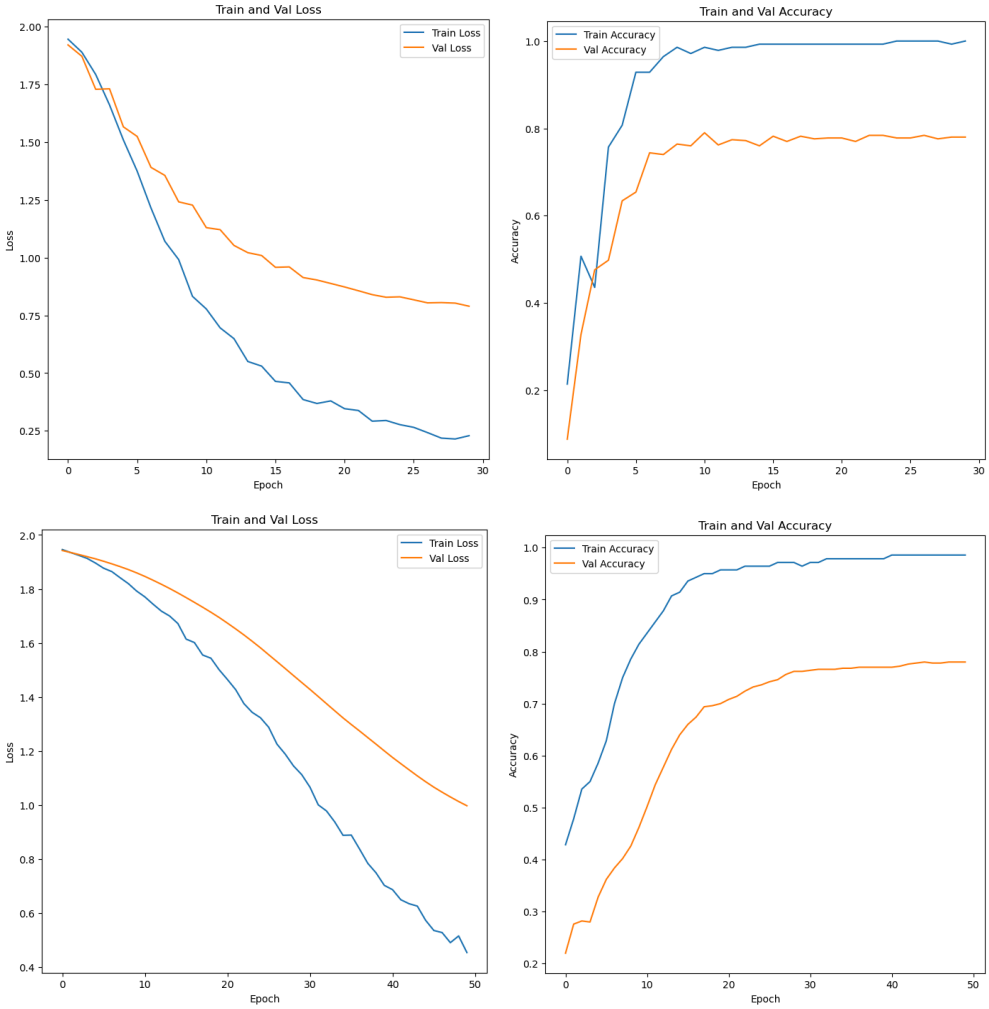


Figure A: Adagrad Learning Curve (top left) and Accuracy Curve(Top Right). Adam Learning Curve (Bottom Left) and Accuracy Curve (bottom right) for **Cora** dataset

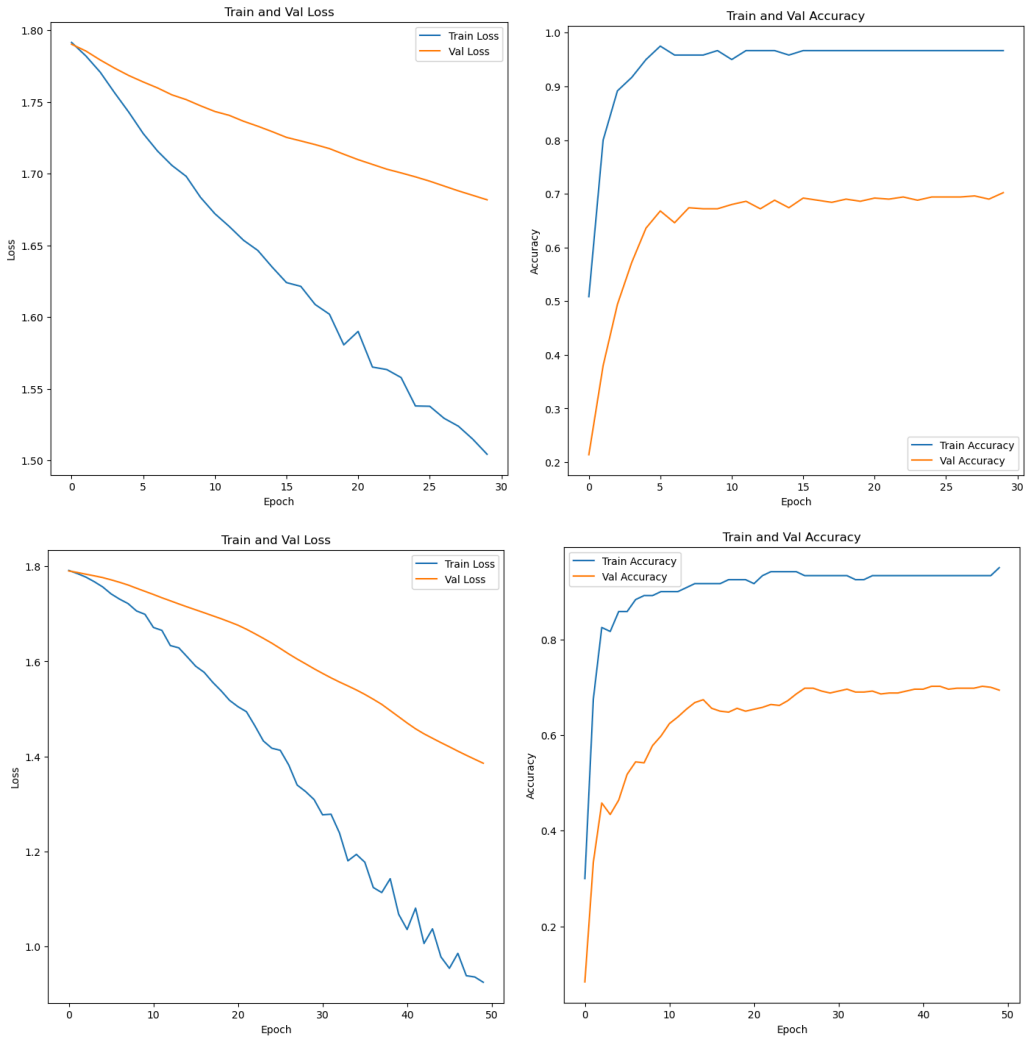


Figure B: Adagrad Learning Curve (top left) and Accuracy Curve(Top Right). Adam Learning Curve (Bottom Left) and Accuracy Curve (bottom right) for **CiteSeer** dataset