Graph Neural Network

CS 747 Project Report

Balassubramanian Srinivasan and Nicholas Brennan-Martin

Abstract

In this paper, we delve into the fascinating concept of Graphical Neural Networks (GNNs) and their potential in various real-world applications. We begin by providing an overview of the fundamental principles of GNNs, including their use of graphs to model data and their ability to incorporate complex relationships between nodes. We then explore three popular types of GNNs - Graph Convolutional Neural Network, Hypergraph Neural Network and GraphSAGE - and discuss their unique features.

Next, we present some examples of how GNNs have been applied in real-world scenarios, including natural language processing and computer vision. Specifically, we showcase our own experiments on both a text dataset and an image graph dataset, demonstrating the capabilities of GNNs in classification tasks. We noticed that the GNNs though can be used to achieve high accuracy and can be extended to larger datasets it requires a lot of computational time and memory.

Overall, our paper highlights the exciting potential of GNNs as a powerful tool for modeling complex data relationships and solving challenging machine learning problems

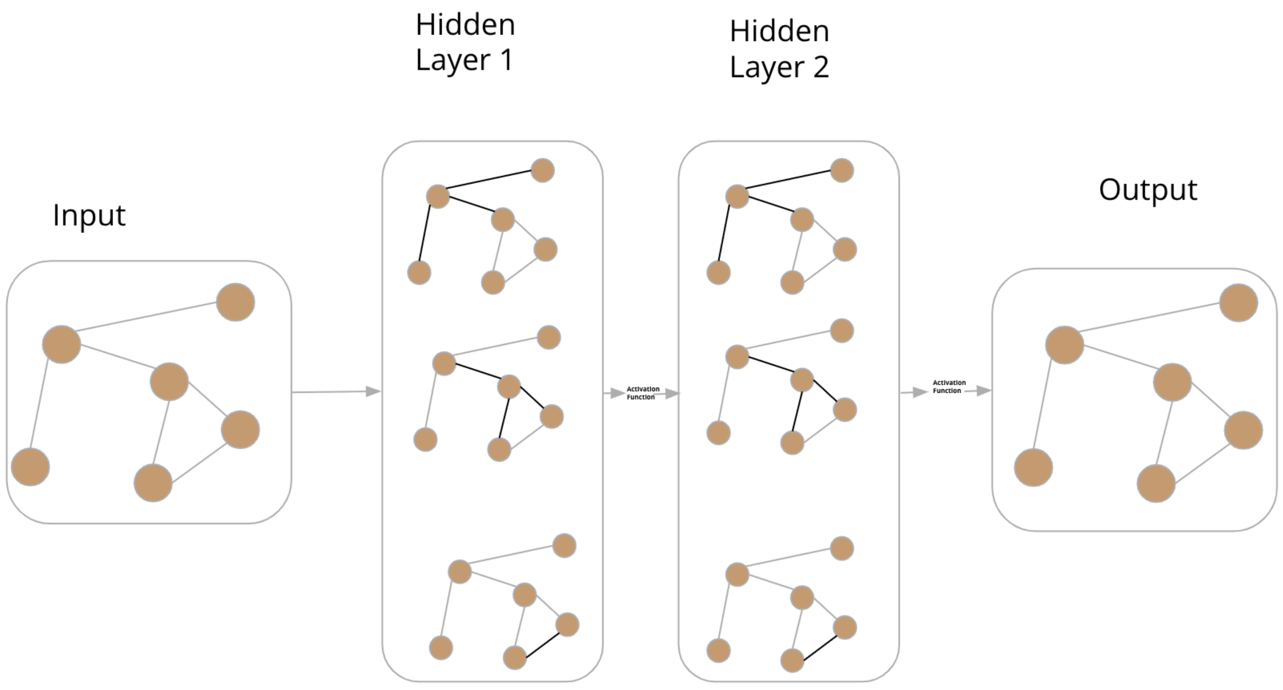


Figure 1: Graph Neural Network Overview

CS 747 Project Report

Balassubramanian Srinivasan and Nicholas Brennan-Martin

May 2023

1 GNN Introduction/Background

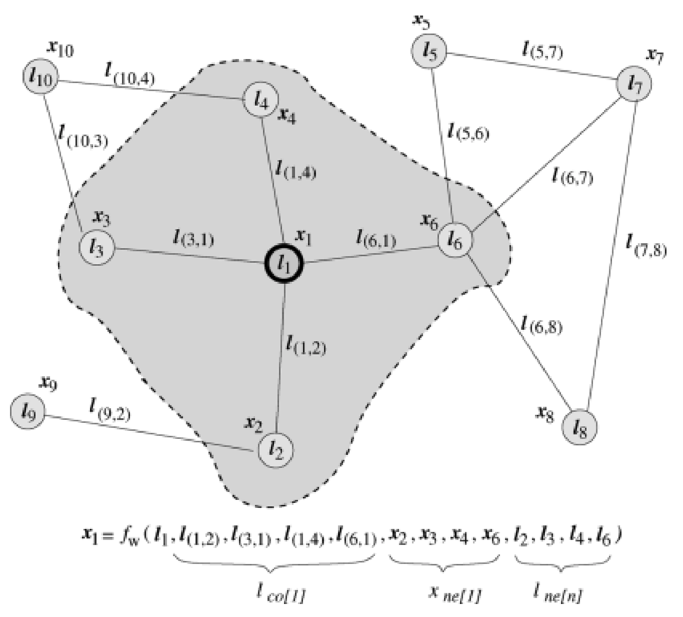


Figure 2: : Graph and the neighborhood of a node.[1]

Graphs are widely used to represent data in various scientific and engineering fields such as molecular biology, pattern recognition, and data mining. Graphical Neural Networks (GNNs) [1] are a type of deep learning model that combines features of both recursive neural networks and random walk models while retaining their characteristics. GNNs are capable of processing a wide range of graph types including: acyclic, cyclic, directed, and undirected graphs, without the need for preprocessing. They also introduce a learning algorithm, expanding the class of processes that can be modeled compared to traditional random walk theory.

GNNs are able to encode the structural information of the graph into the neural network architecture, allowing for the propagation of information between nodes in a graph.

Traditional neural networks operate in Euclidean spaces, where data points are represented as vectors. In contrast, graphs are non-Euclidean data structures, where entities are represented as nodes and their relationships are represented as edges. GNNs extend the traditional neural network architecture to operate on graphs, enabling them to capture the structural information of the graph and make predictions based on that information.

The basic idea of GNNs is to perform message passing between nodes, where each node aggregates information from its neighbors and updates its own representation. This process is repeated iteratively until convergence or for a fixed number of iterations.

The model of GNN can be implemented with the idea that nodes represent objects or concepts and edges represent relationships between objects. Each concept is naturally defined by its features and related concepts. The model attaches a state $x_n \in R^s$ that contains a representation of the concept to each node n. Two parametric functions f_w *and* g_w called the local transition function and local output function or the aggregation function are defined as, a function that expresses the dependence of a node on its neighborhood and a function that describes how the output is produced, respectively. The state and output can be defined as:

$$x_n(t+1) = f_w(l_n, l_{CO[n]}, x_{ne[n]}, l_{ne[n]}) \tag{1}$$

$$o_n(t) = g_w(x_n, l_n) \tag{2}$$

where $l_n, l_{CO[n]}, x_{ne[n]}, l_{ne[n]}$ are the features of the current node n, labels of its edges, the states of neighbouring nodes, and the labels of the nodes in the neighborhood of, respectively. The Learning Framework of GNN can be defined as

$$L = \{(G_i, n_{i,j}, t_{i,j}) | G_i = (N_i, E_i) \in G; n_{i,j} \in N_i; t_{i,j} \in R^m, 1 \le i \le p, \le j \le q_i\} \tag{3}$$

where q_i is the number of supervised nodes in G_i.

