Comparative Study of Graph Neural Networks with Different Optimizers

Mobin Ali Momin
110136413
momin41@uwindsor.ca

Atul Kumar
110143931
atulkum@uwindsor.ca

Abdul Rafey Khan
110157539
khan9x1@uwindsor.ca

ABSTRACT

This project examines the performance of several Graph Neural Network (GNN) versions, such as Graph Autoencoders, Graph Convolutional Networks, Graph Attention Networks, and Graph Sample and Aggregation. We assess the impact of several optimization strategies, including ASGD, LBFGS, RMSprop, and advanced approaches like as Adam, RAdam, NAdam, and SNRAdam, on these models. The purpose of the study is to determine how each optimizer affects training patterns and model accuracy. Our findings will provide useful insights into improving GNNs for better performance in graph-based work.

1 INTRODUCTION

Graph Neural Networks have become an influential class of models for learning on graph-structured data. With considerable potential for a variety of applications, including recommendation systems and social network analysis. As the field advances, more GNN designs have been created, each with special strengths for certain kinds of graph-based problems. The four well-known GNN variations that are examined in this project are Graph Autoencoders, Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and Graph SAGE.

Graph Autoencoders are created for unsupervised learning tasks, utilizing graph configurations to obtain meaningful latent representations. GCNs use localized graph convolutions to collect information from surrounding nodes, allowing for efficient and scalable learning on large graphs. GATs provide an attention mechanism that enables nodes to weigh their neighbors, increasing flexibility in recording complicated interactions. This technique is extended by Graph SAGE, which uses samples and aggregates information from a fixed-size neighborhood, enhancing generalization and scalability.

The effectiveness of these GNN variations can be considerably altered by the optimization strategies utilized during training. Optimization procedures are critical in determining the convergence speed, stability, and overall performance of neural networks. This research looks at a variety of optimizers, such as Adaptive Stochastic Gradient Descent (ASGD), Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), Root Mean Square Propagation (RMSprop), Resilient Propagation (Rprop), and advanced adaptive optimizers like Adam, Adagrad, Adadelta, RAdam, NAdam, AdamW, and SNRAdam.

The goal of this project is to develop effective training techniques for various graph-based tasks by carefully examining the influence of these optimizers on the performance of GNN variations. The findings of this project will help to further our understanding of successful GNN training and provide practical assistance for selecting appropriate optimizers to achieve higher model performance in real-world applications.

2 PROBLEM STATEMENT

2.1 Definitions

Graph Neural Networks (GNNs) are essential for processing and interpreting graph-structured data, prevalent in fields like social networks, biological research, finance, and recommendation systems. Each type of GNN, such as Graph Autoencoders (GAE), Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and Graph Sample and Aggregation (SAGE), brings unique benefits and can perform specific tasks involving and utilizing the graphs. However, the success of these GNN models largely depends on the optimization algorithms used during training. Understanding how different optimizers—like Adaptive Stochastic Gradient Descent (ASGD), Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), Root Mean Square Propagation (RMSprop), Resilient Propagation (Rprop), and adaptive methods such as Adam, Adagrad, Adadelta, RAdam, NAdam, AdamW, and SNRAdam—impact the learning process and final outcomes of these GNN variants is crucial. This research aims to systematically analyze these combinations to determine the best training approaches that enhance model accuracy and efficiency.

2.2 Motivation

The motivation for this research stems from the critical role Graph Neural Networks play in solving complex problems involving graph based structured data. Despite the rapid advancements in GNN architectures, there is a lack of comprehensive studies that explore the interaction between these architectures and various optimization algorithms. Optimizers are crucial as they directly impact the convergence rate, stability, and overall performance of the models. By thoroughly understanding these effects, we can improve the deployment of GNNs in real-world applications. For instance, in social network analysis, better-optimized GNNs can more accurately detect communities or predict user behavior. In biological

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

networks, they can enhance the identification of potential drug targets. This study aims to bridge the knowledge gap, providing practical guidelines for selecting the most effective optimizer for a given GNN variant, thereby advancing the field of graph-based machine learning.

2.3 Justification

The justification for conducting this research lies in its potential to significantly improve the training and performance of GNNs, which are increasingly being used in critical applications. Different optimization algorithms offer various benefits and trade-offs. Some optimizers may lead to faster convergence but at the risk of getting trapped in local minima, while others might provide more stable training but require more computational resources. By systematically comparing these optimizers across multiple GNN variants, we can offer valuable insights into their relative performance and suitability for different tasks. This analysis will help in fine-tuning the training process, ensuring that GNNs can be trained more efficiently and effectively. The results of this study will not only contribute to the academic understanding of GNNs and optimizers but also provide practical recommendations for practitioners in the field, leading to more robust and high-performing models in various applications.

3 RELATED WORK

Graph-based models have received a significant attention in the field of machine learning and research, especially for tasks involving structured data such as social networks, biological networks, and recommendation systems. This paper focuses on the analysis of different graph-based models and examines the effect of various optimizers on their performance. The architecture of GNNs has evolved significantly, with key developments such as Graph Convolutional Networks (GCNs) [1], Graph Attention Networks (GATs) [2], each introducing domain specific and diverse ways to aggregate and propagate information across all the nodes.

Previous studies have shown that different optimizers can significantly affect the performance of graph-based models. For example, in the context of GCNs, the choice of optimizer has been shown to influence the accuracy of node classification tasks [12] where low-dimensional embeddings of nodes in large graphs have proved extremely useful in a variety of prediction tasks, from content recommendation to identifying protein functions. Adam has been found to outperform SGD in many cases due to its adaptive learning rate, which helps in handling the sparse and irregular nature of graph data representing the underlying set of pairwise interactions, but much of the structure within these systems involves interactions that take place among more than two nodes at once [6]

Izadi [14] proposed to employ information-geometric tools to optimize a graph neural network architecture such as the graph convolutional networks. More specifically, we develop optimization algorithms for the graph-based semi-supervised learning by employing the natural gradient information in the optimization process, this utilized the natural gradient for the optimization of graph neural networks that can be extended to other semi-supervised problems and represented the comparision between ADAM and SGD

