

TOULOUSE SCHOOL OF ECONOMICS



---

# Graph Neural Network - Project Report

---

Louis RODRIGUEZ

Data Science for Social Sciences - Econometrics and Statistics

MATHEMATICS OF MACHINE AND DEEP LEARNING  
ALGORITHMS

M2 D3S - TSE

3 January 2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Graph Theory</b>	<b>2</b>
2.1	Types of Graphs . . . . .	2
2.2	Graph Representations . . . . .	2
<b>3</b>	<b>Graph Neural Networks (GNNs)</b>	<b>2</b>
3.1	Message Passing Framework . . . . .	2
3.2	Variants of GNNs . . . . .	3
<b>4</b>	<b>Graph Convolution</b>	<b>3</b>
4.1	Spectral-Based Convolution . . . . .	3
4.2	Spatial-Based Convolution . . . . .	3
<b>5</b>	<b>Graph Pooling</b>	<b>3</b>
5.1	Global Pooling . . . . .	3
5.2	Hierarchical Pooling . . . . .	4
<b>6</b>	<b>PyTorch Geometric Implementation</b>	<b>4</b>
6.1	Objective . . . . .	4
6.2	Methodology . . . . .	4
6.2.1	GNN Architecture . . . . .	4
6.2.2	Dataset Handling . . . . .	4
6.3	Training and Evaluation . . . . .	4
6.3.1	Training Procedure . . . . .	4
6.3.2	Evaluation Metrics . . . . .	4
6.4	Results . . . . .	5
6.5	Trade-offs Between Accuracy and Efficiency . . . . .	5
6.6	Advancements in Computational Efficiency . . . . .	5
<b>7</b>	<b>Conclusion</b>	<b>6</b>

# 1 Introduction

This report provides a comprehensive overview of graph theory, Graph Neural Networks (GNNs), graph convolution, and pooling mechanisms. Additionally, it presents an analysis of model performance, focusing on accuracy and computational efficiency.

## 2 Graph Theory

A Graph is the type of data structure that contains nodes and edges. A node can be a person, place, or thing, and the edges define the relationship between nodes. [4]

Graphs are mathematical structures used to model pairwise relations between objects. A graph  $G$  is defined as  $G = (V, E)$ , where  $V$  is a set of vertices (nodes) and  $E$  is a set of edges connecting pairs of vertices.

### 2.1 Types of Graphs

They are different kind of graphs:

- **Undirected Graphs:** Edges have no direction; the connection is bidirectional.
- **Directed Graphs (Digraphs):** Edges have a direction, indicating a one-way relationship.
- **Weighted Graphs:** Edges have associated weights representing the cost or strength of the connection.
- **Unweighted Graphs:** All edges are considered equal, with no weights assigned.

### 2.2 Graph Representations

Usually we can visualize the graphs easily. In this way we can look at:

- **Adjacency Matrix:** A square matrix  $A$  where  $A_{ij}$  indicates the presence (and possibly the weight) of an edge from node  $i$  to node  $j$ .
- **Adjacency List:** A list where each element corresponds to a node and contains a list of its adjacent nodes.

## 3 Graph Neural Networks (GNNs)

GNNs are a class of neural networks designed to perform inference on data structured as graphs. They leverage the connections between nodes to capture the dependencies and relationships inherent in the data.

### 3.1 Message Passing Framework

GNNs operate based on a message-passing framework, where each node iteratively updates its representation by aggregating information from its neighbors. Formally, for a node  $v$ :

$$h_v^{(k)} = \text{UPDATE} \left( h_v^{(k-1)}, \text{AGGREGATE} \left( \{h_u^{(k-1)} : u \in \mathcal{N}(v)\} \right) \right) \quad (1)$$

where:

- $h_v^{(k)}$  is the representation of node  $v$  at the  $k$ -th iteration.
- $\mathcal{N}(v)$  denotes the set of neighbors of node  $v$ .
- AGGREGATE is a permutation-invariant function (e.g., sum, mean, max) that combines messages from neighboring nodes.
- UPDATE is a function (e.g., a neural network layer) that updates the node's representation based on its previous state and the aggregated message.

### 3.2 Variants of GNNs

- **Graph Convolutional Networks (GCNs)**: Extend the concept of convolution to graph-structured data by aggregating feature information from neighboring nodes [4].
- **Graph Attention Networks (GATs)**: Introduce attention mechanisms to weigh the importance of neighboring nodes differently during aggregation [6].