GNNs can be trained in a supervised or unsupervised manner, depending on the task. Supervised learning involves training the GNN to predict the labels of the nodes or edges in the graph, given their features and the graph structure. Unsupervised learning involves training the GNN to learn the latent representations of the nodes or edges in the graph, without any explicit labels.

GNNs are typically trained using backpropagation over time, which involves computing the gradients of the loss function with respect to the GNN parameters at each iteration of the message passing. The gradients are then used to update the GNN parameters using stochastic gradient descent or one of its variants.

GNNs have shown state-of-the-art performance on a range of tasks, including node classification, link prediction, and graph classification, among others.

2 Types Of Graphical Neural Network

2.1 Graph Convolutional Neural Network

2.1.1 Introduction and Learning in Graph Convolutional Neural Network

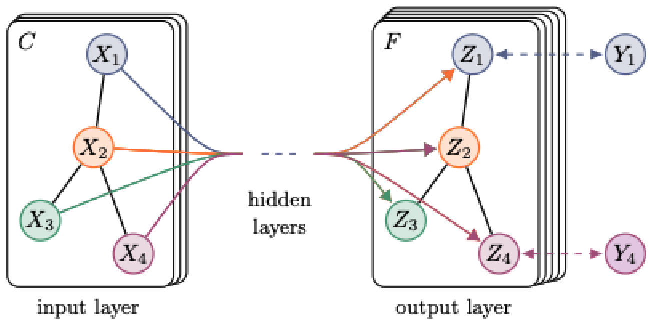


Figure 3: Graph Convolutional Neural Network [2]

Graph Convolutional Neural Network [GCN] [2] is a variant of convolutional neural network that can directly operate on graphs. GNNs operate by propagating information across the graph structure, allowing them to capture the local connectivity patterns and the global structure of the graph simultaneously. This is achieved by defining a set of learnable functions that operate on the graph structure, such as graph convolutions or message passing functions.

$$H^{l+1} = \sigma(\tilde{D}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{4}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections, I_N is the identity matrix, $\tilde{D}_{ii} = \Sigma_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix and σ denotes an activation function.$H^{(l)} \in R^{N \times D}$ is the matrix of activation in the l^{th} layer; $H^{(0)} = X$.

A signal $X \in R^{N \times C}$ with C input channels and F filters can be define as :

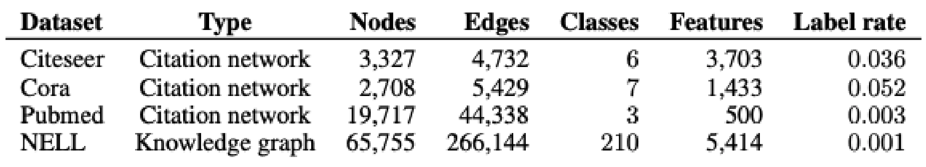
$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta \tag{5}$$

where $\Theta \in R^{C \times F}$ is matrix of filter parameters and $Z \in R^{N \times F}$ is the convolved signal matrix.

2.1.2 Experiment and Results

The paper[2] conducted an experiment that tested a Graph Convolutional Neural Network model on semi-supervised node classification tasks using four datasets. These datasets included three citation networks (Citeseer, Cora, and Pubmed) and one knowledge graph (NELL). The citation networks contained documents as nodes and citation links as edges, while the NELL dataset was a bipartite graph.

In the citation networks, citation links were treated as undirected edges, and a binary, symmetric adjacency matrix A was constructed.



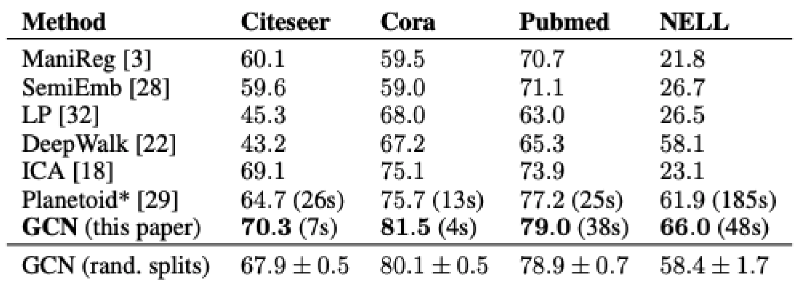
Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Figure 4: Dataset and features [?]

The experiment involved training a two-layer GCN model and evaluating its prediction accuracy on a test set of 1,000 labeled examples. The GCN model was compared against several other models, and it was found that for all the datasets, GCN outperformed the other models.

Overall, the results of the experiment demonstrated the effectiveness of GCN in semi-supervised node classification tasks, particularly on graph-structured data such as citation networks and knowledge graphs.

GCN are a suitable model for simple graph datasets but have limitations. They are restricted to undirected graphs and require significant memory resources, which makes them challenging to scale up. Additionally, it is difficult to extend GCNs to multimodal datasets that contain different types of nodes and edges, which limits their applicability in more complex graph structures.



Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Figure 5: Accuracy Scores [2]

2.2 Hypergraph Neural Network

2.2.1 Introduction

GNN relies on pairwise connections among data. The data structure in real practice could be far more complicated. Data like social media data are multi modal. The traditional graph structure may not be fully effective in capturing complex data correlations, thereby limiting the suitability of GNN for such scenarios. Hypergraph Neural Network [HGNN][3] uses a hypergraph structure to model complex and high-order data correlations, which allows for encoding multi-modal data and capturing complicated relationships beyond pairwise connections. Compared to a standard graph where all edges are required to have a degree of two, a hypergraph has the unique capability to represent high-order data correlations that go beyond pairwise connections, thanks to its degree-free hyperedges.

2.2.2 Learning in Hypergraph Neural Network

The hypergraph G can be denoted by a $|V| \times |E|$ incidence matrix H, where the values are based on :

$$h = \begin{cases} 1, & \text{if } v \in e \\ 0, & \text{if } v \notin e \end{cases} \tag{6}$$

For a vertex v $\in$ V, its degree is defined as $d(v) = \Sigma_{e \in E} \omega(e) h(v, e)$. For an edge e, its degree is defined as $\delta(e) = \Sigma_{v \in V} h(v, e)$. Learning task for node classification can be formulated as

$$arg \min_{f} R_{emp}(f) + \Omega(f) \tag{7}$$

where Ω(f) is a regulariser on hypergraph, $R_{emp}(f)$ denotes the supervised empirical loss and f($\cdot$) is a classification function. $\Omega(f) = f^T \Delta$ and $\Delta = I - \Theta$, Δ is positive semi-definite, and called the hypergraph Laplacian.

