Geometric Deep Learning: Understanding Graph Neural Networks through the Lens of Mathematics

Harshda C. Gore*, Shailesh P. Dhome

Department of Mathematics, Dr. D. Y. Patil Arts, Commerce & Science College, Pimpri, Pune-18, Maharashtra, India

***Corresponding Author**

DOI: https://doi.org/10.51583/IJLTEMAS.2025.1413SP043

Received: 26 June 2025; Accepted: 30 June 2025; Published: 27 October 2025

Abstract: Geometric Deep Learning (GDL) extends traditional neural network paradigms to non-Euclidean data structures, enabling the effective processing of data that lies on manifolds or graphs. Among GDL techniques, Graph Neural Networks (GNNs) have emerged as powerful tools for modelling relational data by leveraging principles from graph theory and algebraic topology. This paper explores GNNs through the lens of mathematics, focusing on how geometric and topological insights drive the architecture and functionality of these networks. By framing GNNs in terms of graph signal processing and spectral theory, we illuminate how GNNs capture dependencies across nodes and edges, offering a structured approach to learning on graph-structured data. We further examine the theoretical underpinnings that make GNNs particularly suited for applications in social networks, molecular biology, and recommendation systems. In doing so, this study provides a mathematical perspective on the capabilities and limitations of GNNs, underscoring the role of invariance, equivariance, and generalization within graph-based learning models.

Keywords: Geometric Deep Learning, Graph Neural Networks, Non-Euclidean Data, Algebraic Topology, Graph Theory

I. Introduction

In recent years, the field of deep learning has experienced a paradigm shift towards geometric deep learning, which seeks to generalize neural network architectures to non-Euclidean domains, particularly graphs and manifolds. Traditional deep learning models primarily operate on Euclidean spaces, where data points are arranged in grid-like structures such as images or sequences. However, many real-world applications involve complex relationships and dependencies that can be naturally represented as graphs, including social networks, biological networks, and recommendation systems. Graph Neural Networks (GNNs) have emerged as a powerful framework within this context, designed to process data structured as graphs by leveraging the inherent relationships between nodes and edges. GNNs incorporate principles from graph theory, allowing for the learning of representations that capture both local and global structural information. These networks have shown remarkable effectiveness in tasks such as node classification, link prediction, and graph classification, demonstrating their versatility across various domains. Mathematically, GNNs draw from concepts in linear algebra, spectral graph theory, and topology to formulate operations that are invariant to graph isomorphism, ensuring that the learned representations are robust to variations in graph structure.

This paper delves into the mathematical foundations of GNNs, elucidating how geometric insights inform their design and operation. By examining these connections, we aim to provide a comprehensive understanding of GNNs, paving the way for future advancements in geometric deep learning.

Mathematical Foundations of Graph Neural Networks

The Mathematical Foundation of Gradient Descent

Graph Neural Networks (GNNs) utilize mathematical concepts from graph theory and linear algebra to process and learn from graph-structured data. The fundamental building blocks of GNNs include nodes (vertices), edges (connections between nodes), and graph embedding's, which represent the features associated with nodes or entire graphs. The learning process involves aggregating information from neighbouring nodes to generate embedding's that capture local connectivity patterns and structural properties. This is typically achieved through message-passing frameworks, where nodes update their representations based on the features of their neighbours. Mathematically, this can be framed as:

$h_v^{(k)} = \sigma (W^{(k)} \cdot \text{AGGREGATE} (\{ h_v^{(k-1)} : u \in N(v) \}))$

where $h_v^{(k)}$ denotes the hidden representation of node v at the kth iteration, $W^{(k)}$ is a learnable weight matrix, and N(v) represents the neighbours of node v. The aggregate function can take various forms, including summation, mean, or max pooling, depending on the specific GNN architecture.

