A Comparison of Node-wise Classification of Graph Neural Networks (GNN)

Candidates: 43577, 45128, 49640

Abstract

This report aims to explore the performance of node classification between different Graph Neural Network architectures and a conventional Feed Forward Neural Network using the `Cora` citations dataset. The analysis involved identifies the best performing configuration settings for an initial Graph Neural Network model and compares the proposed neural network architectures in terms of node classification accuracy on a test dataset.

1. Introduction

Node classification is a vital field within machine learning which enables us to apply heuristic decision making and prediction processes to tasks that involve graph data structures. The process of handling these complex systems allows us to perform complex tasks, such as social network analysis and recommendation systems. The process of learning the interactions between nodes in an intelligent way is paramount to the models predictive capability. Conventional classification models are often unsuitable in this context because of the complex nature of the non-euclidean data and the methods of learning graph networks, as discussed in relevant literature (Bronstein, 2017) (1). In this paper, the authors are comparing the node classification accuracy of several models that learn these representations of both global and local information in graph data on the Cora dataset. The results indicate that graph neural networks outperform a regular feed-forward neural network in this context.

2. Related work

In recent papers, the method of node classification used several different models with varying degrees of success. Common methods used are a graph convolutional network (GCN) with custom layer design as described in a more recent literature (Defferrard, 2016) (3). The paper divulges a learning operation where the classifications utilises a learnt filter to summarise message-passed information more efficiently. Although an intuitive idea, it does not discuss the disadvantages to other methods as described in this project with the GraphSage and parallel architectures. The paper that introduces the concept of GCN (Kipf, 2017) (5) is related in the goal of node-classification. It utilises a semi-supervised method that maps a function of the feature vectors to better suit their labels. The methods used are applicable in this setting and are a good basis to provide simple models to start with, but fall short in providing a comparison of accuracy between newer models. One such newer model is proposed in the "Attention-based Graph neural network" shown in (Zhou, 2018) (14). It proposes an attention mechanism that learns the importance of different nodes and edges in the graph, allowing for a better global aggregation of the neighbourhood. It is tested on the same citation dataset where it is observed to have an accuracy of 83.98%. A significant figure that shows an outperformance in many methods tested prior to 2018. This differs from the approach taken within this project. The authors provided a unique and benchmark-driven environment to observe the behaviour of models in comparison to eachother specifically unlike (Zhou, 2018) (14). Depending on the results shown the AGNN might prove a better more generalise-able model in comparison to methods shown here. One article similar in objective to the current paper is "A Comprehensive Survey on Graph Neural Networks" (Wu, 2021) (11) where the authors main objective was to observe models performance differences based on different contexts and benchmarks. The article focuses more so on supervised learning tasks as opposed to semi-supervised methods. Alongside the lack of interpretability, the paper covers a large number of external papers which increases the complexity and possible room for error for the paper. The current paper will only focus on a few select interesting models on the same dataset, providing an overall exhaustive perspective on performance differences with interpretable results. In conclusion, there are little papers that provide the comparison provided in the current study in a conclusive and intuitive manner which this paper aims to provide.

3. Theoretical Framework

In order to understand how a graph neural network (GNN) differs from a conventional model and the potential performance increases between the variations in models, one

needs to understand how they work initially from an abstracted perspective. When training a GNN the process captures weights and biases within a network that describe the relationship between a feature vector and label. The advantage of using the non-euclidean method of representing data in a graph structure is the ability to capture the relationship between these different instances of each response and hence feature vector. This allows the algorithm to leverage the relationship and answer the following questions:

- *Can we improve accuracy based upon the connections of a node?*
- *Are similar nodes connected to each-other?*
- *What is the best predictor in terms of the class of node connected to others?*

In order to capture and process this information, one needs to have embedded representations that can easily be observed to measure the "difference" in nodes that are connected. This is what the message passing phase attempts to achieve with aggregation and concatenation methods. In general, the process is as follows:

1. Each node has a feature vector initialized for its representation. This feature vector is a node/tuples attribute used to recognise and differentiate papers between each other. *In the context of the Cora citations dataset, the node feature vector corresponds to a one-hot encoded 0/1 valued vector indicating the absence/presence of a word from the research paper*
2. Feature vectors of every node are used to generate "messages" that are passed to nodes adjacent to that node. This message is a function of the features of the sender and the receiver nodes. In most settings this is a concatenation function.
3. The message aggregation step is where all messages that are passed to a node are accumulated. This is done to generate new feature vectors. Neural networks or simple summations can be used here depending on the architecture.
4. The feature vector from Step 1 is updated with this new embedded representation from other nodes. The information contained within the new feature vector is an accumulated representation from node neighbors and connections defined within the dataset.

Figure 6 in the appendix shows the step-by-step process with an illustrated example that is more comprehensible. The colour of the node represents the class with red/green being their respective own categories and white being an unclassified node.

From the process described in appendix Figure (6), the authors develop an "averaged" perspective of feature vectors in the network. The messaging passing process described can be repeated for as many times as necessary in the models architecture to pass and average information from nodes further than one edge away. Figure 6 shows how this process can develop our understanding of relationships in just one summarised vector for each node. When performing new-node classification, the extra node is connected to the network and the message passing is performed again to obtain an aggregation vector. This is observable within the diagram at the bottom of figure 6. The averaged connections are used in creating the vector that describes the label. We can clearly see from the connections in the figure that the new node is classified (with some uncertainty) as **green**.

