# DSC Capstone Section B12: Quarter 1 Project

**Sheena Patel**
s4patel@ucsd.edu

**Gal Mishne, Yusu Wang**
gmishne@ucsd.edu, yusuwang@ucsd.edu

## Abstract

I will begin this paper with the simplest breakdown of the features of a neural network. A **neural network** is a method used in artificial intelligence that teaches computers to process data in the same way that the human brain processes data. Neural networks consist of layers of connected neurons and begin with a mathematical function that takes in an input, processes it, and produces an output that could be fed into another layer depending on the complexity of the model. The final layer is known as the output layer and produces an output based on the prediction task given. Many refer to the use of neural networks as deep learning. Graph learning models or **Graph Neural Networks** (GNN's) are a particle type of neural network that utilizes nodes and edges to detect relationships in graph-structured data. Graph-structured data can come in various sizes and orders. Examples of this type of data include social networks, transportation systems, and molecular structures. GNN's consists of node representations, edge representations, message passing capabilities, aggregation functions, convolutional layers, and graph pooling to aggregate information from layer to layer. We use different GNN architectures and frameworks such as Graph Convolutional Networks, Message-Passing GNN's, and Transformer-Based models, in order to create sharable parameters and effectively train and process the graph data. GNN's excel in relationship detection and are commonly used for graph analysis, node classification, and link prediction. However, when these models are fed deep multi-layered graphs they are often limited when capturing long-range interactions between nodes since distance is hard to uniquely label between undirected node-links. As a result of this limitation and the increasing popularity of graph analysis, capstone section B12 explores ways to capture **long-range interactions in GNN architectures**.

Code: https://github.com/sheenapatel262/dsc_captsoneB12_project_1

# 1 Introduction

*This paper will begin with an overview of the basic GNN architectures I've studied this quarter and then conduct an in-depth exploration of 2 specific GNN's and their model coding implementations in the Method section of the paper. The results will provide a baseline comparison of model performances, the discussion will provide insight on limitations and further research, and the conslusion will summarize overall findings of this Q1 capstone.*

Some of the commonly known GCN models we've studied this quarter are Graph Convolutional Networks (GCNs), Graph Isomorphism Network (GIN), Graph Attention Network V2 (GATv2), and graph transformer models (GraphGPS).

## 1.1 GNN Architectures

### 1.1.1 GCN

**Graph Convolutional Networks (GCNs)** are spectral-based as they use spectral decomposition of the graph **Laplacian matrix** to capture global information in graph-structured data. They take in graph-data as input and perform a layer-wide propagation on it through **eigendecomposition**. The GCN learns node representations by aggregating information from neighboring nodes and computing a weighted sum from their respective features. Through the propagation process, the model learns the structure of the graph as well as relations that are present. However, when the network is deep and multiple layers are present, the model can exhibit oversmoothing which causes nodes to be processed too similarly to distinguish from one another, leading to divergent results. Despite their difficulty to capture large-scale graph interactions they can still be applied to social network analysis, recommendation systems, and biology, where data is inherently organized as graphs.

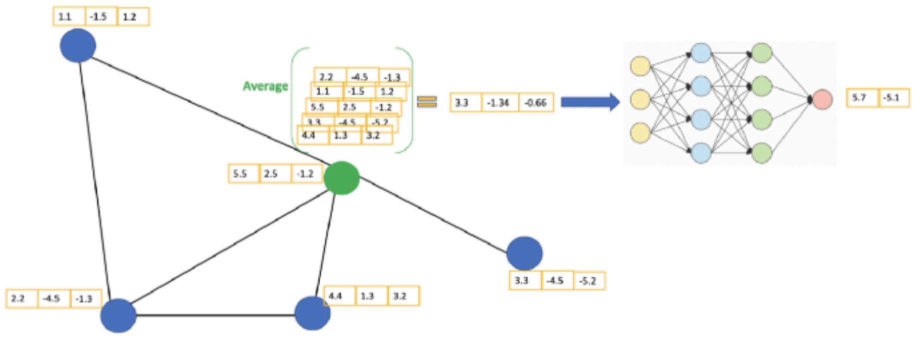


Figure 1: This figure illustrates a simplified GCN representation through the aggregation of the green node into a 2-dimensional vector output.