Types of Graph Neural Networks

Several variants of GNNs have been proposed, each designed to address specific challenges and leverage different mathematical principles:

Graph Convolutional Networks (GCNs): GCNs extend the convolutional neural network framework to graphs by defining convolution operations on the graph structure. They employ localized filters to capture features from neighbouring nodes, making them particularly effective for semi-supervised learning tasks.

Graph Attention Networks (GATs): GATs introduce attention mechanisms to GNNs, allowing nodes to weigh the importance of neighbouring nodes differently during aggregation. This enables the model to focus on more relevant neighbours, enhancing representation quality.

GraphSAGE: This approach employs a sampling method to aggregate information from a fixed-size neighbourhood, making it scalable for large graphs. By utilizing various aggregation functions (mean, LSTM, pooling), GraphSAGE can effectively learn representations without requiring the full graph during training.

Applications of Graph Neural Networks

GNNs have been successfully applied across diverse fields, demonstrating their versatility:

Social Network Analysis: GNNs can analyse relationships between users and predict user behaviour, such as recommending friends or predicting engagement patterns

Biological Networks: In bioinformatics, GNNs are used to model protein-protein interaction networks, aiding in drug discovery and understanding disease mechanisms

Recommendation Systems: GNNs facilitate collaborative filtering by capturing user-item interactions as graphs, improving recommendation accuracy and personalization

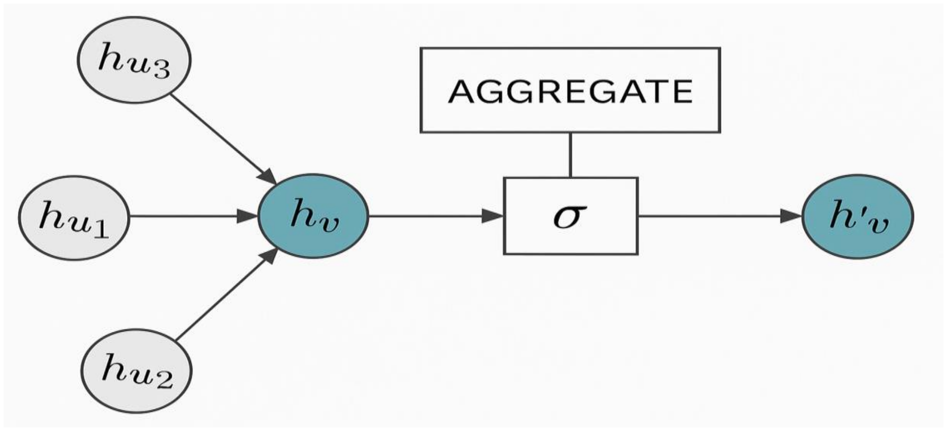
Natural Language Processing: In NLP tasks, GNNs can model the relationships between words or sentences, enhancing tasks such as sentiment analysis and semantic understanding

Momentum-based Gradient Descent

Momentum is an extension to gradient descent that accelerates convergence by incorporating a memory of past gradients. It helps in smoothing out oscillations, especially in steep regions of the loss surface. The momentum based gradient descent algorithm help to find the saddle point of particular feature. We use saddle point to reduce the complexity of dataset. The formula SGD for linear regression w- w[1]

$$v_{t+1} = \beta v_t + \alpha \nabla_\theta L(\theta)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$



where v_t is the velocity term, and β (typically around 0.9) determines the contribution of past gradients. Momentum enables the algorithm to maintain consistent updates in valleys and around saddle points, speeding up convergence.

Comparative Analysis of Gradient Descent Variants

Each variant of gradient descent has unique advantages and limitations. Standard gradient descent offers stable convergence but may be inefficient for large datasets. SGD introduces speed but can be noisy. Mini-Batch Gradient Descent balances these factors, making it a common choice for deep learning. Momentum and adaptive methods like AdaGrad, RMSprop, and Adam enhance the efficiency of gradient descent by incorporating historical information to address oscillations, variable gradients, and non-stationary data. Adam's combination of momentum and adaptive learning rates often achieves fast, smooth convergence, making it the preferred choice in large-scale, high-dimensional deep learning tasks.