2.2.3 Analysis of Hypergraph Neural networks

A technique is employed to capture the intricate correlations within multi-modality datasets by constructing multiple groups of hyperedge structures. These hyperedge groups are then combined to form a hypergraph adjacent matrix, denoted as H. Subsequently, the hypergraph adjacent matrix H and the node features are fed into a specialized neural network, which is designed to operate on hypergraph data. This HGNN leverages the unique capabilities of hypergraphs to capture high-order data correlations and effectively integrate them with node features, resulting in the generation of meaningful output labels for the nodes.
The hyperedge convolutional layer can be formulated as [3]

$$X^{(l+1)} = \sigma(D_v^{-1/2} H W D_e^{-1} H^T D_v^{-1/2} X^{(l)} \Theta^{(l)}) \tag{8}$$

where $X^{(1)} \in R^{N*C}$ is the signal of hypergraph at l layer, $X^{(1)} = X$ and σ denotes any nonlinear activation function such as relu, sigmoid. D_e *and* D_v denote the diagonal matrices of the edge degrees and the vertex degrees, respectively. The HGNN model utilizes spectral convolution on the hypergraph to leverage high-order correlations within the data. It begins by processing the initial node feature $X^{(1)}$ using a learnable filter matrix $\Theta^{(1)}$, which extracts C2-dimensional features. These node features are then gathered based on the hyperedges, resulting in the hyperedge feature $R^{E \times C2}$, obtained through multiplication with the hypergraph adjacent matrix $H^T \in R^{E * \times N}$. Finally, the output node feature is obtained by aggregating the hyperedge features, achieved through matrix multiplications with H, where the diagonal matrices serve as normalization factors. This enables the HGNN layer to effectively capture high-order correlations within the hypergraph through a node-edge-node transformation. When the hyperedges only connect two vertices, the hypergraph is then simplified into a simple graph.

2.2.4 Implementation of Hypergraph Neural networks

In visual object classification tasks the features of N visual object data can be represented as $X = [x1, ..., xn]^T$, and the hypergraph is built using the Euclidean distance. The construction process involves representing each visual object as a vertex and connecting it with its K nearest neighbors to form hyperedges. These hyperedges link K + 1 vertices, resulting in N hyperedges in total. The incidence matrix H $\in R^{N \times N}$ is then generated, where N $\times$ (K + 1) entries are set to 1 and the remaining entries are set to 0.

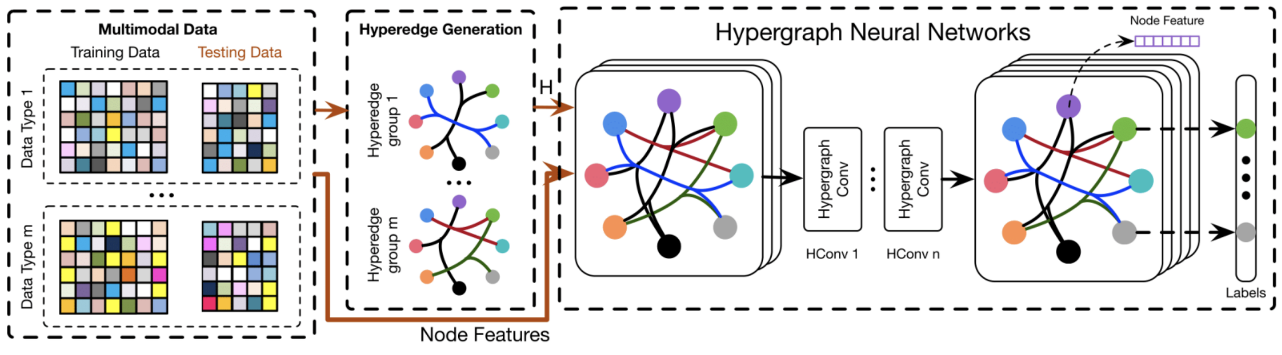


Figure 6: HGNN framework [3]

For node classification task, it involves splitting the dataset into training and test data. From the dataset, a hypergraph is constructed to generate an incidence matrix H and corresponding D_e. Using these hypergraphs, a two-layer Hypergraph Neural Network (HGNN) model is built, taking advantage of the powerful capacity of HGNN layers. Predicted labels are then generated using the softmax function. During training, the parameters Θ are updated using back-propagation and cross-entropy loss for the training data.

In testing, the predicted labels are used to evaluate performance on the test data. When dealing with multi-modal information, hyperedge groups are constructed to incorporate various hyperedges, enabling the modeling of complex relationships in the data. These hyperedges are fused together to capture the intricate relationships and interactions among different modalities.

2.2.5 Experiment and results

In the context of a visual object classification task the paper [3] experiments on two datasets, namely the Princeton ModelNet40 dataset [4] and the National Taiwan University (NTU) [5] 3D model dataset. To construct hypergraphs, two different methods were explored.

The first method involved using a single modality feature, where a centroid object was selected, and its 10 nearest neighbors in the feature space were used to generate hyperedges, including the centroid. This process was repeated for all objects in the dataset, resulting in the construction of a hypergraph denoted as G with N hyperedges.

The second method, on the other hand, utilized multiple features to generate a hypergraph denoted as G, which aimed to model complex multi-modality correlations. For each modality, a hypergraph adjacent matrix denoted as H was constructed, and these matrices were concatenated to build the multi-modality hypergraph adjacent matrix denoted as H.

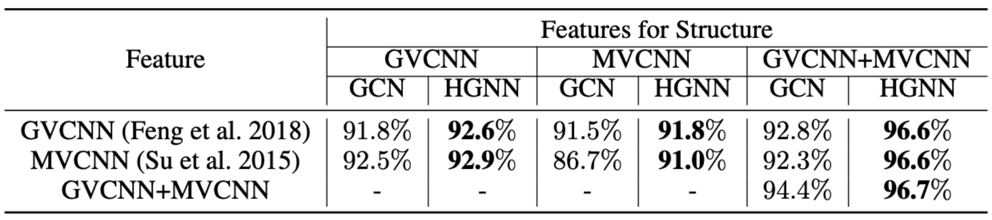
The paper[3] utilizes two advanced shape representation techniques, namely Multi-view Convolutional Neural Network (MVCNN) [6] and Group-View Convolutional Neural Networks (GVCNN) [7], that have shown promising results in representing 3D objects. The authors adopt the same experimental setup as MVCNN and GVCNN to generate multiple views of each 3D object.

Subsequently, the performance of the Hypergraph Neural Network (HGNN) method was compared with other existing methods, including the Multi-view Convolutional Neural Network (MVCNN), Group-View Convolutional Neural Network (GVCNN), and Graphical Convolutional Neural Network, to evaluate their effectiveness in the visual object classification task using the constructed hypergraphs.

The HGNN method proposed in this study outperforms existing object recognition methods on the ModelNet40 dataset. When multiple features are used for graph/hypergraph structure generation, HGNN achieves significant performance gains compared to GCN.

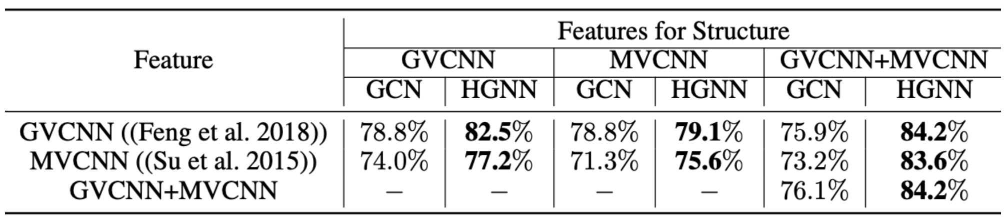
Hypergraph neural networks offer several advantages over standard graph neural networks. One of the main advantages is their flexibility in representing multimodal data. Unlike standard graphs, nodes in a hypergraph can be connected to multiple hyperedges, which can improve the performance of the network. However, this increased flexibility can also make it difficult to understand how the hypergraph arrives at its predictions.