The overall architecture of a GNN can differ from many different aspects such as:

- Aggregation methods
- Concatenation methods
- Classification models on feature vectors
- Number of message passing layers

These aspects are explored within this paper. This report explores four main methods of node classification and a conventional feed-forward model which are compared and analysed in terms of their respective node classification accuracy on a test dataset.

4. Dataset Description

Comparing accuracy of models requires a dataset readily available and comprehensible from many different models. For this reason, this study uses the `Cora` citations dataset, introduced by McCallum, A.K., Nigam et al. in "Automating the Construction of Internet Portals with Machine Learning" (7). This dataset is selected due to its ease of access as a pre-made graph, and consists of 2708 scientific publications classified into one of seven classes. Each scientific publication corresponds to a node in the graph. The citation network consists of 5429 links or edges which connects two publications if one cites the other. Moreover, each publication in the dataset is described by a one-hot encoded 0/1-valued vector indicating the absence/presence of the corresponding word from a dictionary of 1433 unique words. This dataset also contains some interesting features to explore such as:

- Sparsity with some papers only citing other single papers. This means smaller components within the graph that some models may struggle to classify.
- Densely connected components with some nodes having number of degrees, making it more suitable for specific models.
- Non-binary classification data, requiring more complex models to capture relationship between data.

One can observe the graphs components and features described in the appendix Figure (7).

5. Model Architectures

5.1. Feedforward Neural Network Model- Baseline model

The baseline model used is a Feedforward Neural Network (FNN), as introduced by Khalid Salama in "Node Classification with Graph Neural Networks" (8) with the following architecture:

1. Five FFN blocks with skip connections, so that the baseline model has comparable number of parameters as the GNN models.

2. Each FFN block consists of two sequences of: a Batch Normalisation layer followed by a dropout layer that drops out 50% of the features during training, and a dense layer with 32 units and GeLU activation function.

3. The output of the fifth FFN block is fed into a dense layer that computes the logits associated with each class.

The flowchart in Figure (1) provides a visual representation of this model's architecture:

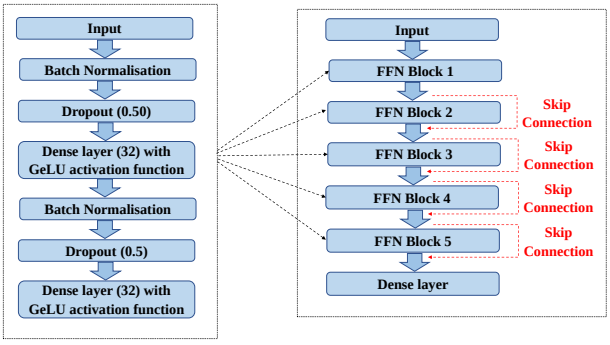

Figure 1. Architecture of Feedforward Neural Network model

5.2. Initial Graph Neural Network Model

The first graph neural network model constructed involves the implementation of a Graph Convolution Layer from scratch using tools from the `Keras` library (8). This graph neural network follows the "Design Space for Graph Neural Networks" (13) approach. In order to provide a clear understanding of this model's architecture, this subsection is split into three parts each detailing its main components.

5.2.1. Intra-Layer Design

The intra-layer design corresponds to the neural network architecture implemented in Step 2 of the Feedforward Neural Network Model in the previous subsection. As part of the hyperparameter selection process, different choices for the activation function, batch normalisation, dropout rate and number of units in each of the dense layers is selected and compared in terms of their respective classification accuracy. This intra-layer design forms the basis for the implementation of the Preparation and Updating steps in the Graph Convolutional layer, presented in the next subsection. Moreover, the intra-layer design is utilised for performing the preprocess and postprocess layers of the inter-layer design to be discussed later. The rationale behind the creation of this intra-layer design is that this common design can be called and used in the various components of the GNN architecture without the need to construct it each time.

5.2.2. Graph Convolutional Layer

As per "Node Classification with Graph Neural Networks" (8), the Graph Convolutional layer performs the following steps:

1. **Preparation**: The input node representations are processed using the intra-layer design introduced above to produce a *message* of information.

2. **Aggregation**: The messages of the neighbours of each node are aggregated with respect to edge weights using a *permutation invariant* pooling operation, such as sum, mean, min, max or prod. This process prepares a single aggregated message for each node.

3. **Updating**: The node representations and aggregated messages produced in the previous step are combined and processed to produce a new state of node representations, or node embeddings. This is achieved through the addition or concatenation of the node representations with the aggregated messages, followed by a processing step using the intra-layer design. According to the same source, "if the combination type is GRU, the node representations and aggregated messages are stacked to create a sequence, then processes by a GRU layer" (8).

The graph convolution layer discussed above adopts ideas from Graph Convolutional Networks (5), GraphSage (4), Graph Isomorphism Network (12), Simple Graph Networks (10) and Gated Graph Sequence Neural Networks (6).

5.2.3. Inter-Layer Design

Having defined the Graph Convolutional Layer, the last and most important level of design is how the pre-defined components of the GNN are organised together into a neural network. In this initial graph neural network model, the design adopts ideas from "Design Space for Graph Neural Networks" (13), as follows:

1. Apply a pre-processing layer using the intra-layer design to the node features to generate initial node representations.

2. Apply two or more graph convolutional layers as defined in the previous subsection, with skip connections to the node

representation to produce node embeddings.