II. Practical Considerations in Choosing an Optimization Method

The choice of gradient descent variant often depends on the specific characteristics of the problem, including dataset size, computational resources, and model complexity. While traditional gradient descent or mini-batch methods are suitable for simpler models or smaller datasets, Adam and RMSprop are more effective for complex, non-convex loss surfaces like those encountered in deep neural networks. Proper tuning of parameters, particularly the learning rate, is essential in optimizing the performance of each gradient descent variant.

Challenges and Limitation

While gradient descent and its variants have become essential in machine learning, they also present several challenges and limitations:

Choice of Learning Rate

The selection of an appropriate learning rate is critical for successful convergence. A high learning rate can cause the algorithm to diverge, oscillating or overshooting the optimal solution, while a low learning rate slows convergence and increases computation time. This trade-off becomes more complex when dealing with large-scale datasets or non-convex optimization problems in deep learning, where the ideal learning rate may change across epochs.

Convergence to Local Minima and Saddle Points

Non-convex loss surfaces in neural networks can have multiple local minima and saddle points where gradients are nearly zero, causing gradient descent to stagnate. Although variants like momentum-based methods and Adam help mitigate these issues, they do not entirely eliminate the risk of getting stuck in suboptimal points. Stochastic methods like SGD also introduce noise, which can aid in escaping local minima but may lead to unstable convergence.

Gradient Vanishing and Exploding

Particularly in deep networks, gradients can either diminish or grow exponentially as they propagate back through layers, a phenomenon known as vanishing or exploding gradients. This can result in poor learning in early layers or numerical instability. Techniques like batch normalization and careful initialization help, but the underlying issue remains a fundamental challenge for gradient-based methods.

Over fitting and Generalization

Gradient descent algorithms optimize the model's performance on the training data, which may lead to over fitting, especially with small datasets or complex models. Regularization techniques, early stopping, and dropout are often required to improve generalization, but they add complexity to the training process and may not always fully address over fitting.

Hyper parameter Tuning

Matrix factorization techniques often involve several hyper parameters, such as the number of latent factors, regularization parameters, and learning rates. Tuning these hyper parameters is critical for achieving optimal performance but can be computationally expensive and time-consuming (5). Poorly chosen hyper parameters can lead to suboptimal model performance, exacerbating issues like over fitting or under fitting.

Computational Cost and Scalability

Variants such as Adam and RMSprop involve additional hyper parameters (e.g., momentum, decay rates), making them more sensitive to hyper parameter tuning than traditional gradient descent. Improper tuning can lead to suboptimal performance or failed convergence. Hyper parameter optimization techniques, like grid search and Bayesian optimization, can help, but they add another layer of complexity to the training process.

Bias in Adaptive Methods

Adaptive methods like AdaGrad, RMSprop, and Adam adjust learning rates based on past gradient information, which can sometimes introduce bias towards specific directions, potentially limiting their performance on certain types of tasks. Some research suggests that adaptive methods may underperform compared to SGD with momentum on simpler, convex problems, leading to slower or suboptimal generalization.

Lack of Robustness in Dynamic Environments

Gradient descent-based methods assume that the data distribution remains constant. In dynamic environments, such as reinforcement learning, where data distributions change over time, these methods can struggle to adapt. They may require regular resetting of accumulated gradient information, adding complexity and potentially slowing adaptation to new data distributions.

Applications of Differential Equations in Modern Neural Networks

Gradient descent and its variants are fundamental to optimizing models across a wide range of machine learning applications. Here are some key areas where these methods are particularly impactful:

Neural Network Training

In deep learning, gradient descent is essential for training neural networks by adjusting weights to minimize the loss function. Variants like SGD, Adam, and RMSprop are commonly used to train models in tasks like image recognition, natural language processing, and speech recognition. For instance, convolutional neural networks (CNNs) for image classification and recurrent neural networks (RNNs) for language modelling rely on efficient gradient-based optimization to achieve high accuracy.