### 1.1.2 GIN

**Graph Isomorphism Networks** (GINs) are a type of message-passing GNN that is **permutation invariant** which means it doesn't rely on graph structure, making it more flexible and applicable to a variety of graph types. The GIN uses a layer-wise operation inspired by the **Weisfeiler-Lehmen Graph Isomorphism Test**. The layer-operation in GIN updates nodes by aggregating information throughout the graph and then applies a **multi-layer perceptron** (MLP) or type of neural network to all aggregations which causes it to be invariant to node orderings. After the model aggregates all information across nodes to produce a graph-level representation, it is commonly used for graph classification. Due to the structure of the model, GIN's typically perform better on classification and regression tasks.

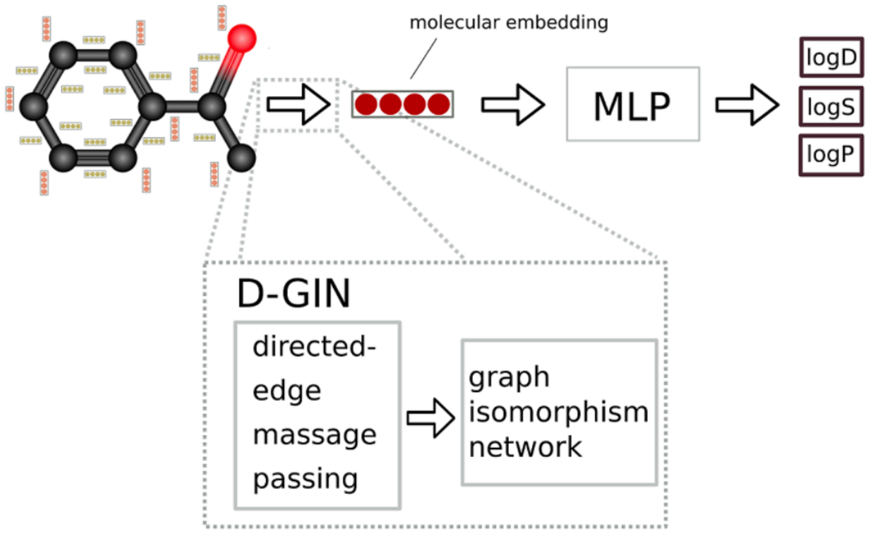


Figure 2: An overview of the way that GIN sums all node embeddings to ignore direction/ structure/relations and then applies further operations (MLP in this case) to prodouce this classification prediction

### 1.1.3 GAT

**The Graph Attention Network** (GATv2) is another type of message passing GNN which incorporates the weights of nodes to the model through attention mechanisms which allow capturing complex dependencies and importance of certain nodes throughout the data. The model performs an attention mechanism using attention heads which are pathways which learn different relationships throughout the model from the various feature values provided. Multiple **attention heads** are used to capture the various relationships and independently compute weights. The model then computes the similarity of features and assigns nodes with more relevant information to higher attention weights. The model then aggregates all attention-weight features to focus on the more informative neighbors during the information propagation. All weights from the heads are then averaged and normalized using the softmax function which ensures that the weights are positive and sum to 1 for a valid probability distribution. The final output is passed through a nonlinear activation function and used for node or graph classification. Given the complexity of the heads, similarity computations, comparisons, and final aggregations of the model, it is definitely more computationally demanding than the previous models explained.