3. Apply a post-processing layer according to the intra-layer design to the node embeddings to generate the final node embeddings.

4. Feed the node embeddings into a Softmax layer to predict the node class.

The main advantage of adopting this inter-layer design is that "each graph convolutional layer added captures information from a further level of neighbours" (8).

Similarly to the intra-layer design approach, as part of the hyperparameter selection process, different choices for the number of graph convolutional layers, aggregation, combination, and skip connection types are selected and compared in terms of their respective classification accuracy.

The flowchart in Figure (2) provides a visual representation of the intra-layer as well as inter-layer design of this model's architecture:

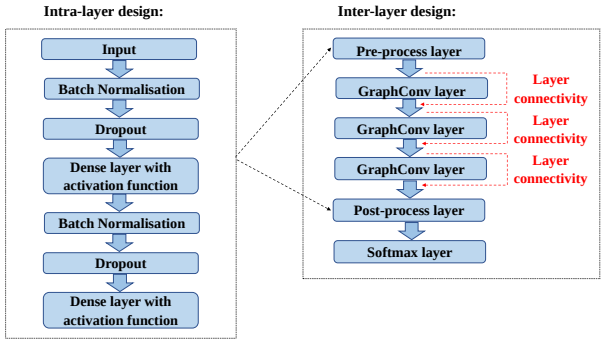

Figure 2. Intra and inter-layer designs of the initial GNN model

5.2.4. Theory

Let $X \in \mathbb{R}^{n \times m}$ be the node representation matrix[1] where n and m is the cardinality of the set N and M containing the nodes and basis vectors of the embedded input data[2] respectively. This can be represented as $n = |N|$ and $m = \dim(M)$.

Let $W_i \in \mathbb{R}^{m \times k_0}$ be the weight matrix of the initial feed forward network layer where k_i is the number of hidden units in the ith layer and let $b_i \in \mathbb{R}^k$ be the bias in the ith layer. The output of the first feed-forward network layer is written as

$$Q^{(i)} = \sigma(X \cdot W_i + b_i)$$

for some activation function σ (such as ReLU).

[1]Row r of the node representation matrix contains the feature vector for node r.

[2]The input data dimension is not necessarily the raw data dimension, but rather an embedded, lower dimensional vector.

In the aggregation stage, let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix such that

$$A_{i,j} = \begin{cases} 1 & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}$$

where $i \sim j$ represents the case when node i is directly connected to node j. Let $D \in \mathbb{R}^{n \times n}$ be the degree matrix such that

$$D_{i,i} = \sum_{k=1}^{n} A_{i,k}$$

which is the sum of the adjacency matrix rows. Compute the matrix

$$\hat{A} = D^{-1/2} \cdot A \cdot D^{-1/2}$$

which normalises the adjacency matrix by the number of adjacent nodes. Let $W^{\text{agg}} \in \mathbb{R}^{k^{\text{in}} \times k^{\text{out}}}$ be the weight[3] matrix for the aggregation operation and compute the output node representation as

$$Z = g(\hat{A} \cdot Q^{(1)} \circ Q^{(2)} \circ Q^{(3)} \cdot W^{\text{agg}})$$

for softmax activation function g. Thus the GCN layer can be represented by

$$Z = g(D^{-1/2} \cdot A \cdot D^{-1/2} \cdot \sigma(\sigma(\sigma(X \cdot W_1 + b_1) \cdot W_2 + b_2) \cdot W_3 + b_3) \cdot W^{\text{agg}}).$$

5.3. Graph Neural Network Model with two parallel branches

The graph neural network model with two parallel branches also utilises tools from the `Keras` library, and has an architecture which is identical to the one of the previous subsection in terms of its intra-layer design and in terms of the construction of the graph convolutional layer, but not in terms of its inter-layer design. Namely, instead of directly stacking the graph convolutional layers with skip connections, this neural network's architecture consists of two separate branches with two convolutional layers in each branch with skip connections, followed by a concatenation layer that concatenates the outputs of the two parallel branches. All other components of the inter-layer design such as the pre-processing, post-processing, and Softmax layers are identical to the ones of the initial graph neural network model. Moreover, the hyperparameters adopted in this model such as aggregation, combination and skip connection types are the ones identified as optimal in terms of classification accuracy during the hyperparameter selection phase of the initial graph neural network model.

The novelty of this graph neural network model as compared to the initial one lies on the adoption of the two parallel

[3]It is important to note that the weight matrix is a parameter to be optimised, and not an inherit structure of the graph.

branches within its inter-layer design. The rationale behind this choice is that the two parallel branches can capture different information from the node's neighbourhood and learn different trends and patterns associated with the aggregated messages of the node's neighbours. The idea is that when the output of the two parallel branches is concatenated, the model incorporates information obtained from both parallel branches and learning processes.