Another limitation of HGNNs is their computational complexity. They may require a large amount of memory and be computationally intensive, especially when dealing with large hypergraphs. To address this issue, techniques such as graph coarsening or mini-batching may be used to handle large hypergraphs. Overall, while HGNNs offer several advantages, their performance and practicality depend on the specific application and the size and complexity of the hypergraph.



Feature	Features for Structure					
	GVCNN		MVCNN		GVCNN+MVCNN	
	GCN	HGNN	GCN	HGNN	GCN	HGNN
GVCNN (Feng et al. 2018)	91.8%	**92.6%**	91.5%	**91.8%**	92.8%	**96.6%**
MVCNN (Su et al. 2015)	92.5%	**92.9%**	86.7%	**91.0%**	92.3%	**96.6%**
GVCNN+MVCNN	-	-	-	-	94.4%	**96.7%**

Figure 7: Comparison between GCN and HGNN on the ModelNet40 dataset [3]



Feature	Features for Structure					
	GVCNN		MVCNN		GVCNN+MVCNN	
	GCN	HGNN	GCN	HGNN	GCN	HGNN
GVCNN ((Feng et al. 2018))	78.8%	**82.5**%	78.8%	**79.1**%	75.9%	**84.2**%
MVCNN ((Su et al. 2015))	74.0%	**77.2**%	71.3%	**75.6**%	73.2%	**83.6**%
GVCNN+MVCNN	–	–	–	–	76.1%	**84.2**%

Figure 8: Comparison between GCN and HGNN on the NTU dataset[3]

2.3 GraphSAGE

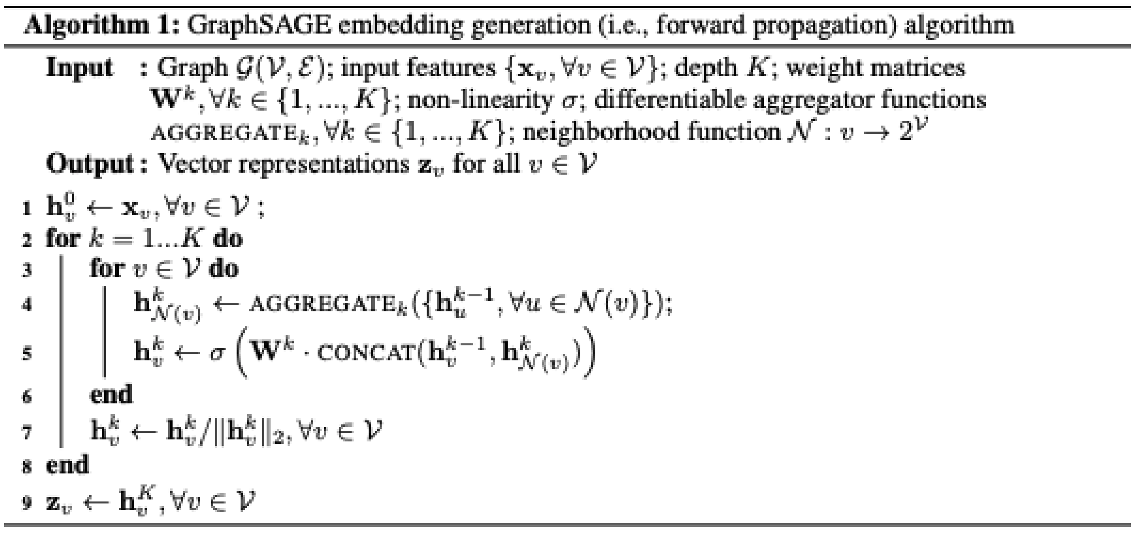
2.3.1 Introduction

GraphSAGE [8] is a type of Graph Neural Network (GNN) that can efficiently generate node embeddings for previously unseen data, which is a crucial requirement for many real-world applications. Unlike other GNN models that focus on node embeddings from a single fixed graph, GraphSAGE leverages node feature information to learn an embedding function that can be generalised to unseen nodes or entirely new graphs or subgraphs.

The key advantage of GraphSAGE is that it learns to recognise the structural properties of a node's neighbourhood, which reveals both the node's local role in the graph and its global position. By using node features, GraphSAGE can learn the topological structure of each node's neighbourhood as well as the distribution of node features in the neighbourhood. This allows it to capture rich contextual information and generate high-quality embeddings.

Unlike matrix factorisation approaches that require a fixed set of training data, GraphSAGE's inductive framework can handle unseen nodes and graphs by using node feature information. This makes it a powerful tool for various applications, such as social network analysis and recommendation systems, where the graph data is constantly evolving and the model needs to adapt to new data quickly.