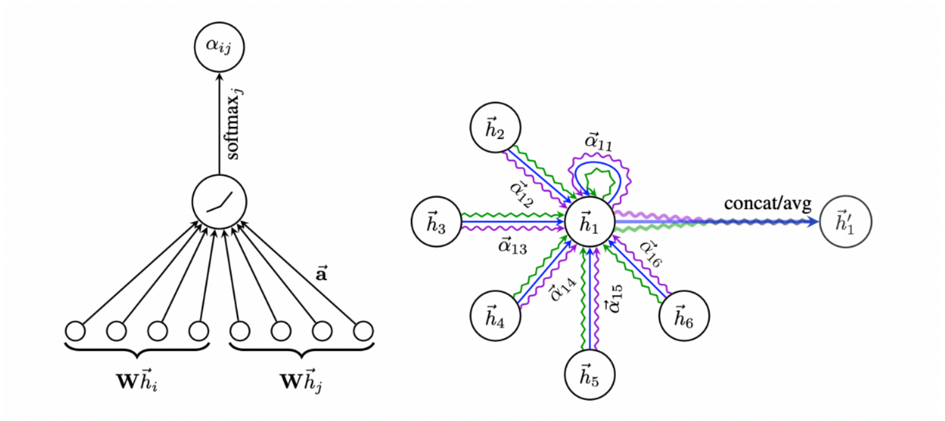


Figure 3: This figure illustrates how a GAT extracts different weights to an output prediction with 3 heads applied

### 1.1.4 Graph GPS

The **GraphGps** is a **transformer-based** GNN. A Graph Transformer is a neural network architecture that computes attention scores based on graph structure to consider the connectivity as well as incorporating positional encodings while capturing spatial relationships. The use of both operations allows Graph transformers to be a specialized GNN that can capture complex relationships and dependencies.

## 1.2 Issues with Capturing Long-Range Dependencies

### 1.2.1 Oversquashing

The phenomenon of **oversquashing** in GNNs is when there are issues passing messages and information between distant nodes in the graph so data is embedded into a fixed length vector. A **bottleneck** in GNNs is when the number of layers increases which causes the number of nodes to increase in a field exponentially. This leads to long-range information failure. Oversquashing is definitely a phenomenon that is more prevalent in GCN and Graph Isomorphism Network (GIN) because they do not incorporate attentional mechanisms that allow more weight to informative connections like GNN's such as Graph Attention Network (GAT) . To improve the problem we can add additional layers to allow nodes yo to access information from a broader range of neighbor, use attention mechanism to add more weight to relevant neighbors and vise-versa, and apply node aggregations to retain the important information in distant nodes so it isn't "squashed."

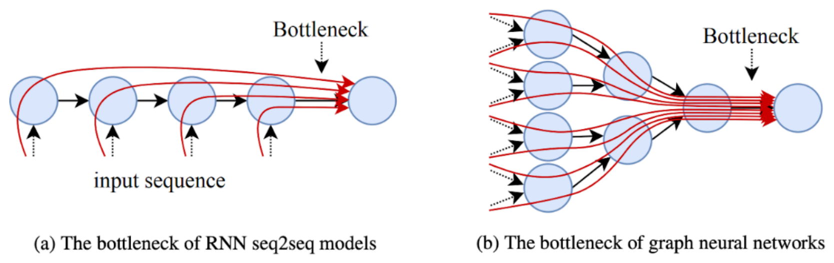

Bottleneck
input sequence
(a) The bottleneck of RNN seq2seq models
Bottleneck
(b) The bottleneck of graph neural networks

Figure 4: A representation of the phenomena of oversquashing converging long-range node dependencies

### 1.2.2 Oversmoothing

**Oversmoothing** occurs when the depth of GNN's leads to all the nodes having similar embeddings, making it challenging to distinguish between nodes and different characteristics. This occurs because as depth increases, nodes access information from distant neighbors which make them similar. Typically we can use **Mean Average Distance** (MAD) which measures the average distance between nodes representation or embedding in a graph or MADGap which is an extension of MAD that measures the MAD between does of different classes and nodes of the same classes and compares them to determine whether oversmoothing is affecting the models results. The most straight-foward solution to this issue is layer reduction; however, it prevents the model from taking in complex graphs. Regularization and pooling to encourage diversity in node embeddings and a focus on the most important nodes and subgraphs. **Differential Group Normalization** (DGN) can also be used to prevent similar embedding due to long-range graphs as DGN assigns nodes in groups and normalizes them independently to create a distinction between them.