The flowchart in Figure (3) is a visual representation of the inter-layer design of this model's architecture:

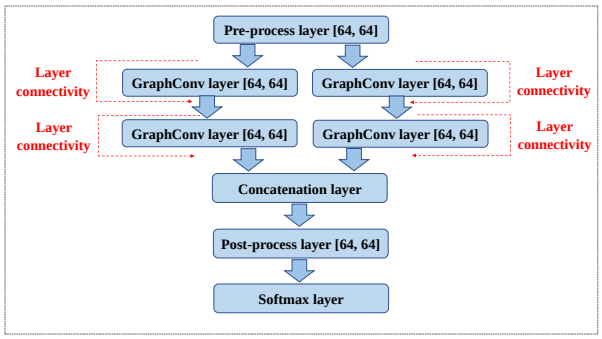

Figure 3. Inter-layer design of the two parallel branches GNN

5.3.1. Theory

Similar to the notation above, let X be the node representation matrix and $Q^{(1)} \in \mathbb{R}^{k_1 \times k_1}, Q^{(2)} \in \mathbb{R}^{k_1 \times k_1}, L^{(1)} \in \mathbb{R}^{n \times k_1}, L^{(2)} \in \mathbb{R}^{n \times k_1}$, be the GCN output for each branch respectively. Then

$$L^{(i)} = \sigma(D^{-1/2} \cdot A \cdot D^{-1/2} \cdot X \cdot W_i + b_i),$$

$$Q^{(i)} = \sigma(D^{-1/2} \cdot A \cdot D^{-1/2} \cdot L_i \cdot W_i + b_i),$$

where $W_i \in \mathbb{R}^{k_i \times k_i}$ are the trainable weight matrices for branch $i = 1, 2$. After computing the branches, combine the outputs such that

$$Q^* = f(Q^{(1)}, Q^{(2)}),$$

for some combination type function f. Then apply the fully connected layer to output the final node representations

$$Z = g(Q^* \cdot W^{\text{agg}})$$

for some trainable aggregation matrix $W^{\text{agg}} \in \mathbb{R}^{k^{\text{in}} \times k^{\text{out}}}$, softmax activation g, ReLU activation function σ, concatenation combination function f, and $k_1 = k^{\text{in}} = k^{\text{out}} = 64$.

5.4. Graph Neural Network Model with four parallel branches

Consider four parallel branches each with a different hidden unit dimension rather than the fixed [64, 64] size in the previous experiment. In each branch, a single layer is present in an attempt to learn features at different levels of abstraction.

5.4.1. Theory

Similar to the notation above, let X be the node representation matrix and $Q^{(1)} \in \mathbb{R}^{n \times k_1}, Q^{(2)} \in \mathbb{R}^{n \times k_2}, Q^{(3)} \in \mathbb{R}^{n \times k_3}$ and $Q^{(4)} \in \mathbb{R}^{n \times k_4}$ be the GCN output for each branch respectively. Then for branch $i = 1, ...4$

$$Q^{(i)} = \sigma(D^{-1/2} \cdot A \cdot D^{-1/2} \cdot X \cdot W_i + b_i),$$

where $W_i \in \mathbb{R}^{k_i \times k_j}$ are the trainable weight matrices for $i = 1, ..., 4$. In this implementation, we choose a square hidden layer representation such that $k_i = k_j$. After computing the branches, combine the outputs such that

$$Q^* = f(Q^{(1)}, Q^{(2)}, Q^{(3)}, Q^{(4)})$$

for some combination type function f. Then apply the fully connected layer to output the final node representations

$$Z = g(Q^* \cdot W^{\text{agg}})$$

for some trainable aggregation matrix $W^{\text{agg}} \in \mathbb{R}^{k^{\text{in}} \times k^{\text{out}}}$ and activation g. Each branch will have a unique hidden dimension size to capture different levels of abstraction, and the concatenation is an attempt to feed this specific configuration directly into the fully connected layer. Our configuration will use

$$\begin{aligned} k_1 &= 2 \\ k_2 &= 4 \\ k_3 &= 8 \\ k_4 &= 16 \\ k^{\text{in}} = k^{\text{out}} &= 32. \end{aligned}$$

with ReLU activation function σ, concatenation combination function f, and softmax activation function g.

The flowchart in Figure (4) illustrates the inter-layer design corresponding to the graph neural network with the four parallel branches:

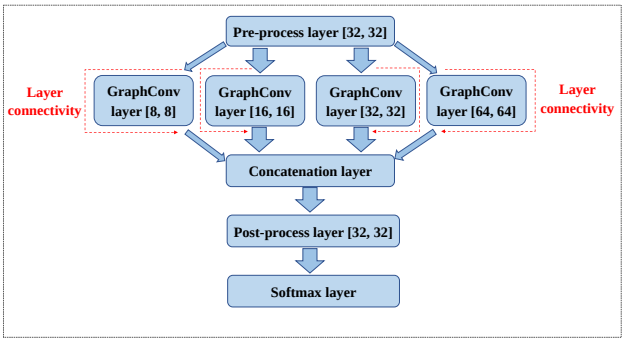

Figure 4. Inter-layer design of the four parallel branches GNN

5.5. GraphSage model

The GraphSage model is unique in its approach to node-wise classification, primarily due to its neighborhood aggregation technique. Unlike other models, GraphSage aggregates neighborhood information both globally and locally in an iterative process that collects information from nodes that are multiple hops away from the target feature vector. This iterative approach can be thought of as a single layer of message passing that results in a highly aggregated feature vector for each node. After this aggregation process, the resulting feature vectors are passed through a neural network to produce a final embedding that can be used for node classification. This approach is distinguished from other models, which may use multiple message passing layers instead. The process is denoted as *Inductive Representation Learning* and is an advantage of this approach, as stated in the relevant literature (Hamilton, 2017) (4). It enables the GraphSage to classify previously unseen nodes with high degrees of accuracy due to this comprehensive understanding of the "graph-space".