2.3.2 Learning in GraphSAGE



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

```
Input  : Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices
         $\mathbf{W}^k, \forall k \in \{1,...,K\}$; non-linearity $\sigma$; differentiable aggregator functions
         $\text{AGGREGATE}_k, \forall k \in \{1,...,K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$
Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

Figure 9: : Forward Propagation algorithm for GraphSAGE [8]

GraphSAGE is a graph neural network approach that consists of two main processes: node embedding generation and model parameter learning. The node embedding generation algorithm assumes that the model has been trained and the parameters are fixed. The algorithm works by iterative aggregation of information from the local neighborhood of each node in the graph. This process continues until nodes have access to information from nodes that are further away in the graph. The graph G=(V, E) and features for all nodes x_v are provided as input.

The learning algorithm involves applying a graph-based loss function to the output representation of each node, denoted as z_u, where u is a node in the graph. The weight matrices, W^k, and the parameters of the aggregator functions are learned via stochastic gradient descent, for all k in the range of 1 to K. The learning algorithm works by minimizing the loss function and adjusting the weight matrices and parameters of the aggregator functions accordingly.

The objective of the loss function is to encourage the representations of nodes that are close to each other in the graph to be similar, while at the same time ensuring that the representations of nodes that are not closely related are very different from each other. It can be expressed as :

$$J_G(z_u) = -log(\sigma(z_u^T z_v)) - Q.E_{v_n \sim P_n(v)} log(\sigma(-z_u^T z_{v_n})), \tag{9}$$

where v is a node that co-occurs near u on a fixed-length random walk, σ is the sigmoid function, P_n is a negative sampling distribution, and Q defines the number of negative samples. The z_u in the loss function for GraphSAGE is generated from the features contained within a node's local neighborhood.

GraphSAGE uses aggregator functions to combine information from the neighborhood of a given node. These functions must operate over an unordered set of vectors and be symmetric, meaning they can be applied to arbitrary ordered node neighborhood feature sets. The paper explores three different aggregator functions.

- The Mean Aggregator function calculates the element-wise mean of the vectors in the neighborhood of a node.
- The LSTM Aggregator function adapts Long Short-Term Memory (LSTM) networks by applying them to a random permutation of the node's neighbors.
- The Pooling Aggregator function independently feeds each neighbor's vector through a fully-connected neural network, then applies an element-wise max-pooling operation to aggregate information across the neighborhood. The resulting vector is symmetric and can be used as input to the next layer of the neural network.

$$AGGREGATE_k^{pool} = max(\{\sigma(W_k h_{u_i}^k + b), \forall u_i \in N(v)\}) \tag{10}$$

where max denotes the element-wise max operator and σ is a nonlinear activation function.

2.3.3 Experiments and results

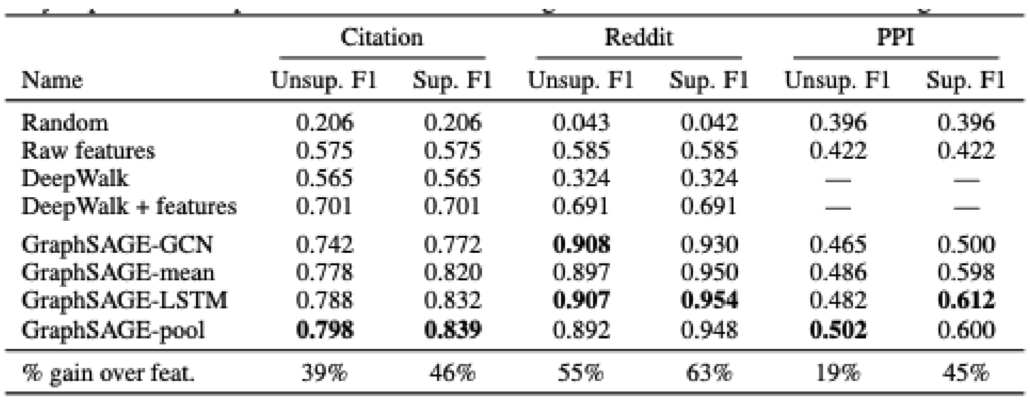
The GraphSAGE algorithm was evaluated on three distinct tasks in the paper[8]. These were: (i) classification of academic papers into various subjects using the Web of Science citation dataset, (ii) classification of Reddit posts into different communities, and (iii) classification of protein functions across various biological protein-protein interaction (PPI) graphs.

To compare the performance of GraphSAGE with other methods, four other models were used: a random classifier, a logistic regression feature-based classifier, a DEEPWALK algorithm, and a concatenation of the raw features and DeepWalk embeddings. Different aggregator functions were also tested with GraphSAGE, including LSTM-, pool-, and mean-based aggregators, as well as the "convolutional" variant of GraphSAGE called GraphSAGE-GCN.

The results showed that the LSTM and pool-based aggregators performed the best on average, both in terms of overall performance and the number of experimental settings where they outperformed other methods. All three aggregators (LSTM, pool, and mean) were found to provide statistically significant gains over the GCN-based approach. However, there was no significant difference between the LSTM and pool approaches.

It was noted that GraphSAGE-LSTM was found to be significantly slower than GraphSAGE-pool, with a speed difference of approximately 2 times. This could give the pooling-based aggregator a slight edge overall.

One of the key benefits of GraphSAGE is its scalability to large graphs by learning node representations from only local neighborhood information. However, it is crucial to experiment with different aggregation functions as the choice of aggregation function can affect the model's performance. Additionally, GraphSAGE can be computationally intensive, especially for larger graphs.



	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

Figure 10: Prediction results for the three datasets[8]

3 Applications

As a graph is made of structured-data there are many fields in which they can be used. Therefore, it makes sense that the use of GNNs could also be applied to many fields.

]Images [9][10] [11] [12]

One major application of GNNs is for them to be used for image classification. While this can be done with other neural network types like CNN, the process is much different. The main difference is that unlike CNN, where the data is entered in a regular grid, the data entered is in the structure of a graph which allows for better recognition of the relationship of the pixels. While this may seem like it is better in every way, as the resolution of the image increases so does the size of the graph needed as the number of pixels increases. As the pixels are considered the nodes this means along for every additional pixel there are even more edges needed to be connected from that one pixel to other pixels. One more thing to take note of is that the pixels can be connected in various ways. The most basic connection is where every pixel is connected with an edge to an adjacent pixel. Another way to connect them is to include pixels that are diagonal from them as even though they are not adjacent to the pixels they are touching, therefore, it makes sense for them to be connected. An extreme connection case is for the biggest graph that you can make is for the pixels to be set it up so that each pixel is affected/connected by every other pixel.

One image classification type would be multi-label classification where the model can identify multiple labels in an image. One further step than is to use this multi-label classification and convert the image into a graph-like structure that shows the relationship between the labels in the image. For example, if an image of fish on a table a graph is constructed that shows the placement of the fish relative to the table. This application of image classification gives a greater ability to describe the image. Therefore when you wish to test this upon an unlabelled test dataset you will be able to generate a description that more accurately describes the test image.

A similar enhanced aspect of this method is that GNNs can be used to detect fake images. This aspect also shows that to create these fake images, you would need the ability to convert the image given into graph form without much interaction from the user. One useful application for detecting false images is deepfake detection. As some image graphs can look at pixels that are not just adjacent to them, the graph neural network model would be able to detect subtle differences that would not be possible if a graph was not used. These subtle differences would allow the model to determine if the image has been manipulated in any way.

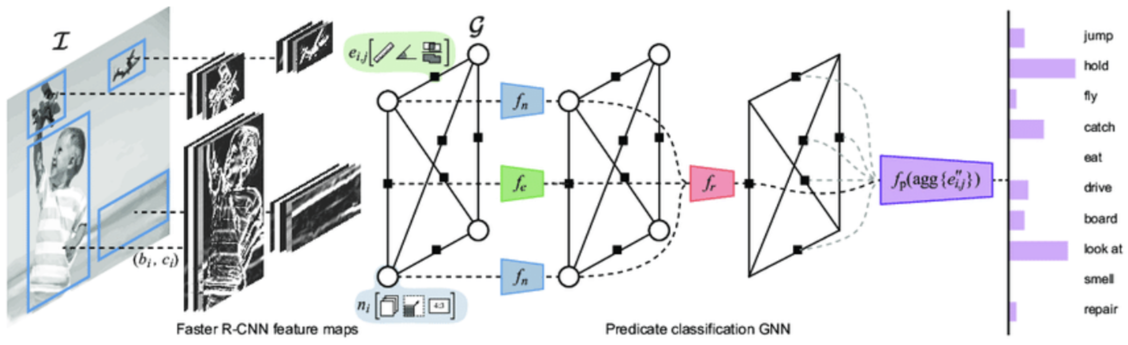


Figure 11: : Example of Image Classification.[11]

3.0.1 Images Limitations/Challenges

A common challenge for images is that they are highly affected by noise as even a slight amount of noise can completely throw off an image classification task on a regular neural network, and the problem can be even worse for a graph neural network. This is due to how the pixels can have a greater connection with other pixels, so it makes sense that the noise has a greater impact on the surrounding pixels. Depending on the number of connections between pixels, the impact can be even greater.