Reinforcement Learning

Gradient-based methods are used in reinforcement learning (RL) to optimize policies in complex, dynamic environments. Policy gradient methods directly optimize policy parameters to maximize cumulative reward, often using SGD or Adam. Applications include game-playing agents (like AlphaGo), robotics, and real-time decision-making in automated systems, where adaptive methods help manage the non-stationary data and reward shifts over time.

Logistic Regression and Linear Regression

In supervised learning, gradient descent is widely used in linear and logistic regression to find the optimal weights that minimize error. These models are applied in fields like healthcare for disease prediction, finance for risk assessment, and marketing for customer segmentation. Batch or mini-batch gradient descent is often sufficient for these simpler models, ensuring efficient and stable convergence.

Collaborative Filtering in Recommender Systems

Gradient descent plays a central role in training matrix factorization models in collaborative filtering. These methods are used in recommender systems, like those for streaming platforms or e-commerce sites, to learn latent factors representing user and item preferences. Gradient-based optimization helps minimize the difference between predicted and actual user ratings, enabling personalized recommendations.

Natural Language Processing (NLP)

Applications in NLP, including machine translation, sentiment analysis, and text generation, rely on deep learning models trained using gradient descent methods. Word embedding’s, such as Word2Vec and GloVe, use SGD to adjust vector representations of words. Models like transformers and BERT for language understanding are optimized using Adam, as it handles the non-convex loss surfaces characteristic of NLP tasks.

III. Conclusion and Recommendations

Gradient descent and its variants are indispensable in machine learning, providing robust frameworks for optimizing complex models across a wide array of applications. While the foundational gradient descent method is straightforward, practical challenges—such as slow convergence, sensitivity to learning rates, and susceptibility to local minima—have driven the development of advanced variants like SGD, Mini-Batch Gradient Descent, Momentum, AdaGrad, RMSprop, and Adam. These variants address specific limitations by introducing mechanisms to stabilize and accelerate convergence, handle sparse data, and adapt to dynamic gradients, making them highly effective in training modern machine learning models, particularly deep neural networks.

References

1. Netrapalli, P. (2019). Stochastic Gradient Descent and Its Variants in Machine Learning. Journal of the Indian Institute of Science, Vol 99, pp. 201–213.
2. Kumar, S., & Gupta, V. (2022). Gradient Descent Optimization Techniques in Deep Learning. Journal of Applied Computing and Machine Learning, Indian Academy of Sciences.
3. Patel, R., & Sharma, T. (2020). Adaptive Gradient Descent Methods for Enhancing Neural Network Training. Advances in Computational Intelligence, Vol 25, pp. 67–75.
4. Rao, N. (2023). Understanding Gradient Descent through Visual and Mathematical Perspectives. International Journal of Research Publication and Reviews, Vol 4, pp. 3458–3466.
5. Reddy, V., & Joshi, M. (2021). Applications of Gradient-Based Algorithms in Natural Language Processing. Journal of Data Science and AI, Indian Institute of Technology.
6. Kulkarni, A., & Singh, H. (2018). Convergence Properties of Gradient Descent in Convex Optimization Problems. Indian Journal of Engineering Mathematics, Vol 12, pp. 120–130.
7. Das, S., & Verma, P. (2019). Enhanced SGD Techniques for Image Classification Models. Journal of Machine Learning Applications, pp. 45–60.
8. Ghosh, R., & Nayak, S. (2021). Gradient Descent Variants for Efficient Model Training in Cloud Environments. Indian Journal of Cloud Computing and AI.
9. Mehta, P. (2022). Mathematical Approaches to Gradient Descent in Large-Scale ML Models. Journal of Advanced Data Science, pp. 213–226.
10. Kaur, B., & Singh, M. (2020). Applications of Gradient Descent in Financial Analytics and Predictive Modeling. Journal of Financial Data Science, Vol 8, pp.98-110.