## 2 Methods

### 2.1 Dataset Analysis

*Throughout quarter 1 of this capstone, we used the Cora dataset, Enzymes dataset, IMDB-Binary dataset, and finally the Long Range Graph Benchmark dataset provided by Pytorch Geometric. We applied and compared accuracies of the following architectures: GCN, GIN, GAT, GPS*

**Pytorch Geometric** is an extension library for PyTorch designed specifically for deep learning on graph-structured data. The library provides GNN layers, graph data handling, and a collection of benchmark datasets that we used to test our archetecture implementations.

### 2.1.1 Cora Dataset

The **Cora dataset** calls for a method of node classification in order to predict the class of a scientific paper given citation relationships between papers. Nodes in the graph represent different scientific papers, edges represent relationships from paper to paper, features typically are bag-of-word representations of paper content, and node labels are predefined research topic categorizations. There is 1 graph, 2708 nodes, 7 classes, and 1433 node features in this dataset.

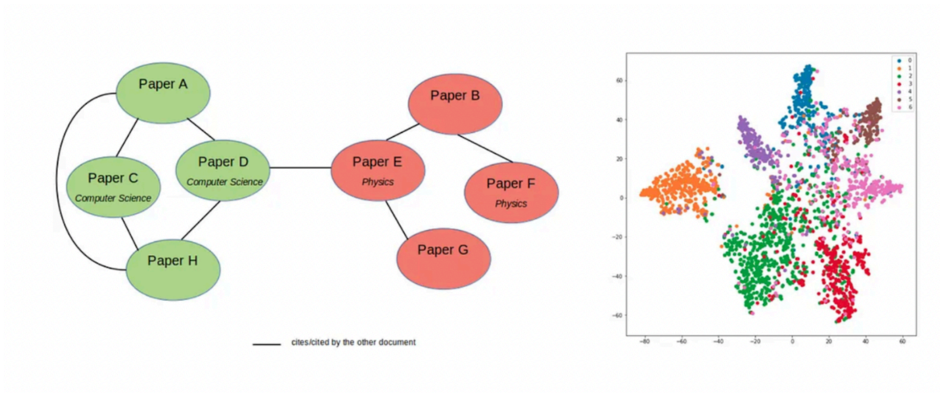


Figure 5: The structure and classification of the Cora Dataset.

### 2.1.2 Enzymes Dataset

The **Enzymes dataset** calls for a method of graph classification in order to assign enzymes to one of 6 labels of classes that represent a catalyzed chemical reaction. Each node in the dataset represents a secondary structure of elements and each edge connects two nodes if they are neighbors along the amino acid sequences.

### 2.1.3 IMDB-Binary Dataset

The **IMDB-binary dataset** calls for graph classification to identify movie genres. The dataset contains actors and actresses as nodes and edges between them if they appear in the same movie.

### 2.1.4 Pascal Dataset

The **Pascal dataset** is derived from the Long Range Benchmark (LRGB) datasets which are a collection of 5 graph learning datasets with tasks based on their long-range dependencies. The PascalVOC-SP (Pascal Visual Object Classes- Superpixel ) falls into the computer vision domain and presents a node classification task. The dataset contains 20 object categories such as vehicles, dogs, chairs, etc. with each image containing a pixel-level segmentation.

Each node will represent a region of the image belonging to a particular class and we will predict the class for each image (region).

## 2.2 Implementation of Message Passing

### 2.2.1 GCN Message Passing

The **GCN model** I created for node classification of the Cora Dataset consists of 2 graph convolution layers. The first layer `self.conv1` takes as input the number of features in the dataset and outputs the hidden channels instantiated for the particular network. The Graph Convolutional Network (GCN) updates are done through a series of mathematical operations involving graph convolution layers.

The final output of a GCN layer is typically the concatenation or aggregation of the updated representations of all nodes. The process is then repeated for multiple layers to capture information from distant nodes in the graph.

It's important to note that variations of the formula may exist depending on the specific implementation, and additional considerations such as dropout, activation functions, and normalization methods might be included in practice. The choice of these components can impact the performance of the GCN model on specific tasks and datasets.

The second layer `self.conv2` takes the number of hidden layers as input and outputs the number of classes in the cora dataset. The forward pass takes in the input features x and the graph connectivity `edge index` which are processed through the first convolutional layer. In this layer, each node aggregates information from its neighbors based on the graph structure given by the `edge index`. After the convolution, a Rectified Linear Unit Activation function is applied to introduce non-linearity by applying a simple threshold to the input.