Several comparative studies have benchmarked different optimizers on various GNN architectures and tasks. For instance, a comprehensive study on GATs compared the performance of SGD, Adam, RMSprop, and other optimizers on node classification and graph classification tasks. The results indicated that while Adam generally provided robust performance, RMSprop offered advantages in terms of stability and convergence speed under certain conditions [11]. Moreover, the exploration of more sophisticated optimization techniques, such as LARS (Layer-wise Adaptive Rate Scaling) and LAMB (Layer-wise Adaptive Moments for Batch training), has shown promise in further enhancing GNN training efficiency. Using LARS, we scaled Alexnet up to a batch size of 8K, and Resnet-50 to a batch size of 32K without loss in accuracy. These optimizers have been particularly effective in large-scale GNN training, where traditional optimizers like Adam may struggle with scalability issues [24]

Beyond convergence speed and accuracy, the choice of optimizer also impacts the generalization capabilities of GNNs. Research has indicated that optimizers with adaptive learning rates, such as Adam and RMSprop, can sometimes lead to overfitting, especially in small-scale datasets [17]. In contrast, SGD, with its fixed learning rate schedule, often promotes better generalization, albeit at the cost of slower convergence.

Network modeling with deep neural networks is a another recent topic proposed in this domain [19] [22] with few pioneering attempts. The closest works to our contribution are first Deep-Q [23], where the authors infer the QoS of a network using the traffic matrix as an input using Deep Generative Models. And second [21], where a fully-connected feed-forward neural network is used to model the mean delay of a set of networks using as input the traffic matrix, the main goal of the authors is to understand how fundamental network characteristics (such as traffic intensity) relate with basic neural network parameters (depth of the neural network).

Recent advances in optimization algorithms continue to push the boundaries of GNN performance. Techniques such as Lookahead Optimizer and Ranger, which combines the strengths of multiple optimization methods, have shown promising results in various deep learning tasks, including graph-based models [25]. These hybrid optimizers aim to balance fast convergence with robust generalization, addressing some of the limitations of traditional optimizers. Additionally, the development of meta-learning approaches for optimizer selection and tuning is an emerging area of research. Meta-learning techniques aim to automatically adapt the optimizer parameters based on the specific characteristics of the task and dataset, potentially leading to more efficient and effective training processes for GNNs [16].

The choice of optimizer is a critical factor which influence the performance and efficiency of graph models. While traditional optimizers like SGD and Adam have been extensively studied and researched, emerging optimization techniques and hybrid methods offer new opportunities for enhancing GNN training. This paper contributes to the ongoing research by systematically analyzing the effects of different optimizers on graph-based models, providing insights that can inform the selection and tuning of optimizers in various applications.

4 METHODOLOGY

The central idea of this research is to classify the nodes and assess how different Graph Neural Network (GNN) variants perform with different optimization algorithms and understand the impact of the optimizers on the model performance. We aim to evaluate four type of GNN variants namely Graph Convolution Network (GCN), Graph Attention Network (GAT), Graph Sample and Aggregation (GraphSAGE), and Graph Autoencoders using a range of different optimization methods on the ***Cora*** graph dataset.

As we know, Graph Neural Networks (GNN) are a class of neural networks and designed to operate on graph-structured data. They are quite flexible in terms of handling the irregular and dynamic nature of graphs, where nodes represent entities and edges represent the relationship between them. Graph Neural Networks emerged as a solution when Convolution Neural Networks struggled to produce optimal results due to the arbitrary size of the graph and the presence of complex structure [5]. This is why, they are significantly shaped by Convolution Neural Networks and Graph Embedding techniques.

4.1 Material and Data

To evaluate the performance of different GNN models and optimization algorithms, we used the **Cora** dataset, which is widely used for the development and benchmarking of various algorithms. This dataset comprises 2708 scientific publications, classified into one of seven categories [2]. Each publication is represented by a binary word vector which is also called feature vector. This directed graph contains 5429 edges based on the how papers cited each other.

The Cora dataset is widely used in various applications:

(1) **Node Classification:** This is also the goal of our research to predict the node (scientific paper) belongs to which class. The models like GCN, GraphSAGE, GAT, etc. can be utilized to predict the class of nodes. This can help in organizing academic literature and can be used for content-based recommendation systems.
(2) **Graph Classification:** This involves classification of subgraphs or induced subgraphs. This can help understanding how certain structures in the Cora dataset graph relate to specific categories.
(3) **Link Prediction:** Invloves predicting the link between between the two nodes which are not directly connected yet, but may connect in the future depending on the likelihood score. This is useful for suggesting potential collaborations or predicting future citations.
(4) **Graph Embedding:** This is used to learn low-dimensional embedding of nodes that preserve the topological information of a graph through reconstruction loss. This embedding are useful for the various downstream tasks like node classification. During this research, we used Graph Autoencoders, which does the same mentioned thing. We used the embedding that we get from Graph Autoencoder to classify the nodes.

This dataset could also be used for many other tasks since it gets used for the benchmarking of various algorithms and arouse as a perfect choice for our research.

As a part of our preprocessing steps, we normalized the node feature vectors using the NormalizeFeatures() transformation. This helps to ensure that all node features contribute equally to the learning process by scaling the feature vector to a standard range.

4.2 Proposed Solutions

In this research, we implemented several advanced techniques within the Graph Neural Network (GNN) framework to address the graph-based tasks. Each of the implemented variants has unique approach to process and learn from the graph data (which will be discussed in just a moment). Before that, let's discuss the key characteristics of GNNs architecture:

(1) **Node Embedding [13]:** GNNs are able to learn from different components of graph such as node features, edge features, or the graph overall structure. Each node in the graph can be mapped to low-dimensional vectors which is called feature vector (node embedding) as shown in Figure 1, which can be used for variety of tasks like node classification, link prediction, etc.

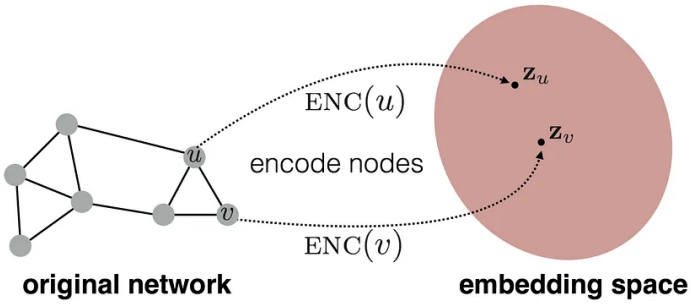


Figure 1: Node Embedding [13]

(2) **Message Passing & Aggregation:** This mechanism allows nodes to exchange the information with the neighboring nodes in a graph. During this process, Every node aggregates the information from the surrounding's nodes and update its feature representation to capture the dependencies and relationships within the graph structure. As we can see in Figure 2, every node is represented by the aggregation results of the surrounding nodes, which helps capturing the local context and relationships.