In order to have an effective model that can tell the relationship between the labels in a multi-label classification you need to have your training data to be properly annotated. This is needed because because if the relationship between the labels is not properly explained it would cause the model to be trained on an incorrect description. Additionally, you will need to have enough of the same description in the training set in order to properly define the relationship. For example, if the relationship between ball and shirt is that the ball is wearing the shirt but this description only appears once or twice when the model begins testing on the testing set there would be not enough training data to properly define this relationship.

One of the universal challenges for graph neural networks is the computation time. This is especially true for image datasets as the pixels are the nodes, which means that for images of higher quality or size, the node count would be large and the number of edges is even greater. As the number of edges depends on how the image is set up, if there are only edges for the pixels that are next to (adjacent or diagonal) it then it might be somewhat reasonable, but if you have your nodes be connected globally, meaning that every node is connected to every other node, it creates a substantially large number of edges.

3.0.2 Experiment

We have decided to explore superpixel, one of the possible solutions to decrease the computational runtime. Superpixels are pixels combined certain pixels into one giant pixel which will also count as one node. We ran this superpixel version of the MNIST dataset called MINSTSuperpixels.The MNIST Superpixels dataset is a graph classification benchmark dataset provided by the PyTorch Geometric library. It is based on the classic MNIST dataset of handwritten digits but with additional annotations for superpixels. In this dataset, each image is represented as a graph, where nodes correspond to individual pixels, and edges connect adjacent pixels. Each node is also associated with a feature vector representing the pixel's color intensity value. The graph is annotated with a target class label indicating the digit that the image represents. We passed this dataset to train a simple Graph Neural Network model. MNIST Superpixels dataset is not inherently constrained between 0 and 1, but it is often preprocessed to have input values within that range for compatibility with machine learning models. We used negative log-likelihood loss as it encourages the model to output high probabilities for the correct class label, and low probabilities for the incorrect ones. The reason why we chose a simple model is due to a few reasons. The first reason was the unknown expected runtime. We were unsure about the time we needed to debug the model if there were errors so we figured that a simpler model would be easier to debug. Additionally, we also figured that a simple model with fewer layers would have a significantly less runtime than a more complicated model with more layers. Another

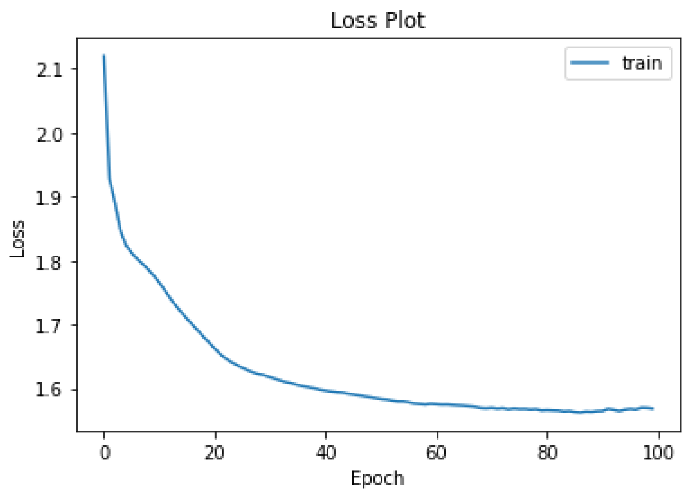


Figure 12: : Training Loss Plot

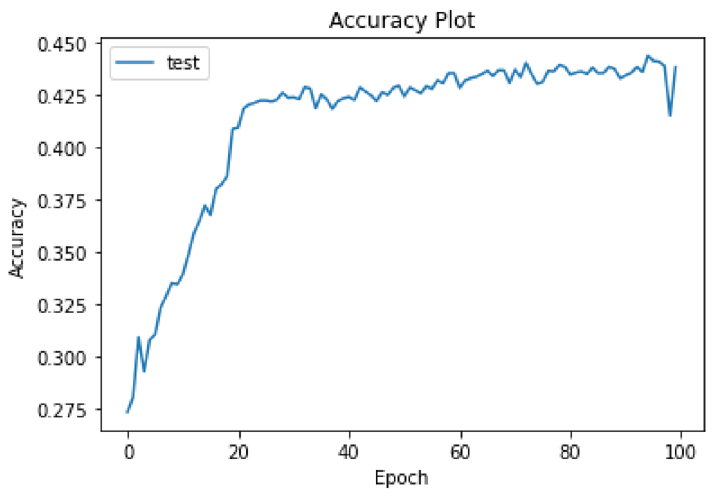


Figure 13: : Test Accuracy

reason why we chose a simple model, was that we wanted to see the impact of computational runtime as some more complicated models are capable of reducing the runtime through certain techniques. We noticed that the training loss of the model reduced with every iteration and it begins to steady after the 80th iteration. The best test accuracy of the model was 0.441 which was at the 95th epoch. While our model was not able to receive the best accuracy, that was our secondary objective as we were able to properly measure the computational runtime. The total runtime for 100 epochs on the superpixel MNIST dataset using a simple model was about 15,600 seconds or about 4 hours and 20 minutes.

3.1 Text [13][14][15][4]

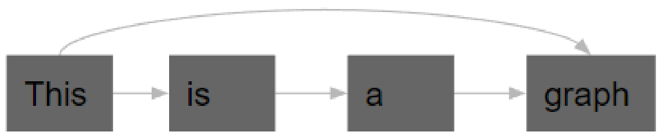


Figure 14: : Simple text graph.

GNNs can easily be modified for NLP problems, such as text classification, because in NLP problems you need to take into account how the words interact with each other and what their relationship is with each other, so it makes sense to put this into a graph form for a better representation. This advantage allows for the words' relationship to be much easier represented. While it is easier to show the words' relationship, there are many different ways to construct the graph needed to show this relationship. As seen in Figure 14, this is an example of a simple text graph that shows how every word is related. This can be considered one of the most basic as it just shows the relationship between only the whole words. A more advanced graph would be to incorporate syntax into the graph as it better defines the structure of the sentence. Another way to make a graph is similar to the GloVe embeddings (GloVe embeddings show the probability of the co-occurrence between words as vectors), but instead, the vectors are transformed into a graph version. This version would work for sentimental analysis because you already have something similar to GloVe and the aspects of what the results should be like are pretty clear.

One application of using this information is that you can train a GNN to detect fake news. This is an important application as anyone can post whatever they want online, it becomes important to know what is real and what is fake. As this is a more advanced technique of using GNN on text data, you need to know that to do this it performs both sentiment analysis on the fake news to help to determine its score which is used with text classification to finalize the classification of whether the text information given is fake news or not.

3.1.1 Text Limitations/Challenges

There are limitations to using graph neural networks in NLP problems. For example, it is best to use datasets that are completely correct but that is not possible in the real world, especially in terms of a person's writing ability. Imagine that you have trained a graph neural network on a well-label dataset with a well-tuned model but the text given to be tested is incorrect as it contains numerous grammar, spelling, and other errors. So it would be harder to convert this bad text given to a graph as it does not contain a well-defined structure like the text in the training set. One thing to take note of as like most NLP problems is that a large amount of data is needed to train an NLP model correctly and a graph-based model might be more accurate but it also might take a larger time to train.

As the English language can be quite complicated, it stands to reason that computers may not be able to fully understand some of the expressions we use and one example of this is sarcasm. As sarcasm is used to say the opposite of what a person really means, this would be hard for the graph to realize as if sentiment analysis was used on a sarcastic sentence then it would most likely get a score opposite of what it should really get. A similar problem that graph neural network encounters is poor performance in logical reasoning. As logical reasoning runs on logic meaning that the model would need to have an understanding of the complex rules of the English language. Additionally, even if the model is able to understand the complex rules it does not mean that the results would be correct as there are statements that can be ambiguous and can not be properly learned. For example, the statement ”Put those clothes in the box on the table” can be interpreted in two different ways. The first common way would be to assume that the clothes need to be in the box and the box has to be on the table. However, the interpretation that the clothes just need to be in the box that is currently placed on the table is also correct. While this may seem like not much of a difference, this is only an example of a simple ambiguous statement. More complex ambiguous statements would be even harder to identify correctly and there is a higher chance of different interpretations having different meanings. This would be especially true if the model was only trained on datasets that contain straightforward, clear statements that do not contain any ambiguity.

3.1.2 Experiment

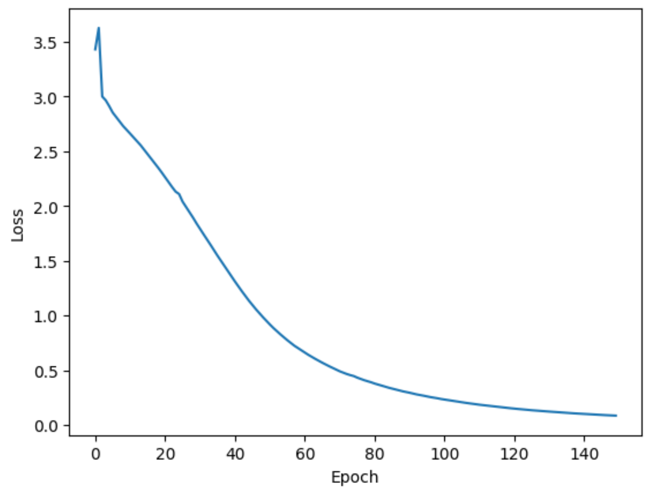


Figure 15: : Training Loss Plot

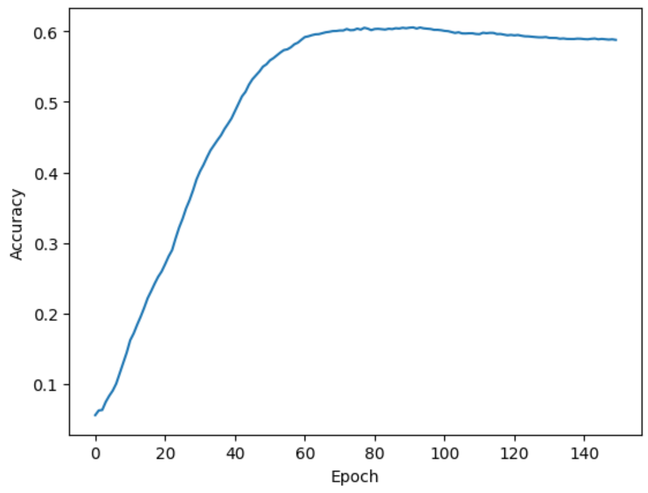