Next a dropout is applied to the x values for regularization. The dropout randomly zeros out a fraction (probability) in the neurons in the layer during the training in order to prevent overfitting and improve generalization. Finally, the activated x is passed through the second convolutional layer to produce an output of the model's output. The model's GCNConv layer offered by PyTorch Geometric encapsulates the graph convolution operation, facilitating the efficient propagation of information through the graph structure.This design adheres to the fundamental concept of message passing in graph neural networks, where nodes aggregate information from their neighbors to update their own representations.After creating the actual GCN function, I initialized it with 20 hidden channels. Initially, I started off with 16 because it was default but experimented and found that 20 performed better on average. Then I set up an Adam optimizer with a learning rate of 0.005 and weight decay of .0005. I also choose to use the Cross Entropy Loss function because it is most suitable for classification tasks. Next, I iterated the model through 50 epochs which each epoch zeroing out the gradient, computing the model output, calculating the loss, computing the gradient of the loss, and then updating the model parameters using the Adam optimizer (optimizer.step()) in the direction that reduces loss. Lastly, I evaluated the model on the test set to obtain an accuracy that ranged from 78-80. This accuracy makes sense since GCN's are known for better performance with node classification tasks as they are able to capture localized and neighborhood information for their predictions.

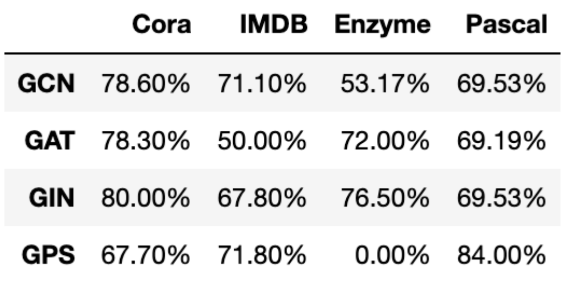
#### 2.2.2 Graph GPS Message Passing

The transformer-based model I implemented was GPSConvNet. This Graph GPS is initialized with `in channels`, `hidden channels`, `out channels`, `heads`, `dropout p`, and an activation function (default `act = relu`). $Relu(x) = max(0, x)$

The code begins by preprocessing the input data by sequentially applying a linear layer that inputs `in channels` and outputs 2*h (number of hidden channels) and then applies a Gaussian Error Linear Unit activation function which is non-linear. This is implemented twice to preprocess the data. The model then loops over all `hidden channels` to create various GPSConv layers that consist of both a GATv2Conv layer to allow for attention mechanisms in the module and a GPSConv layer. The model's final layer is a GATv2Conv layer that inputs all previous layer data and outputs the number of classes. In the forward method of the model, the data features (x) are preprocessed. Next, the preprocessed features are fed into each GPSConv layer initiated and a Relu activation function and dropout is applied to each output. The final x is passed through the final convolutional layer and the log softmax function is applied to it. The next part of the model uses Adam's optimizer to update the model parameters, compute the negative log-likelihood loss `F.nll loss`, and backpropagate the gradients to update the model parameters.

While the output varied with each run, I found that increasing the epochs eventually begins to increase the loss. The average accuracy returned ranged from 70%-78%. The accuracy remained consistent with the other model accuracies as transformers are used to improve long-range interactions and the cora dataset is not as complex to see a large difference.

## 3 Results



|  | Cora | IMDB | Enzyme | Pascal |
|---|---|---|---|---|
| GCN | 78.60% | 71.10% | 53.17% | 69.53% |
| GAT | 78.30% | 50.00% | 72.00% | 69.19% |
| GIN | 80.00% | 67.80% | 76.50% | 69.53% |
| GPS | 67.70% | 71.80% | 0.00% | 84.00% |

Figure 6: A performance chart of model vs. dataset type

The GCN model implemented had a consistent performance with slight differences based on run due to the variation of test and train masks used on each run. This demonstrates the model's robust behavior. As it achieves relatively high accuracy (ranging from 55%-80%) on all models.