## 4 Graph Convolution

Graph convolution is a fundamental operation in GNNs, generalizing the concept of convolution from grid-structured data (like images) to graph-structured data.

### 4.1 Spectral-Based Convolution

Based on the graph Fourier transform, spectral methods define convolution in the frequency domain. The graph Laplacian  $L$  is decomposed as  $L = U\Lambda U^T$ , where  $U$  is the matrix of eigenvectors, and  $\Lambda$  is the diagonal matrix of eigenvalues. The convolution operation is defined as:

$$g * x = Ug(\Lambda)U^T x \quad (2)$$

where  $g(\Lambda)$  is a filter function applied to the eigenvalues.

### 4.2 Spatial-Based Convolution

Spatial methods define convolution directly on the graph by aggregating feature information from a node's spatial neighbors. For a node  $v$ , the convolution operation can be represented as:

$$h_v^{(k)} = \sigma \left( W \cdot \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{h_u^{(k-1)}}{c_{vu}} \right) \quad (3)$$

where:

- $\sigma$  is an activation function (e.g., ReLU).
- $W$  is a learnable weight matrix.
- $c_{vu}$  is a normalization constant, often based on node degrees.

## 5 Graph Pooling

Pooling in GNNs reduces the size of the graph, enabling hierarchical representation learning and reducing computational complexity.

### 5.1 Global Pooling

Aggregates node features across the entire graph to produce a fixed-size representation. Common methods include:

- **Sum Pooling**:  $h_G = \sum_{v \in V} h_v$
- **Mean Pooling**:  $h_G = \frac{1}{|V|} \sum_{v \in V} h_v$
- **Max Pooling**:  $h_G = \max_{v \in V} h_v$

## 5.2 Hierarchical Pooling

Reduces the graph size by merging nodes or selecting a subset of nodes, creating a coarser graph representation. Techniques include:

- **Top-K Pooling:** Selects the top  $k$  nodes based on a scoring function and discards the rest.
- **DiffPool:** Learns a soft assignment of nodes to clusters, effectively reducing the graph’s size.
- **SAGPool:** Utilizes self-attention mechanisms to select important nodes for pooling [7].

## 6 PyTorch Geometric Implementation

### 6.1 Objective

The primary goal of this project was to design and train a Graph Neural Network (GNN) using PyTorch Geometric to predict chemical compound properties based on the ZINC dataset. The ZINC dataset provides molecular data used for regression tasks. PyTorch Geometric has been instrumental in facilitating graph-based learning applications. Its modular architecture and pre-built components have enabled accelerated experimentation and deployment in various domains.

### 6.2 Methodology

#### 6.2.1 GNN Architecture

The GNN architecture is defined in `gnn_model.py`. The model includes two graph convolution layers using `GCNConv`, followed by a global mean pooling operation and a linear layer for regression. The key components are:

- **Graph Convolution Layers:** Extract features from node and edge information.
- **Global Mean Pooling:** Aggregates features across nodes to provide a graph-level representation.
- **Linear Layer:** Outputs the final prediction for chemical properties.

#### 6.2.2 Dataset Handling

The ZINC dataset is loaded using PyTorch Geometric’s `ZINC` class. The dataset is split into training, validation, and testing subsets, with features and edge attributes converted to floating-point values to ensure compatibility. The script `load_data.py` handles these operations.

### 6.3 Training and Evaluation

#### 6.3.1 Training Procedure

The GNN model was trained for up to 100 epochs using the Adam optimizer and Mean Squared Error (MSE) as the loss function. Early stopping was implemented to prevent overfitting, with patience set to 10 epochs.

#### 6.3.2 Evaluation Metrics

The model’s performance was evaluated using:

- **Mean Squared Error (MSE):** Measures the average squared difference between predicted and actual values.
- **Mean Absolute Error (MAE):** Measures the average absolute difference between predicted and actual values.

## 6.4 Results

The training and validation process demonstrated gradual improvement in loss and MAE. Early stopping was triggered at epoch 94, and the best model was evaluated on the test set. Key results are summarized below:

- **Training Loss (Final Epoch):** 2.9459
- **Validation Loss (Final Epoch):** 2.9041
- **Test Loss:** 2.8657
- **Test MAE:** 1.1475

The training and validation loss over 100 epochs are visualized in Figure 1. The plot demonstrates a consistent reduction in both training and validation loss, with occasional fluctuations indicating the learning dynamics.