Figure 16: : Test Accuracy

We used a graph neural network to classify 20,000 newsgroup documents into 20 different categories like politics, religion, and sports. We converted text data into numerical vectors using CountVectorizer and created a graph where each node corresponds to a document, and edges connect similar pairs. We defined a GraphSAGE model with SAGEConv as the layer for updating node representations and GraphConv as the final layer for mapping them to class labels. The model uses a weighted sum of neighbor representations and the node's own representation to update node representations. Finally, we passed the output through a softmax function to get the class probabilities. We ran the model for 150 epochs and noticed that the training loss continued to steadily decrease and being flattening out after 140 epochs. We noticed that the accuracy continued to improve to 0.6 and then flattens out.

3.2 Recommendation System/Social Networking [16] [17]

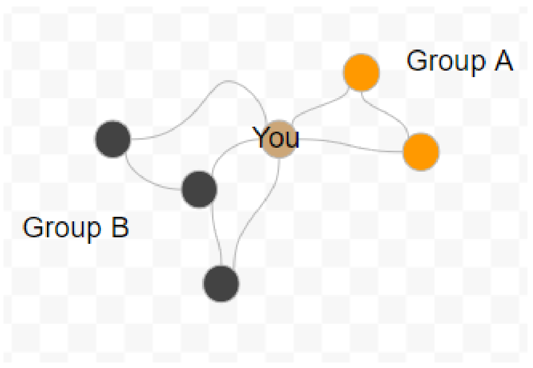


Figure 17: : Social Networking Graph

Another application for graph neural networks is a recommendation system. For a recommendation system, the nodes in the graphs will be the recommended items and the edges be based on the user's past search history and feedback. Additionally, you will need the graph to be able to add new users and items. As in real life there are always new products coming out and users joining. Even more so, by adding new items/users the graph will need to be updated to better reflect its current state allowing you to have access to new/preferred searches that fit based on the updated graph.

A similar approach is used for analyzing/creating a social networking graph. In a social networking graph, a node would be a person and the edges would be the connection they have with other people. This allows for a person to be given an idea of what other people are into if they have a connection too. One difference from the recommendation system is that the social networking system could be more complex as the edges can take much more into account depending on the amount of information given. For example, if a person may have a connection based on a television show and the person could be constantly discussing their opinions of the show, but if the show takes a sharp turn then there is a chance that the person's opinion of the show could completely change. This sudden change of opinion enforces the need for a graph's edges to be updated as a lack of updates could connect you with people who you no longer have a connection with.

3.2.1 Limitations/Challenges

One of the biggest challenges when dealing with graph neural networks is that the graphs themselves can be very big. For the recommendation system and social networking graph mentioned in this section, there is no limit on how big these graphs can get, and a bigger graph means that the computation time will increase and even then those recommendations could either be completely biased from just one topic to any little topic you have information for depending on the users use. This will cause problems in these types of graphs as you can be dealing with millions of different items/users. One possible solution is to use a method similar to GraphSAGE where you intentionally look at fewer nodes and edges to overcome this runtime issue and it is highly likely that as the deeper into the graph, the results will be less relevant. Inversely the opposite faces the opposite problems. If there is not enough data then the results would be lopsided which would especially be the case for new users as new users are lacking any sort of data making it hard to give any recommendations and the ones they do give would be heavily biased to what is the current most popular item in the graph. Another problem with the recommendation system is the lack of a proper reason why an item is recommended. For example, if you looked at a movie and were recommended a table and the reason for the recommendation was that other users also looked at both of these items before. This lack of a proper reason makes it harder for users to trust the recommendation process as they would like to know a more in-depth reason why the recommendation item is good for them.

3.3 Traffic Prediction

For traffic prediction you would initially assume that the nodes would be the destinations/cities and the edges would be the roads in between the nodes. However, the nodes would also include every intersection of roads. The reason for this is that whenever roads intersect means that there multiple directions that the car can go in. For example, for a 4-way intersection, a car has 3 legal directions and one illegal direction, but this illegal direction can be ignored as it makes no sense for a car to travel backward. One thing that you need to take into account when building a traffic prediction graph is that not all vehicles will be allowed on all roads. For example, if you are driving a large truck and there is a bridge that the truck can not fit under you will need to take into account an alternative route. Depending on what/where you are trying to make a traffic map the graph itself could be very big. That graph would be even bigger when you take into account that the edges between the nodes could be either directional or bidirectional as you need to calculate the traffic going in both directions for it to be accurate. Two ways to model graphs for traffic prediction are to either rely on past structured data or have it be updated constantly, such as a traffic accident.

3.3.1 Limitations/Challenges

One of the challenges with relying on past structured data for traffic prediction is that it does not properly take into account any unexpected events that might make a road unusable. These graphs might have to incorporate how likely a traffic accident is going to happen, when it is going to happen, and how long the delay will take. Another problem with relying on past structured data is that the roads may not be up to date therefore, the graph is missing roads that have been added and the target destination may no longer be in the place recorded in the graph. This is one of the reasons why some traffic prediction graphs are updated in real-time, but they are not without their challenges. One such challenge is deciding how deep the graph needs to go to accurately calculate the traffic prediction from your starting point to your target destination. For example, if the graph is set up so that you travel the shortest distance it does not necessarily mean that this way is the fastest. This could be due to the path taking you into a city where there is a large amount of traffic and it could be faster to avoid the city entirely. A problem with having a graph that is being deeply searched you may end up looking at routes that have no real purpose, such as if you are traveling from the middle of a state/country to one end but the graph might end up looking at the entire state/country as they are theoretically the same distance apart.

3.4 Medical [18]

The use of graph neural networks can also able to be applied to the medical field in many different ways. The first way is for diagnosis prediction and analysis. By treating the symptoms as the edges which connect possible diagnoses as nodes you can predict some of the possible diagnoses. This can be done as a graph neural network can take in data from multiple sources needed about a person's medical history. This would allow for medical professionals to obtain a second opinion on a patient which can be vital as sometimes medical professionals are busy and will not have much time for each patient. This lack of time can cause them to make mistakes such as not interpreting their symptoms correctly or even missing those symptoms completely.

Another medical use is to treat the brain itself as a giant graph to track brain activity. By mapping the brain synapses, you be able to convert this mapping into a graph form to be able to track if there is anything wrong with the brain. This can be used to see if the brain is properly responding to certain stimuli. This would allow us to better show the relationship between brain activity and cognitive functions. The results would allow researchers to know more about neurological disorders.

3.4.1 Limitations