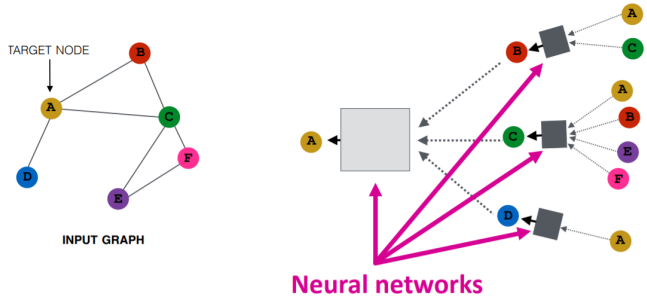


Figure 2: Message Passing & Aggregation [18]

(3) **Embedding Learning:** The node embedding can be used for various tasks such as node classification, link prediction, graph classification, etc (as shown in Figure 3).

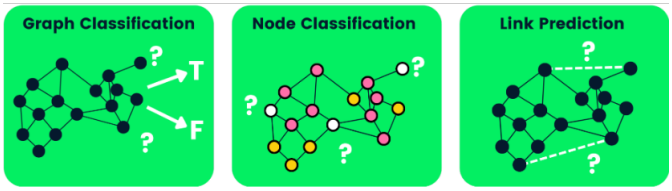


Figure 3: Classification Tasks [5]

4.2.1 ***GNNs Architecture:*** Now let's delve into the details of each of the GNN's implemented variants. We aim to test several optimizers on 4 types of Graph Neural Networks variants namely Graph Convolution Network (GCN), Graph Attention Network (GAT), GraphSAGE, and Graph Autoencoder (GAE). How each of the model works and why we chose are given as follows:

(1) **GCN** can be seen as an extended version of traditional Convolution Neural Networks, where we now perform the convolution operations on graphs instead of images and perform aggregations of neighboring node features to get the node embedding of the target node as discussed above. Each node updates it embedding based on the aggregated value of neighboring nodes, followed by non-linear activation function (ReLU). Like CNN, this model is quite effective in capturing the local graph topology and implemented with multiple convolution layers to learn hierarchical feature representations.
(2) **GAT** assign each neighbor a attention coefficient, which tells how important that node is for the feature update of the target node. This is different from GCN in a way that it considers neighbor importance for the feature update unike GCN which average the features of its neighbors [15]. This attention mechanism enhances the model ability of capture complex relationships between graph nodes.
(3) **GraphSAGE [4]** is also used for learning graphs embeddings for the nodes in a graph like above-discussed GNNs. However, most node embedding algorithms are transductive which means the entire graph need to be present at training time to achieve the good performance. But the advantage of GraphSAGE is that nodes can be added at a later time and can easily be generalized to unseen data, without any need to retrain the model.
(4) **GAE** is unsupervised learning technique of node embedding. This model consists of an encoder that generate embedding of graph data by using neural network architecture and then the decoder that reconstructs the graph structure from these embeddings. The encoder typically uses GCN layers to encode node features into a latent space. Then this embeddings can be used for downstreams tasks like node classification, link prediction, etc.

It is quite important to mention that for GCN, GAT, and GraphSAGE, node classification is performed directly on the Cora graph dataset. We trained these models using labeled dataset to predict the class of node based on the features. However, due to the unsupervised nature of GAE, we adopted a different approach. Unlike other GNN variants, it doesn't directly predict the class, instead it encodes the graph and node features into lower dimensional space which we call as embedding. Then we pass these embedding through the linear layer to perform node classification.

4.2.2 ***Optimization Algorithms:*** We explore a variety of carefully selected optimizers to understand their effect on the performance of GNN models. They are given as following:

- **Tradition Optimizers:** This includes traditional and well-established optimizers such as Stochastic Gradient Descent (SGD) and Adaptive Stochastic Gradient Descent (ASGD). ASGD can be considered as enhanced version of SGD. ASGD has the adaptive learning, which means it could be adjusted for each paramter based on the magnitude of gradient. Also, ASGD converges faster, can escape local minima, and handle plateaus better than SGD. This study would give us an opportunity to test both of them in the same settings and verify the claim [9].
- **Advanced Optimizers:** This includes advanced and more recent optimizers such as:
  - Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) [1]: This uses an approximation of Hessian matrix to improve convergence and accuracy and being a memory-efficient algorithm, it is suitable for large-scale optimization problems.
  - Root Mean Square Propagation (RMSprop) [7]: It's also one of the fast and popular optimizer for the deep learning models. As it is quite fast to go through the saddle point than other by scaling the learning rate.
  - Adam and its variants [3]: These optimizers including Adam, Adagrad, Adadelta, RAdam, NAdam, AdamW, and SNRAdam, enhance tarining efficiency and model performance by adjusting learning rates dynamically. The approach to adjust the learning rate vary optimizer by optimizer, but it's just to improve the convergence speed and stability across different types of data and training conditions.

4.2.3 ***Highlights and Implications:*** In essence, the combination of various GNN models and optimization algorithms, including some recent ones, enables a comprehensive analysis of their performance on graph-based tasks. As we know, each of the GNN variant, we are using, process and learning from the data in a unique way which we couple with the range of traditional and advanced optimization algorithms to enhance the overall capabilities of the model and form the basis of our comparative study.

The insights that we get from this research would help us understand how different architectures and optimizers interact to affect the model accuracy and efficiency. After doing in-depth analysis, it helps us understand the most effective approach out of the all implemented.

4.3 Conditions and Assumptions

4.3.1 ***Conditions:***

(1) Firstly, all the models are trained with consistent hyperparameters settings. They may vary model to model, however, required that particular model have same hyperparameters for all the optimizers. Hyperparamters include learning rates, dropout rates for the GNNs layers, and batch sizes, to make sure that different optimizers can be compared across different experiments.
(2) Every model with each optimizer is trained for a predetermined number of epochs, with early stopping criteria implemented to prevent overfitting and make the model more robust.