The GAT model achieved the overall highest performance as it considered the graph structure and relations applying weight to nodes with similar features. The model performed the lowest on the IMDB-Binary dataset which suggests that the task did not emphasize graph structure compared to the others such as Cora which performed the highest and makes sense as papers with similar content would have higher weights and improve predictions.

The GIN model performed the second best overall as it aggregates nodes but disregarded graph structure. The Cora dataset performed very well while the IMDB-Binary dataset had a lower accuracy.

Lastly, the GPS model was definitely the hardest to implement so we did face some obstacles. However, we used a GitHub source code to replicate and optimize. Overall, the model performed the lowest due to the lack of longe-range datasets in our experiment pool. Our implementation for Enzymes received a 0% accuracy and the model had a runtime error on the PASCAL dataset due to its depth. As a filler, we estimated the average accuracy performed on that dataset on GraphGPS posted by various sources online.

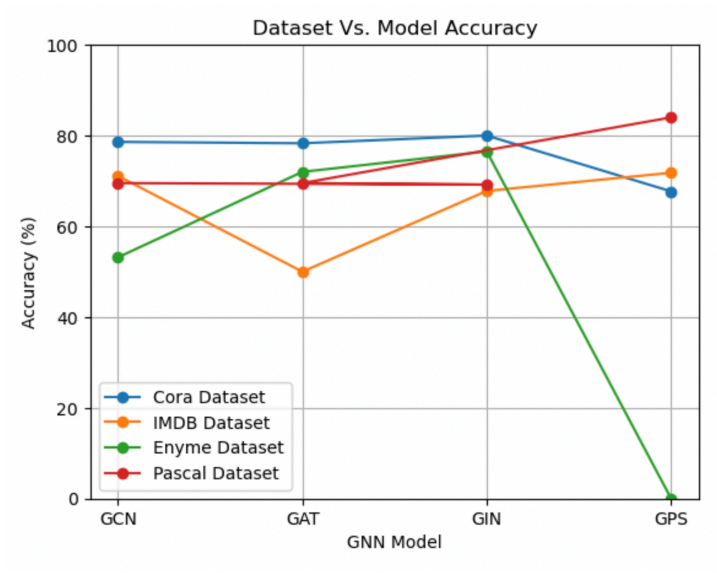


Figure 7: A line graph comparing model performance and results.

## 4 Discussion, Limitations, Future Work

Due to the limited time frame of this project. I was not able to experiment with the parameters enough to effectively optimize all models so further exploration is definitely required for enhanced performance. Moreover, I did face runtime errors in the GPSConv model, particularly with the Pascal dataset, necessitating future troubleshooting and experimentation with CPU usage. I was able to find historcial performances on the dataset online to fill in accuracy until achieved individually.

## 5 Conclusion

Overall, the results align with existing research, emphasizing the strengths and limitations of each GNN model. The robustness of GCN, the graph structure sensitivity of GAT, the versatility of GIN, and the challenges faced by GPSConv in long-range datasets are highlighted. Insights gained from these findings underscore the importance of selecting GNN models based on dataset characteristics and specific task requirements.

## 6 References

**Alon, Uri, and Eran Yahav.** "On the Bottleneck of Graph Neural Networks and Its Practical Implications." ArXiv.org, 9 Mar. 2021, [Link].

**Oono, Kenta, and Taiji Suzuki.** "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification." ArXiv.org, 6 Jan. 2021, arxiv.org/abs/1905.10947. Accessed 12 Dec. 2023, [Link].

**Rampášek, Ladislav,** et al. "Recipe for a General, Powerful, Scalable Graph Transformer." ArXiv.org, 15 Jan. 2023, arxiv.org/abs/2205.12454. Accessed 12 Dec. 2023, [Link].

**Veličković, Petar,** et al. "Graph Attention Networks." ArXiv:1710.10903 [Cs, Stat], 4 Feb. 2018, [Link].

**Xu, Keyulu,** et al. "How Powerful Are Graph Neural Networks?" ArXiv:1810.00826 [Cs, Stat], 22 Feb. 201, [Link].