One aspect of the model used in this paper is the introduction of channels for each GraphSage layer. One can specify a separate number of channels with their own sets of weights. These weights relate to the portion of GraphSage where the feature vector is passed through a feed-forward neural network. The feature vector is passed through as many feed-forward networks to generate information for each of the channels specified. This allows us to learn specific features about the graph in a similar method to how one would capture information from different areas in an image with Conv2D layers and their channels.

The layout of this network is similar to the layout observed in previous models, as follows:

- GraphSage Convolutional Layer
- Dropout Layer
- Batch Normalisation
- Flatten Transformation
- Dense Output layer

Although simpler in terms of architecture to the parallel and single track networks, it is able to process information faster and has comparable accuracy as shown in the numerical results section.

6. Training Methods

The methods of training differ across the models proposed due to their different architectures.

- **Loss Function**: The multi-class classification model requires cross-entropy loss to be used for the 7 available classes in the dataset. Across all models, sparse multi-class cross-entropy loss is used due to the memory advantage to the hot-one encoding system over the conventional cross entropy loss.
- **Hyperparameter Selection process**: The hyper parameter selection process is an iterative and exhaustive search process for the different configurable options in the models. These options are compared in terms of their respective classification accuracy. The results of this process are presented in the next section.
- **Regularisation Techniques**: Whilst no direct kernel regularisation methods are used in the layers, dropout layers are added with the best-performing selection of probability in terms of classification accuracy. The related literature describes "that dropout rates of around 0.1 to 0.5 in the input and 0.5 in the hidden layers work well in practice". Due to the sparsity of connections and the size of the graph network, larger dropout rates are typically picked since a "lower dropout rate is beneficial for larger datasets" and vice versa (Srivastava et al., 2014, p. 1947) (9).
- **Optimisers and learning rates**: The default choice of optimiser used is the Adam optimiser due to its superior classification performance over the Stochastic Gradient Descent optimiser. Their respective results are presented in the next section. The superior performance of the Adam optimiser is also confirmed by relevant literature as "Adam is generally better than a naive SGD" (Jiaxuan You, Rex Ying, Jure Leskovec, 2021, p.7). (13). The learning rate chosen is 0.01 also due to its superior classification accuracy over the other choices attempted. This is further supported by the same source claiming that "learning rate of 0.01 is also favorable, also because it has significantly lower probability mass of being ranked 3rd" (Jiaxuan You, Rex Ying, Jure Leskovec, 2021, p.7) (13).
- **Early stopping**: During training, an early stopping callback is created to monitor the validation accuracy of the fitted model. A `patience` of 50 is set, whereby training is stopped if no improvement is made on validation accuracy after 50 epochs.
- **Validation split**: A validation split of 15% is chosen to evaluate the model during training. This implies that 15% of the training data is placed aside during training and used to evaluate the loss and accuracy at the end of each epoch.

7. Numerical results

The neural networks proposed are evaluated and compared in terms of their average test accuracy rate across 10 random 50%/50% train/test splits, as proposed by Khalid Salama (8). This metric is used throughout this section.

7.1. Feedforward Neural Network Model

The Feedforward Neural Network model used as a baseline produces an average test accuracy rate of 73.89%.

The pair of plots in the appendix, Figure (8) illustrates the loss and accuracy across the trained epochs for the training and validation data sets. Clearly, the baseline model performs quite strongly in terms of predictive accuracy on the training dataset, but not well on the validation dataset. This demonstrates the weakness of the baseline model to generalise on the validation set, suggesting the presence of overfitting.

7.2. Initial Graph Neural Network Model

In order to achieve meaningful comparisons between the model's predictive performance under different configurations, each time a particular setting's choice is considered, all other settings are held constant at their baseline choice. This baseline choice corresponds to the top cell of each setting in the tables below.

The results of the initial graph neural network model are summarised in Table (1), Table (2) and Table (3) associated with the intra-layer, inter-layer and learning configurations respectively.

Setting	Choice	Average Test Accuracy
Activation function in last layer of intra-layer design	**ReLU**	**81.51%**
	GeLU	81.35%
	PReLU	80.06%
Batch Normalisation	**True**	**80.58%**
	False	77.13%
Dropout Rate	**0.4**	**81.92%**
	0.5	80.87%
	0.6	70.56%
Number of units in dense layers	[32,32]	81.42%
	[8, 8]	64.62%
	[32, 64]	81.89%
	[64, 64]	**82.71%**
	[128, 128]	82.39%

Table 1. Results on the intra-layer design

A Dropout rate of 0.0 is also attempted yielding an average test accuracy of 54.43%. This value is considerably lower than the respective ones achieved under all other dropout rate choices attempted above. Thus, it can be seen that a zero dropout rate is not a good choice as it leads to a model with lower predictive accuracy. Moreover, a dropout rate of 0.2 yields an average test accuracy of 77.59%, which is larger than that achieved with zero dropout rate but still lower than those achieved with 0.4 and 0.5 dropout rates.

Setting	Choice	Average Test Accuracy
Aggregation Type	**Sum**	**81.33%**
	Mean	80.28%
	Min	11.14%
	Max	10.81%
	Prod	71.62%
Combination Type	**Concatenation**	**80.82%**
	Addition	77.13%
	GRU	73.24%
Number of Graph Convolutional layers	2	80.84%
	3	**81.29%**
	4	80.86%
Skip Connection Type	Sum	81.30%
	Concatenation	**83.45%**

Table 2. Results on the inter-layer design

As far as the results on the inter-layer design are concerned, key findings of "Design space for Graph Neural Networks" (13) are validated, namely that the optimal aggregation type is summation, while the optimal skip connection type is concatenation.