4.3.2 *Assumptions:*

(1) First assumption that we are making is that the **Cora** dataset is representative of a typical graph like datasets and contains enough variations in terms of feature vectors of nodes and real-world like edges collection.
(2) Second assumption we are making about our model architecture that it is appropriate for the task and has sufficient layers to capture best complex structures. Although, we performed trail and error method to come up with the numbers of layer that we currently have in our models.

4.4 Formal Complexity

Since every implemented variant has unique approach to process and learn from the graph data, and that's why the complexities are also different for each of them. The complexity of all the variants mainly depend on the numbers of layers, node, edges, and feature vector.

(1) **GCN [8]:** The training and inference time complexity of GCN layers can be bounded by $\mathcal{O}(L|E|F + L|V|F^2)$, where L represents number of layers, $|V|$ & $|E|$ denote the number of nodes and edges in graph, and F denotes the dimension of node's feature vector. It consists costs of sparse-dense matrix multiplication and feature transformation.
(2) **GAT [21]:** The time complexity for a single attention head in a GAT model, when computing F' features, can be described as $\mathcal{O}(|V|FF' + |E|F')$. Here again, F represents number of input feature and $|V|$ & $|E|$ denote the number of nodes and edges in graph.
(3) **GraphSAGE [10]:** The computational complexity of one batch of GraphSAGE is given as $\mathcal{O}(bLd^2k^L)$, where b is the number of training nodes, L is the number of layers or hops, d denotes the number of features, and k is the number of sampled neighbors per hops.
(4) **GAE [20]:** The overall complexity of the autoencoder is $\mathcal{O}(N)$, where N is the number of nodes. After this, we are passing through the embedding through a linear layer which has the computational complexity of $\mathcal{O}(n \cdot m)$, where n and m are the number of input and output features respectively. This means the total complexity of both the operations are $\mathcal{O}(N + n \cdot m)$.

5 COMPUTATIONAL EXPERIMENTS

5.1 What Experiments

The objective is to evaluate a variety of carefully chosen optimizers including some of the latest ones on the above-mentioned GNN architectures. As we know, selection of appropriate optimizers for a model is quite important as it determines how effectively and efficienty the model is learning from data. By testing a range of optimizers, including some latest ones, we aim to gain insights on the overall model performance.

During testing, we looked out for the following parameters to understand the effects of different optimization algorithms:

(1) Firstly we look out for the effects of optimizers on the convergence rate during training. To better generalize the model, we have implemented early stopping method, so it stops once it starts overfitting. As we will discuss in the results in detail, model for some of the optimizers have converged fast and for some of them it had to go through all the epochs.
(2) Secondly, we evaluated the GNN models overall performance with different optimizers. By comparing the optimizers within the same model, we identified which ones lead to better accuracy and generalization of the model.
(3) Lastly, we assess how each optimizer affects the training time across the models, since different optimizers may have varying computational efficiencies.

We analyzed the impact of different optimizers on the performance of a Graph Convolutional Network (GCN) when applied to the Cora dataset, a commonly used benchmark dataset for graph-based models. We have created Graph based models like GCN, GAT, GraphSage, GAE constructed with convolutional, attention, embedding and activation layers. The layers usually have 128, 64, 16, and the final layer has 7 output units corresponding to the number of classes. Each layer is followed by a ReLU activation and dropout for regularization. The model is trained using various optimizers, including ASGD, LBFGS, RMSprop, Rprop, Adam, Adagrad, Adadelta, RAdam, NAdam, AdamW, and SNRAdam. Each optimizer is configured with a learning rate of 0.01 to test in a common ground and other hyperparameters specific to the optimizer. Early stopping is implemented with a patience parameter of 10 epochs to prevent overfitting and to save the best model based on validation loss. The training process involves calculating the loss and updating the model weights accordingly. The model's performance is evaluated at each epoch on the validation set where we progress and apply early stopping to ignore long training time.

5.2 What Evaluation Metrics

The primary evaluation metric used in this study is Accuracy, Precision, and Recall, which provide a comprehensive view of the model's classification performance.Accuracy is the ratio of correctly predicted labels to the total number of labels. It is a straightforward measure of model performance in classification tasks. The formula for accuracy is given by

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It is a measure of the accuracy of the positive predictions

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

Recall (also known as Sensitivity or True Positive Rate) is the ratio of correctly predicted positive observations to all the observations in the actual class.

$$\text{Recall} = \frac{Truepositive}{Truepositive + Falsenegative}$$

we are calculating accuracy for the training, validation, and test sets to provide a comprehensive view of the model's performance.

- Training Accuracy: It measures how well the model fits the training data. High training accuracy indicates that the model has learned the patterns present in the training set.
- Validation Accuracy: It is used to evaluate the model's performance on unseen data during the training process. It helps in tuning hyperparameters and selecting the best model while preventing overfitting.
- Test Accuracy: This assesses the model's ability to generalize to completely unseen data. High test accuracy indicates that the model performs well on new, previously unseen examples.

Evaluating the model with Accuracy, Precision, and Recall is crucial due to the potential imbalance in class distributions. Certain classes may have significantly fewer samples than others, leading to a situation where a model could achieve high accuracy by favoring the majority class while poorly predicting the minority classes. Hence, relying on accuracy alone can be misleading, So we brought Precision, Recall, F1-Score into the picture as it can observed in the 4

5.3 Implementation Details

The computational experiments for this study are executed using Google Colab, a cloud-based platform that provides a robust environment for machine learning tasks. Google Colab offers the convenience of free access to GPUs, which are crucial for accelerating the training and evaluation of deep learning models, particularly for complex Graph Neural Networks (GNNs). This setup ensures that the experiments can be run efficiently, even when dealing with large-scale datasets and intricate models.

The dataset used in this study is the Cora dataset, a well-established benchmark in the domain of graph-based learning. The Cora dataset consists of a citation network where nodes represent scientific papers and edges denote citation relationships between them. It includes node features, edge indices, and pre-defined masks for training, validation, and testing. The node features are used as input to the GNN models, while the edge indices define the graph structure. The masks are employed to partition the data into training, validation, and test sets, ensuring that the models are evaluated on unseen data.