There are currently a few issues present for using graph neural networks in the medical field. The first issue is the accuracy of using a person's information. It would not be uncommon for a person to be unaware of a condition that they have or to lie about their condition. This means that you are entering incorrect/false information into the graph neural network which will in turn be used to update the information already present. As the amount of wrong information is increased this will result in the accuracy to decrease. While graph neural networks are able to predict nodes and edges, there is the possibility that due to the amount of wrong information given that the model would not be able to overcome this issue.

Another challenge with using a person's information in the case of rare or new diseases. For rare diseases, there will not be much information from the dataset for these cases which could make it harder to be predicted accurately or alternatively give a false positive result. In a false positive result case, the person would then need to take tests to truly confirm they have this rare disease and it might take a mental toll on them. Conversely, in the false negative case, it would be not that much different than if a graph neural network was not used as it can take a long time for rare diseases to be diagnosed correctly. There is a bigger problem if the person has an unknown new disease. In this case, the correct diagnosis will never be properly given and would have to wait until the new disease is identified, but even then it would be hard to use a graph neural network. This is due to how you would have to update the dataset to include this new disease. Meaning that you have to accurately update the graph to include the new disease, but the problem is due to the lack of information about it accurately adding it to the graph is basically impossible.

Another issue is that scientists are still learning about the human body. Unlike in the text and images sections where we know all about the meaning behind the data and by using the same information with different models, the accuracy can be improved. In the human body case, every new discovery means that there is a chance that any preexisting model for prediction will have to be changed to properly reflect the data given. Additionally, this means that there is not one fit-all model but rather multiple models that have to be remade. Furthermore, there are cases like the brain where it might end up being too complex to model.

4 Conclusion

One of the biggest challenges in making a graph neural network model from scratch is if you are using a custom dataset then you have a large amount of data to label. We learned this when we attempted to make a dataset made from tweets scraped from Twitter. At first, we thought we just had to run a sentimental analysis on every tweet and then put them into our graph, but then we realized that we were unsure of how to connect them properly. The keyword here is properly as we realized that while we could technically through any value in as the edges there is no way to tell if the values we put in were correct or not. This is a huge problem as incorrectly weighted edges makes the purpose of a graph to be worthless. We learned from the experience how difficult the setup for a graph neural network dataset can be as you do not just obtain the data for a node, you also have to calculate the best initial value for the edges. As the number of node increase so does the number of edges. This does help account for why graph neural networks can be computationally expensive as there are graphs whose nodes are connected to every other node. While depending on the application type, there have been successful attempts to lower the computational time which normally means that they end up not using the whole graph. However this may sound bad at first, but it does happen as some graph such as the social networking and recommendation system has the potential for the information on the graph to never stop being fed new information. While for traffic prediction, most likely you will not care about the traffic on the other side of the country so there is no reason why you should take that information into account.

In conclusion, graph neural networks have shown significant promise in various applications such as image and text classification, recommendation systems/social networking, traffic prediction, and medical use. Graph neural networks can successfully analyze and explain complex images. It can also be used to analyze and classify text correctly and detect incorrect information. Recommendation systems and social networking offer the ability to link to new items/people that would not have originally. Traffic prediction is a useful tool in everyday life. Graph neural networks in the medical field are still an ever-evolving process to better model the human body. Hypergraph neural networks offer increased flexibility in representing multimodal data and can outperform standard graph neural networks in certain scenarios. However, they can be computationally intensive and difficult to interpret. GraphSAGE is a scalable approach to learning from large graphs by using only local neighborhood information. Overall, graph neural networks are a powerful tool for analyzing structured data and have the potential to significantly advance many fields.

5 Individual Contribution

Work done	**Who**
Report Writing Up	All
Main Coder	Balassubramanian
Coder Helper	Nicholas

Table 1: Contribution

6 Code

https://drive.google.com/drive/folders/1l0IgTZzCrw5sIULCODH16KObflzOs$_8T?usp = sharing$

7 References

References

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," Feb 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[3] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," Feb 2019. [Online]. Available: https://arxiv.org/abs/1809.09401

[4] Z. Wu, J. Xiao, X. Tang, L. Zhang, F. Yu, A. Khosla, and S. Song, "3d shapenets: A deep representation for volumetric shapes." [Online]. Available: https://3dvision.princeton.edu/projects/2014/3DShapeNets/

[5] P.-C. Wu, Y.-Y. Lee, H.-Y. Tseng, H.-I. Ho, M.-H. Yang, and S.-Y. Chien, "A benchmark dataset for 6dof object pose tracking," 2017.

[6] H. Su, S. Maji, E. Kalogerakis, and E. Learned Miller, "Multi-view convolutional neural networks for 3d shape recognition - arxiv." [Online]. Available: https://arxiv.org/pdf/1505.00880.pdf

[7] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "Gvcnn: Group-view convolutional neural networks for 3d shape recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 264–272.

[8] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Sep 2018. [Online]. Available: https://arxiv.org/abs/1706.02216

[9] F. Baldassarre and H. Azizpour, "Explainability techniques for graph convolutional networks," May 2019. [Online]. Available: https://arxiv.org/abs/1905.13686

[10] M. Khademi and O. Schulte, "Deep generative probabilistic graph neural networks for scene graph generation." [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6783

[11] F. Baldassarre, K. Smith, J. Sullivan, and H. Azizpour, "Explanation-based weakly-supervised learning of visual relations with graph networks," Jul 2020. [Online]. Available: https://arxiv.org/abs/2006.09562

[12]

[13] M. Ryabinin, S. Popov, L. Prokhorenkova, and E. Voita, "Embedding words in non-vector space with unsupervised graph learning." [Online]. Available: https://aclanthology.org/2020.emnlp-main.594/

[14] S. Rode-Hasinger, A. Kruspe, and X. X. Zhu, "True or false? detecting false information on social media using graph neural networks." [Online]. Available: https://aclanthology.org/2022.wnut-1.24/

[15] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song, "Can graph neural networks help logic reasoning?" Sep 2019. [Online]. Available: https://arxiv.org/abs/1906.02111

[16] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," Jan 2019. [Online]. Available: https://arxiv.org/abs/1811.00855

[17] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," Nov 2019. [Online]. Available: https://arxiv.org/abs/1902.07243

[18] D. Ahmedt-Aristizabal, M. A. Armin, S. Denman, C. Fookes, and L. Petersson, "Graph-based deep learning for medical diagnosis and analysis: Past, present and future," May 2021. [Online]. Available: https://arxiv.org/abs/2105.13137

[19] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," Jun 2015. [Online]. Available: https://arxiv.org/abs/1506.05163

[20] W. Xu, J. Wu, Q. Liu, S. Wu, and L. Wang, "Evidence-aware fake news detection with graph neural networks," Feb 2022. [Online]. Available: https://arxiv.org/abs/2201.06885

[21] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157