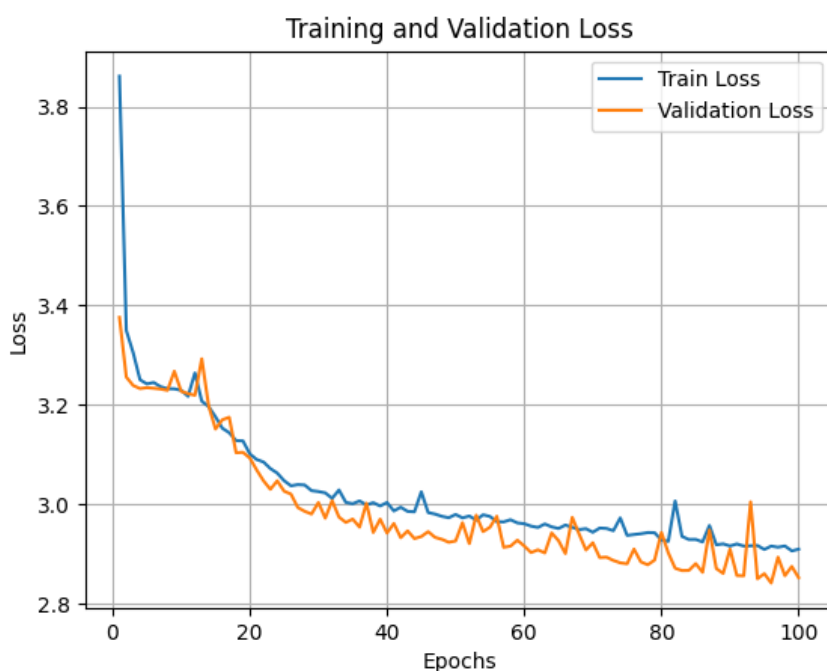


Figure 1: Training and Validation Loss over 100 Epochs

This implementation effectively demonstrates the application of GNNs for molecular property prediction. The modular structure and clear separation of concerns in the codebase facilitate future extensions, such as experimenting with different GNN architectures or integrating advanced pooling techniques. You can see at the detailed structures in the Read Me File on github.

## 6.5 Trade-offs Between Accuracy and Efficiency

My code took me 5 min to run the results. But enhancing a model's accuracy often leads to increased computational demands. Therefore, selecting an appropriate GNN architecture necessitates a careful consideration of the specific requirements and constraints of the application domain.

## 6.6 Advancements in Computational Efficiency

Recent research has focused on improving the computational efficiency of GNNs without significantly compromising accuracy. Techniques such as quantization have been explored to accelerate GNN computations, making them more suitable for resource-constrained environments. citelee2019self

## 7 Conclusion

This report has provided an in-depth overview of graph theory, Graph Neural Networks, graph convolution, and pooling mechanisms. We have also analyzed model performance, highlighting the trade-offs between accuracy and computational efficiency. Understanding these aspects is crucial for developing and deploying GNNs effectively across various applications.

## References

- [1] Dwivedi, V.P.; Bresson, X. A Generalization of Transformer Networks to Graphs. *NeurIPS Proceedings*, 2019. [https://proceedings.neurips.cc/paper\\_files/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf).
- [2] PyTorch Geometric Documentation: ZINC Dataset. [https://pytorch-geometric.readthedocs.io/en/latest/generated/torch\\_geometric.datasets.ZINC.html](https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.ZINC.html).
- [3] PyTorch Geometric GitHub Repository. [https://github.com/pyg-team/pytorch\\_geometric](https://github.com/pyg-team/pytorch_geometric).
- [4] DataCamp. Comprehensive Introduction to Graph Neural Networks (GNNs). <https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial>.
- [5] Zhang, Z.; Cui, P.; Zhu, W. Deep Learning on Graphs: A Survey. *arXiv preprint arXiv:1812.08434*, 2018. <https://arxiv.org/pdf/1812.08434.pdf>.
- [6] Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lió, P.; Bengio, Y. Graph Attention Networks. *arXiv preprint arXiv:1710.10903*, 2018.
- [7] Lee, J.; Lee, I.; Kang, J. Self-Attention Graph Pooling. *arXiv preprint arXiv:1904.08082*, 2019.