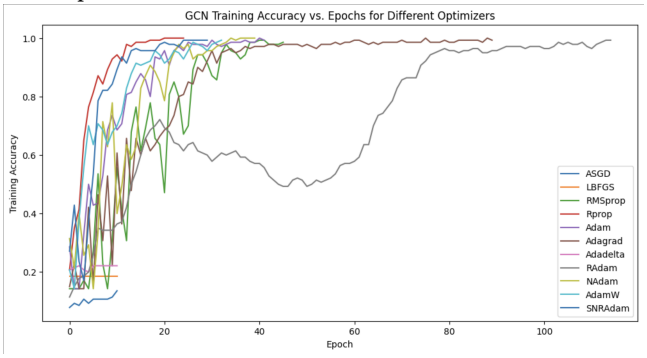
The implementation is carried out using Python, a versatile and widely-used programming language in the machine learning community. For the deep learning aspect of the experiments, we rely on PyTorch, a powerful framework known for its flexibility and dynamic computational graph capabilities. PyTorch is particularly well-suited for developing GNNs due to its extensive support for tensor operations and its modular design, which allows for the easy construction and training of neural network architectures.

In addition to PyTorch, several Python libraries are utilized to facilitate data handling, visualization, and analysis. Pandas (pd) is used for efficient manipulation and analysis of the dataset, allowing for operations such as merging, filtering, and aggregating data. matplotlib.pyplot is employed to create visualizations of training and bar-charts for test and train accuracy's, accuracy plots of each optimizers against epochs, which are crucial for monitoring model performance throughout the training process. numpy is used for numerical operations and to handle array-based computations, supporting various mathematical operations required during the experiments.

For graph visualization and analysis, Networkx is used. Networkx provides tools to visualize the graph structure of the Cora dataset, which helps in understanding the topology of the graph and assessing the impact of different GNN architectures on the graph's structure. This visualization aids in interpreting model performance and understanding how well the models capture the underlying graph patterns.

5.4 Results and Discussion

5.4.1 ***Graph Convolution Network***. We tested the Graph Convolution Network model with optimizers mentioned in section 4.2.2. We ran all the 11 mentioned optimizers for the training and testing purposes. The epoch that we predefined for the model for 200 along with the implementation of Early Stopping, so if it sees data is overfitting, we stop the training. The line chart below tells how the training accuracy is varying of the GCN model with different optimizers:



GCN Training Accuracy for Different Optimizers

As we can see, GCN with optimizers like ASGD, LBFGS, and Adadelta perform the worst out of all on the training set and so did on the test set as shown in Figure 5 and have early stopped due to issue of overfitting while training. This plot also tells us early stopping is correctly implemented and have stopped training when it has seen overfitting in the model. All the adam and its variants have performed well on training set and a bit less on the test set (may be due to some overfitting). Surprisingly, Adagrad and

	GCN				GAT				Graph SAGE				GAE			
Optimizer	Test Accuracy	Test Precision	Test Recall	Test F1-Score	Test Accuracy	Test Precision	Test Recal	Test F1-Score	Test Accuracy	Test Precision	Test Recal	Test F1-Score	Test Accuracy	Test Precision	Test Recall	Test F1-Score
ASGD	0.12	0.10	0.12	0.09	0.23	0.14	0.23	0.17	0.14	0.02	0.14	0.04	0.10	0.01	0.10	0.02
LBFGS	0.15	0.09	0.15	0.06	0.12	0.11	0.12	0.08	0.09	0.01	0.09	0.02	0.13	0.08	0.13	0.04
RMSprop	0.81	0.81	0.81	0.81	0.15	0.02	0.15	0.04	0.76	0.77	0.76	0.75	0.10	0.01	0.10	0.02
Rprop	0.79	0.80	0.79	0.79	0.78	0.80	0.78	0.78	0.76	0.77	0.76	0.76	0.10	0.01	0.10	0.02
Adam	0.77	0.79	0.77	0.77	0.79	0.80	0.79	0.79	0.62	0.64	0.62	0.61	0.11	0.03	0.11	0.04
Adagrad	0.81	0.82	0.81	0.81	0.78	0.80	0.78	0.77	0.73	0.74	0.73	0.73	0.14	0.04	0.14	0.05
Adadelta	0.18	0.17	0.18	0.08	0.06	0.20	0.06	0.06	0.32	0.10	0.32	0.15	0.10	0.01	0.10	0.02
RAdam	0.75	0.77	0.75	0.75	0.76	0.79	0.76	0.76	0.29	0.17	0.29	0.21	0.10	0.01	0.10	0.02
NAdam	0.79	0.80	0.79	0.79	0.76	0.79	0.76	0.74	0.76	0.78	0.76	0.76	0.10	0.01	0.10	0.02
AdamW	0.79	0.80	0.79	0.79	0.77	0.79	0.77	0.77	0.71	0.74	0.71	0.71	0.10	0.01	0.10	0.02
SNRAdam	0.79	0.80	0.79	0.79	0.71	0.78	0.71	0.71	0.73	0.74	0.73	0.73	0.10	0.01	0.10	0.02

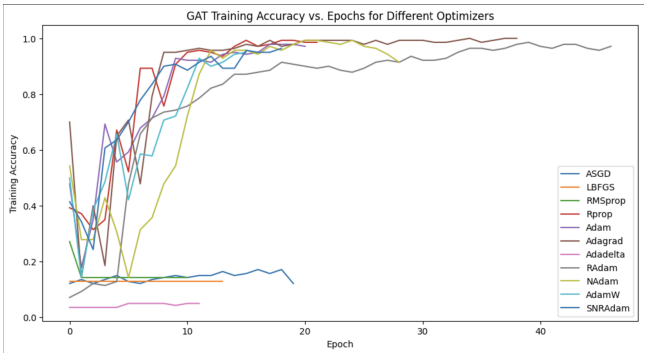
Figure 4: Evaluation Metrics

RMSprop has surpassed the Adam and all its variants in terms of both training and test accuracy.

RAdam has taken 115 epochs in total to converge to the solution and still achieved low accuracy on test set compared to the well performing models such as Adagrad, RMSprop, etc. and so do in terms of test accuracy in Figure 5. This is clear from lower training and test accuracy of these models that they may not be able to handle high dimensional graph data. This may due to the reasons such as stochastic nature of ASGD, dynamic learning rate of Adadelta, etc.