Setting	Choice	Average Test Accuracy
Batch Size	**256**	**83.09%**
	512	79.30%
	1024	74.22%
Learning Rate	**0.01**	**82.47%**
	0.05	81.80%
	0.1	30.40%
Optimiser	**Adam**	**83.04%**
	SGD	30.14%

Table 3. Results on learning configurations

The optimal optimiser is Adam. Combined with a learning rate of 0.01, this serves as the best-performing combination.

7.3. Graph Neural Network Model with two parallel branches

The best performing hyperparameters identified in the initial GNN model (except from the number of graph convolutional layers) are adopted to compile and train the GNN model with the two parallel branches. The best performing hyperparameters are the ones in bold in the three tables of the previous section. The average test accuracy achieved for this model is 83.07%, which is the third highest achieved so far.

7.4. Graph Neural Network Model with four parallel branches

The GNN model with four parallel branches achieves an average test accuracy of 82.83%, which is the lowest predictive accuracy within the class of GNNs tested. It should also be noted that this architecture requires only 86000 trainable parameters which is amongst the lowest of models tested.

7.5. GraphSage

The results from the GraphSage node-classification task allowed us to observe the performance increases comparatively to other methods. As discussed previously, the global and local method of aggregation helps each node capture larger-scaled relationships between papers. Table (4) shows the observed differences in performance on the benchmark dataset. The highlighted configurations are combined with the parameters in previous sections. This includes using the `ReLU` activation function and a similar optimiser layout.

What is observable in Table (4) in comparison to other methods is the lack of accuracy. By only reaching around 80% accuracy at most it seems this method of node-classification is outperformed by the previous models. As mentioned prior, the layout of the network is much more simple with a lower training time in comparison to the initial `Keras` implementation of GNN.

Utilising the most optimal parameters shown above and obtaining a final testing accuracy provides us with the accuracy of around 80% as shown in Table (5). One interesting discovery is the effect of the aggregation methods on the accuracy of the model. Attention based aggregation excels in comparison to mean and sum in most cases. As stated in the relevant literature though "the effectiveness of attention-based aggregation also depends on the properties of the graph being analyzed" (Chen et al., 2020, p. 838) (2) which conveys to the authors that perhaps the Cora sparsity and unweighted properties make it suitable for such a type over stated attention methods.

Setting	Choice	Average Test Accuracy
Aggregation type	Sum	75.2%
	Mean	**78.3%**
	Min	75.4%
	Max	76.7%
	Prod	74.7%
Batch Normalisation	**True**	**79.1%**
	False	74.7%
Channel Size	**4**	**76.3%**
	8	74.9%
	12	74.3%

Table 4. Results on GraphSage Learning Configurations

Model Name	Final Accuracy
Baseline FFN	73.89%
Simple GNN	**83.45%**
GNN, Two Parallel Branches	83.07%
GNN, Four Parallel Branches	82.83%
GraphSage	80.1%

Table 5. A comparison of final model accuracy with best parameter selection

8. Discussion

The authors proposed that graph neural networks should improve classification accuracy using additional information about the relationship between nodes, compared to regular neural networks where samples are treated as independent to each other. The results indicate that graph neural networks provide a greater accuracy as compared to a regular feed-forward neural network when applied to the Cora dataset. Furthermore, our research has indicated that computing the layers does not improve classification accuracy when the parallel branches have a constant hidden layer units within the inter-layer design. Furthermore, the model with four parallel branches each with a single layer containing unique hidden units proved to underperform all other tested GNN architectures. This suggests that GNNs sufficiently capture complex relationships through a small number of parameter, deeming branching and varying hidden unit sizes to be detrimental to performance.

9. Conclusion

The paper has highlights the advantage of using graph neural networks as an alternative to regular feed forward neural networks. Furthermore, the paper introduces a variant to the inter-layer design which incorporate parallel layers. All three GNN architectures outperformed the baseline FFN, highlighting the advantages of using graph neural networks in classification. The use of parallel layers did not improve accuracy compared to the regular GNN architecture, suggesting that branching with or without unique hidden unit sizes does not improve the model. However, further research into the effect of more parallel layers should be explored as well as techniques such as pooling layers within the branches. The paper implements graph neural networks on a *small* dataset as the goal of the paper is to compare the performance of different architectures. Further study into the performance of these architectures to larger datasets is a natural progression from this paper.

References

[1] Michael Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[2] Tianlong Chen, Yifan Feng, and Changjie Fan. Which gnn architecture suits your graph? In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 837–847. ACM, 2020.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[7] Nigam K. Rennie J. et al McCallum, A.K. Automating the construction of internet portals with machine learning. Information Retrieval 3, 2000.

[8] Khalid Salama. Node classification with graph neural networks. Webpage, 2021. Accessed on: April 25, 2023.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Understanding dropout. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

[11] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[13] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks, 2021.

