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

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

Graph theory is the study of graphs, which 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

- **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

- **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:

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

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 [1].
- **Graph Attention Networks (GATs)**: Introduce attention mechanisms to weigh the importance of neighboring nodes differently during aggregation [2].
- **Graph Isomorphism Networks (GINs)**: Designed to be as powerful as the Weisfeiler-Lehman graph isomorphism test, allowing them to distinguish a broader class of graph structures [3].

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 Λ is the diagonal matrix of eigenvalues. The convolution operation is defined as:

$$g * x = Ug(\Lambda)U^T x \tag{2}$$

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:

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

where:

- σ 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**: $h_G :: contentReference[oaicite:0]index = 0$

$= \sum_{v \in V} h_v$

- **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 [4].
- **DiffPool**: Learns a soft assignment of nodes to clusters, effectively reducing the graph's size [5].
- **SAGPool**: Utilizes self-attention mechanisms to select important nodes for pooling [6].

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.

6.2 Methodology

6.2.1 Project Structure

The implementation is organized into modular components, ensuring scalability and maintainability. The directory structure is as follows:

```
project/

 models/
    gnn_model.py        # Defines the GNN architecture.

 data/
    load_data.py        # Utility script to load and split the ZINC dataset.

 utils/
    metrics.py          # Evaluation metrics, including Mean Absolute Error.

 main.py                 # Main script to train and evaluate the model.
 README.md               # Project documentation and instructions.
 requirements.txt        # Dependencies for the project.
```

6.2.2 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.

6.2.3 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

6.5 Challenges and Observations

- The initial implementation faced issues with node and edge feature compatibility, which were resolved by ensuring all features were of type `float`.
- A warning regarding missing dependencies (e.g., `torch-scatter`, `torch-sparse`) was noted, but it did not impact the results significantly.

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.

6.6 Performance of Various GNN Models

Different GNN architectures offer varying balances between accuracy and computational efficiency:

- **Graph Convolutional Networks (GCNs)**: Provide a good balance between performance and computational cost but may struggle with capturing complex dependencies in large graphs [1].
- **Graph Attention Networks (GATs)**: Introduce attention mechanisms that can improve accuracy by assigning different importances to neighboring nodes, though at the expense of increased computational complexity [2].
- **Graph Isomorphism Networks (GINs)**: Designed to be more expressive in distinguishing graph structures, potentially leading to higher accuracy, but may require more computational resources [3].

6.7 Trade-offs Between Accuracy and Efficiency

Enhancing a model's accuracy often leads to increased computational demands. For instance, incorporating attention mechanisms in GATs can improve performance but also raises the computational burden. Therefore, selecting an appropriate GNN architecture necessitates a careful consideration of the specific requirements and constraints of the application domain.

6.8 Advancements in Computational Efficiency

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

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] Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, 2017.

[2] Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. *arXiv preprint arXiv:1710.10903*, 2018.

[3] Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? *arXiv preprint arXiv:1810.00826*, 2019.

[4] Gao, H.; Ji, S. Graph U-Nets. *arXiv preprint arXiv:1905.05178*, 2019.

[5] Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. *arXiv preprint arXiv:1806.08804*, 2018.

[6] Lee, J.; Lee, I.; Kang, J. Self-Attention Graph Pooling. *arXiv preprint arXiv:1904.08082*, 2019.