Rprop and SNRAdam have performed almost the same in terms of both test and training accuracy and converged to solution in almost same number of epochs.

5.4.2 ***Graph Attention Network:*** We tested the Graph Attention Network model with same optimizers again. We ran the GAT model with different optimizers and for maximum of 200 epochs with Early Stopping implemented. The line chart below tells how the training accuracy is varying of the GAT model with different optimizers.

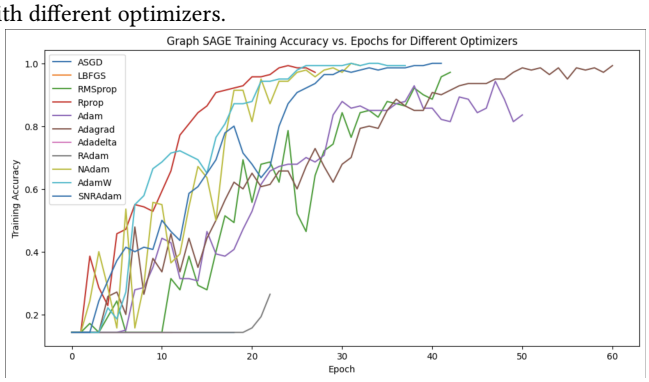


GAT Training Accuracy for Different Optimizers

Again, ASGD, LBFGC, Adadelta perform the worst on both the training and test sets as shown in Figure 7. Surprisingly this time, RMsprop optimizer has also caused low accuracy score around 15 % on the test set (which was the best performing in the case of GCN).

Other than these 4 worst performing models in the case of GAT, rest of the models were able to achieve almost the same accuracy. This time, Adam has surpassed all the models in terms of training accuracy which is in just 21 epochs. AdamW has also achieved almost the same accuracy as Adam but just in 16 epochs which is kind of surprising as it converged to solution that quickly,

5.4.3 ***GraphSAGE:.*** We tested again the same optimizers on the GraphSAGE model with maximum of 200 epochs. The line chart below tells how the training accuracy is varying of the GAT model with different optimizers.



GraphSAGE Training Accuracy for Different Optimizers

Again, ASGD, LBFGS, and Adadelta performed the worst on both the training and testing set shown in Figure 7. This time, accuracy for the RAdam, on the test set, fell down to only 29%.

RMSprop, Rprop, and NAdam optimizers have achieved the highest of all accuracy and exactly the same. Even though, they all achieved the same accuracy but Rprop converged quickly than any other model, so it's computationally efficient and effective optimizer to use with GraphSAGE for the datasets like **Cora**. Point to be acknowledged, Adam which considered to be the best in many cases as an optimizer, this time it just got up to an accuracy of 62% on the test set and with 51 epochs, which is quite interesting to see.

5.4.4 ***Graph Autoencoders:*** As we discussed in GNNs architecture, GAE is the unsupervised learning technique, which means we can't perform node classification directly using GAE layers. So we followed the methodology where we were encoding (using GAE layers) the representation learned through convolution layers. Once we have the embedding from the GAE layer, we were passing through a classifier containing only one linear layer. However, this approach failed badly and may not support the node classification on the **Cora** dataset.

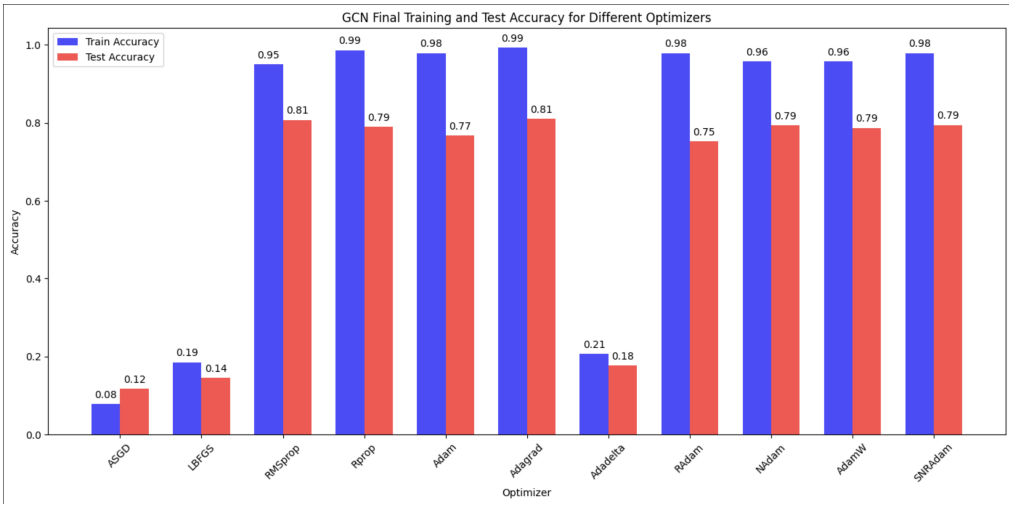


Figure 5: Training and Test Accuracy for the GCN Model for Different Optimizers:

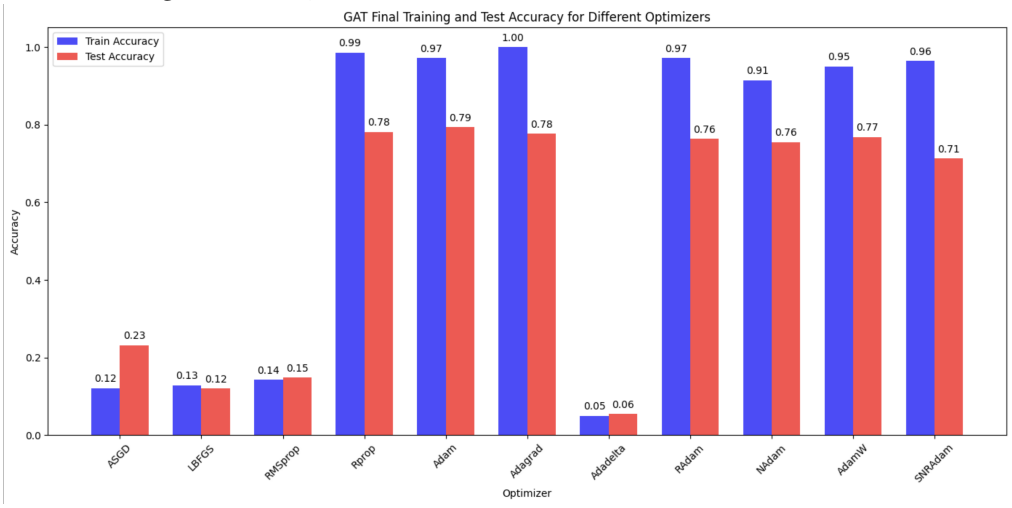


Figure 6: Training and Test Accuracy for the GAT Model for Different Optimizers:

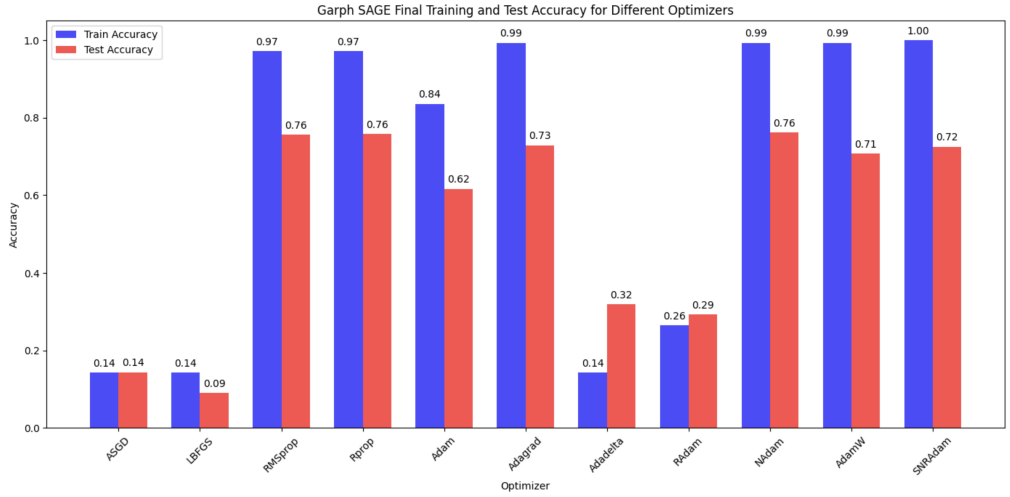
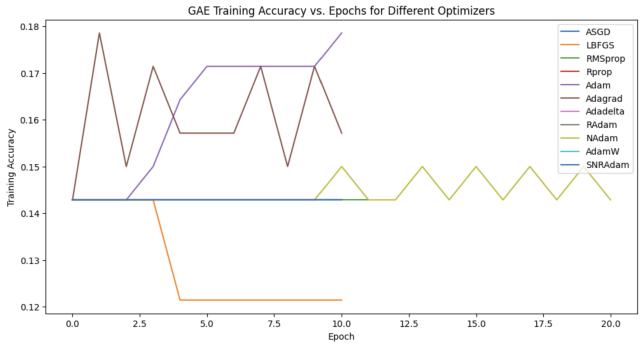


Figure 7: Training and Test Accuracy for the GraphSAGE Model for Different Optimizers

As we can see, there are very abnormal patterns which shouldn't be happening during learning:



GAE Training Accuracy for Different Optimizers

All the optimizers on the GAE model just ran for the 11 epochs except NAdam because Early Stopping has stopped the model learning there because of may be not much learning. NAdam survived until 11 epochs however couldn't bring any higher accuracy to table compared to another optimizers, as it can be observed in 8

We tested Graph Autoencoders with different number of layers, but still lead to this low accuracy. So, we will this as an opportunity to improve and test on the different datasets in the future to analyze the effectiveness of our approach.

6 CONCLUSION

6.1 Summary

In this paper, we analyzed the performance of variants of graph neural network models using different optimization algorithms to understand the impact of each optimizer on model accuracy, precision, and recall. Utilizing the well-known Cora dataset. Our findings indicate that different optimizers can significantly affect the performance metrics of GNN models. For instance, the Adam and latest SRNAdam optimizer demonstrated superior performance in terms of both accuracy and convergence speed, aligning with previous research highlighting its potency in training graph models

6.2 Future Work

(1) **Exploration of Additional GNN Variants:** While this study focuses on Graph Autoencoders, Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and Graph SAGE, there are numerous other GNN variants that warrant investigation. Future work could include exploring architectures such as Graph Isomorphism Networks (GINs), Graph Neural Networks with Edge Features (GNN-EFs), and higher-order GNNs. Each of these models may offer unique advantages for specific types of graph-based problems.

(2) **Extended Evaluation of Optimizers:** The current study evaluates a selection of optimization algorithms. Future research could expand this evaluation to include additional optimizers, such as those based on second-order methods or meta-learning approaches. Comparing these with the ones studied may provide further insights into how various optimization technique impact GNN performance.

(3) **Incoportation of Real-World Datasets:** While Cora dataset is a standard benchmark, future work could involve evaluating GNN models on real-world datasets with more complex and diverse graph structures. This would test the models' robustness and generalized ability across different domains, such as social network, biological networks, or large-scale knowledge graphs.

(4) **Hyper-parameter Optimization:** The experiments in this study use fixed hyper-parameter for both GNN architectures and optimization algorithms. Future research could involove systematic hyper-parameter tuning to optimize model performance further. Techniques such as grid search, random search, or Bayesian optimization could be employed to find the most effective hyper-parameter configurations.

(5) **Scalability and Efficiency Improvements:** Investigating methods to improve the scalability of GNN's particularly for very large graphs, is crucial. This includes exploring techniques for distributed training, reducing memory consumption, and optimizing computational efficiency. Future work could focus on developing and evaluating approaches to handle large-scale graph data more effectively.

6.3 Open Problem

(1) **GAE inconsistency with node classification:** GAE's are designed for unsupervised learning falls short on domain of node classification. They reconstruct the adjacency matrix of the graph and capture the structure of the graph rather than the node features and hence we are getting low train and test accuracy scores, which remain and open challenge and can be solved by converting data into expected formats

(2) **Handling Dynamic Graphs:** Many real-world graphs are dynamic, meaning their structure evolves over time. Most existing GNN models assume static graphs, and extending these models to handle dynamic graphs remains an open challenge. Research is needed to develop GNN architectures and optimization strategies that can adapt to changing graph structures.