[14] Jie Zhou, Ganqu Cui, Zhiyuan Zhang, Chengyang Yang, Zhiyuan Liu, and Maosong Sun. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

A. Appendix

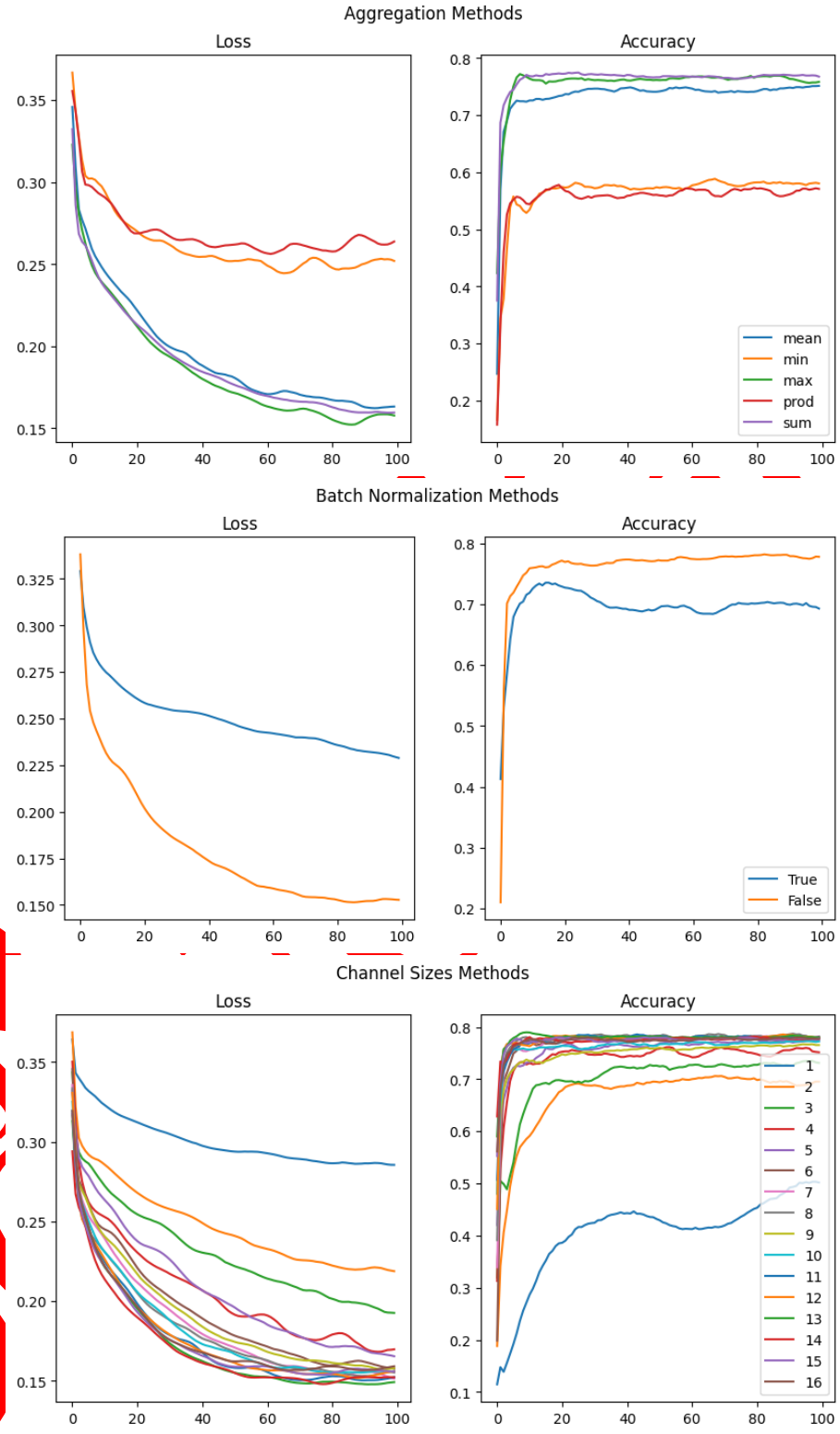

Figure 5. Figure Of Loss and Aggregation Curves of the GraphSage Model

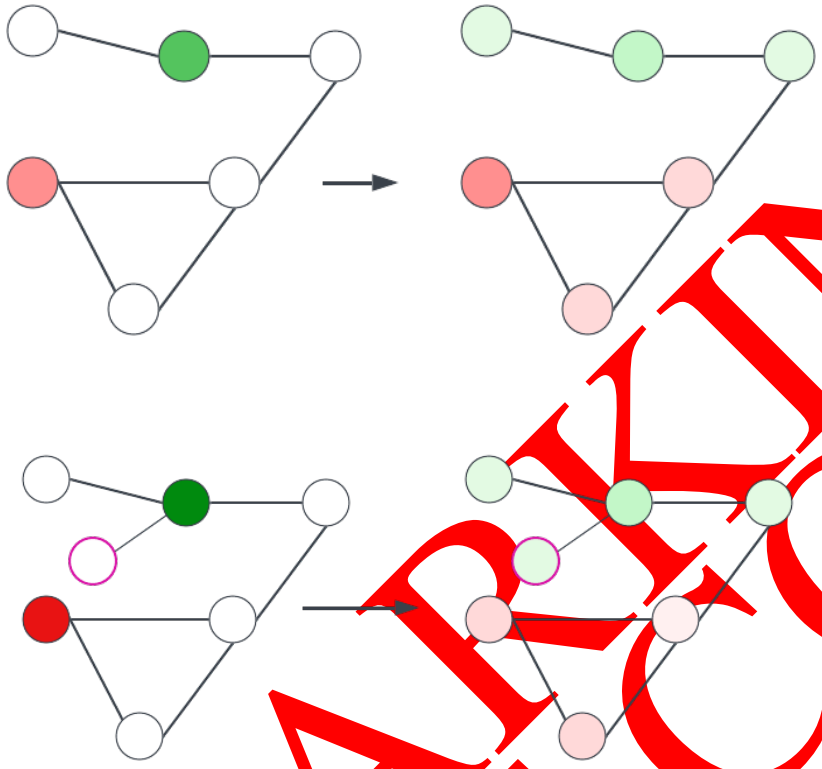

Figure 6. The message passing Phase (**Top**), New Node classification shown with the magenta node (**Bottom**)