(3) **Robustness to Adversarial Attacks:** Like other neural networks, GNNs are susceptible to adversarial attacks, where small disturbance in input data can lead to incorrect predictions. Investigating and developing robust GNN models that can withstand adversarial manipulations is an important area for future research.

(4) **Integration with Other Machine Learning Techniques:** Combining GNNs with other machine learning techniques, such as reinforcement learning or transfer learning, presents opportunities for enhancing model performance and versatility. Exploring these integration's and their benefits for graph-based tasks could open new research directions.

(5) **Ethical Considerations and Fairness:** Ensuring that GNN models are fair and do not perpetuate or amplify biases present in the data is critical. Future work should address ethical considerations, including bias mitigation and fairness, to ensure that GNNs are used responsibility and equitably.

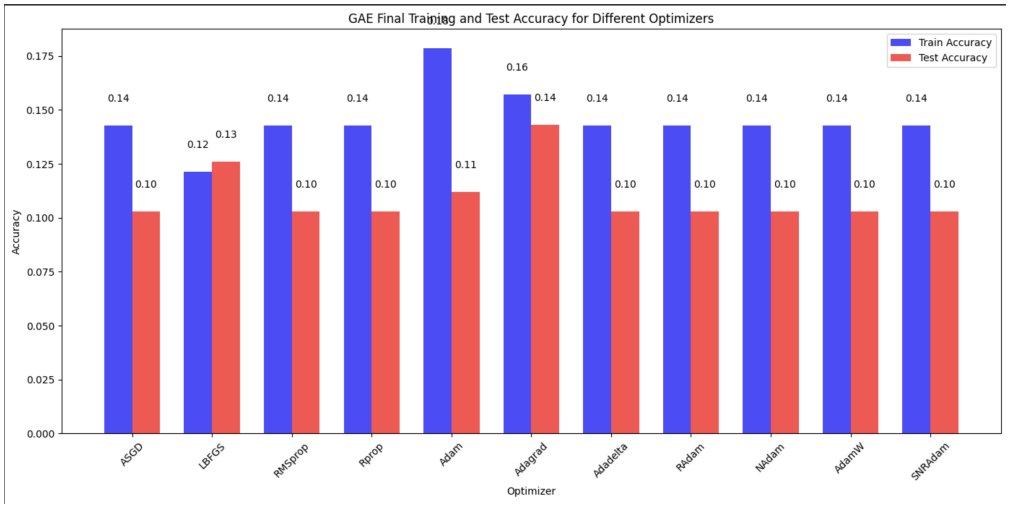


Figure 8: Training and Test Accuracy for the GAE Model for Different Optimizers

REFERENCES

[1] 2023. Limited-memory BFGS. (2023). https://en.wikipedia.org/wiki/Limited-memory_BFGS Accessed: 2024-07-19.

[2] 2024. Cora Dataset. (2024). https://www.geeksforgeeks.org/cora-dataset/ Accessed: 2024-07-15.

[3] 2024. What is Adam Optimizer? (2024). https://www.geeksforgeeks.org/adam-optimizer/ Accessed: 2024-07-21.

[4] Nabila Abraham. 2020. OhMyGraphs: GraphSAGE and inductive representation learning. (2020). https://medium.com/analytics-vidhya/ohmygraphs-graphsage-and-inductive-representation-learning-ea26d2835331 Accessed: 2024-07-17.

[5] Abid Awan. 2022. A Comprehensive Introduction to Graph Neural Networks (GNNs). (2022). https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial Accessed: 2024-07-15.

[6] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[7] Vitaly Bushaev. 2018. Understanding RMSprop — faster neural network learning. (2018). https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a Accessed: 2024-07-20.

[8] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Scalable Graph Neural Networks via Bidirectional Propagation. (2021). arXiv:cs.LG/2010.15421 https://arxiv.org/abs/2010.15421

[9] Ruijuan Chen, Xiaoquan Tang, and Xiuting Li. 2022. Adaptive Stochastic Gradient Descent Method for Convex and Non-Convex optimization. *Fractal Fract* 6, 12 (2022), 709. https://doi.org/10.3390/fractalfract6120709

[10] Fabrizio Frasca and Michael Bronstein. 2021. Simple scalable graph neural networks. (2021). https://blog.x.com/engineering/en_us/topics/insights/2021/simple-scalable-graph-neural-networks Accessed: 2024-07-23.

[11] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *international conference on machine learning*. PMLR, 2083–2092.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[13] Omar Hussein. 2022. Graph Neural Networks Series | Part 3 | Node embedding. (2022). https://medium.com/the-modern-scientist/graph-neural-networks-series-part-3-node-embedding-36613cc967d5 Accessed: 2024-07-17.

[14] Mohammad Rasool Izadi, Yihao Fang, Robert Stevenson, and Lizhen Lin. 2020. Optimization of graph neural networks with natural gradient descent. In *2020 IEEE international conference on big data (big data)*. IEEE, 171–179.

[15] Farzad Karami. 2023. Understanding Graph Attention Networks: A Practical Exploration. (2023). https://medium.com/@farzad.karami/understanding-graph-attention-networks-a-practical-exploration-cf033a8f3d9d Accessed: 2024-07-16.

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[17] Ilya Loshchilov, Frank Hutter, et al. 2017. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101* 5 (2017).

[18] Amal Menzli. 2023. Graph Neural Network and Some of GNN Applications: Everything You Need to Know. (2023). https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications Accessed: 2024-07-23.

[19] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Unveiling the potential of graph neural networks for network modeling and optimization in SDN. In *Proceedings of the 2019 ACM Symposium on SDN Research*. 140–151.

[20] Phi Vu Tran. 2018. Learning to Make Predictions on Graphs with Autoencoders. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. https://doi.org/10.1109/dsaa.2018.00034

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. (2018). arXiv:stat.ML/1710.10903 https://arxiv.org/abs/1710.10903

[22] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. 2017. Machine learning for networking: Workflow, advances and opportunities. *Ieee Network* 32, 2 (2017), 92–99.

[23] Shihan Xiao, Dongdong He, and Zhibo Gong. 2018. Deep-q: Traffic-driven qos inference using deep generative network. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 67–73.

[24] Yang You, Igor Gitman, and Boris Ginsburg. 2017. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888* (2017).

[25] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. 2019. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems* 32 (2019).