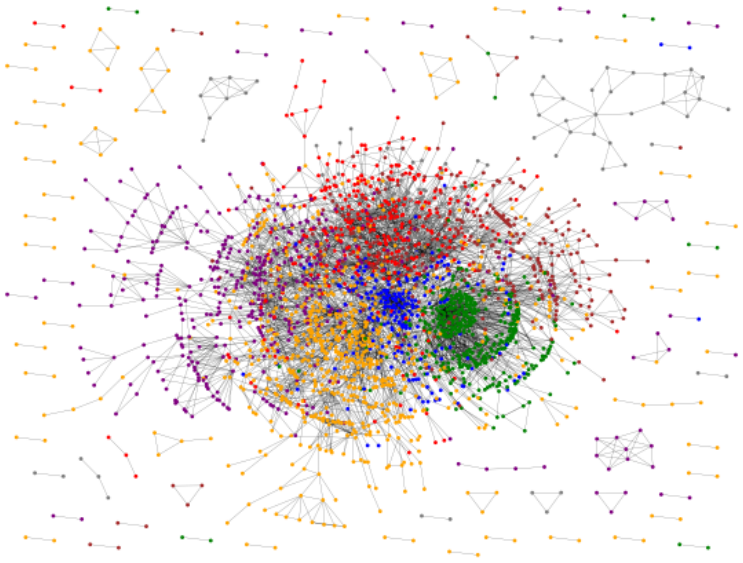

Figure 7. Citations Graph Visualised using the *GraphViz* **neato** algorithm

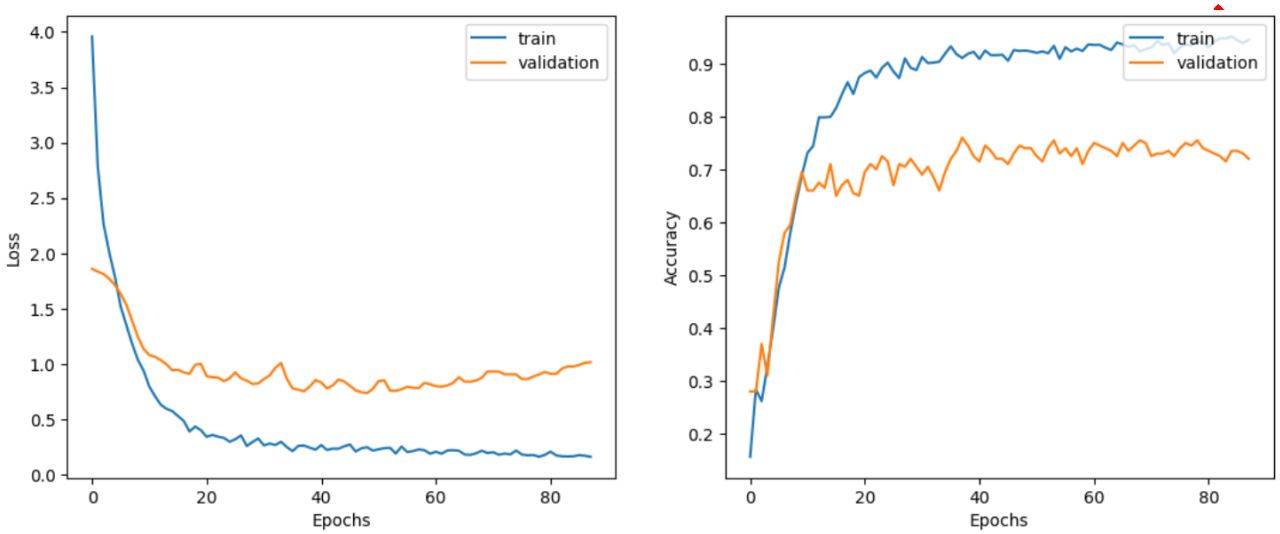

Figure 8. Loss and accuracy across epochs on training and validation data sets for baseline model

Statement about individual contributions

All group members contributed an equal amount of work to the project overall, demonstrating strong collaboration and teamwork. The paper is a combined effort from the three team members: Candidate 43577, Candidate 45128, Candidate 49640. Candidate 43577 worked on the mathematical notation throughout the paper, as well as the model architecture information. They also helped implement every model except for GraphSage and played a crucial role in drafting the written report. Candidate 45128 focused on maintaining the consistency of testing across each model and assisted in the implementation of all models. Candidate 45128's testing code was used to build the numerical results section, and their guidance contributed significantly to the paper's successful completion. Candidate 49640 was responsible for implementing the GraphSage model and creating the majority of visualizations seen in the paper. They also contributed to the introduction, related work, and theoretical framework sections with support from the other team members. Throughout the project, the team members communicated effectively, provided feedback, and supported each other in overcoming challenges. In conclusion, the equal and significant contributions from all members were vital to the paper's progress and eventual completion.