Chair of Computer Science 6 (i6)
Learning on Graphs (LoG)

Exploration of the influence of graph parameters on the generalization error of Graph Neural Networks

Bachelor Thesis from

Wensheng Zhang

Supervisor
Chendi Qian, M. Sc.
1.Examiner
Prof. Christopher Morris
2.Examiner
Prof. Michael Schaub

---

29. Januar 2025

Contents

Acknowledgements

First, I would like to thank Prof. Christopher Morris and Prof. Michael Schaub for the opportunity of my thesis. Secondly, I would like to thank my supervisor Chendi Qian for giving me the great support to do this thesis, and also for his help in understanding the content of the thesis at the beginning, and for his help in the content and code throughout my entire bachelor thesis time.

List of Figures

List of Tables

Abstract

The goal of this bachelor thesis is to understand empirically how graph parameters/characteristics influence the generalization error of Graph Neural Networks (GNNs).

1 Introduction

Graph learning became a hot topic in the field of machine learning in recent years. Graph Neural Networks (GNNs) [GMS05], also known as deep learning on graphs, are a class of neural networks that can operate on graph-structured data, such as social networks [EK+10], biological networks [BO04], and images [SK17]. GNNs use the structure of a graph to perform "message passing", allowing information to propagate across nodes and edges. The key idea is to iteratively aggregate and update node features by exchanging information with local neighboring nodes, which allows GNNs to capture the structural information of the graph. This information is then used to make predictions. GNNs have been demonstrated to be effective in a wide range of tasks, such as node classification [HYL17], edge prediction [Mor+21], and graph classification [Gil+17]. Popular GNNs in recent years include Graph Convolutional Networks (GCN) [KW16], Graph Attention Networks (GAT) [Vel+20], and Message Passing Neural Networks (MPNN) [Gil+17].

However, the behavior of GNNs is not well understood. Particularly, the relationship between graph parameters/characteristics and the empirical generalization error of GNNs is not clear. The goal of this bachelor thesis is to understand empirically how graph parameters/characteristics impact the generalization error of GNNs.

To reach this goal, I conduct two sets of experiments. First, I calculate the graph parameters for a set of datasets: Each graph has its own unique properties, such as the average degree. In graph theory, *graph parameter* is used to quantify them. And a dataset consists of multiple graphs, each with its graph parameter. In the case, I calculate the average value of our desired graph parameters over all graphs in the dataset. In this thesis, 50 datasets from the TUDataset [Mor+20] is used for the calculation. Second, I train different GNN models on each dataset and find that their generalization abilities varied. I show that there is a certain relationship between generalization ability and different graph parameters.

The thesis is organized as follows. In Chapter 2, I provide background information on graph neural networks and generalization. In Chapter 3, I describe the methodology used to conduct the experiments. In Chapter 4, I present the experimental results and discuss the results and their implications. Finally, in Chapter 5, I conclude the thesis and discuss future work.

1.1 Related work

In the following section, I will discuss the related work on graph neural networks and generalization.

1.1.1 Graph Neural Networks

Graph neural networks were initially proposed by Gori et al. [GMS05] and Scarselli et al. [Sca+08] as a variant of recurrent neural networks and demonstrated its generalization capabilities. The work of Kipf and Welling [KW16] introduced the Graph Convolutional Networks (GCN) model, which received widespread attention from the research community. Velickovic et al. [Vel+20] proposed the Graph Attention Networks (GAT) model, which uses attention mechanisms to aggregate information from neighboring nodes. Brody et al. [BAY21] proposed the GATv2 model, which improves the performance of the GAT model by introducing residual connections and multi-head attention. Gilmer et al. [Gil+17] introduced the Message Passing Neural Networks (MPNN) model, which demonstrated the significant results on the molecular property prediction benchmark;

1.1.2 Generalization

While significant progress has been made in understanding the generalization properties of feed-forward neural networks (FNNs) and recurrent neural networks (RNNs), the generalization abilities of graph neural networks (GNNs) remain less explored due to their of irregular graph structures.

Golowich et al. [GRS18] showed the generalization abilities of Feed-forward neural networks (FNNs) with Rademacher complexity. The work of Zhang et al. [ZLD18] used SVD-based parameterization to improve the generalization by effectively addressing gradient issues.

In terms of Recurrent neural networks(RNNs), Chen et al. [CLZ19] studied the generalization properties of vanilla RNNs as well as their variants. Z. Allen-Zhu et al. [AL19] demonstrates that vanilla SGD enables RNNs to efficiently learn concept classes and overcomes prior exponential generalization bounds.

But unlike the FNNs and RNNs, GNNs deal with irregular data structures. The related work on the generalization of GNNs is limited. Scarselli et al. [STH18] explored the generalization ability of GNNs by using the Vapnik-Chervonenkis (VC) dimension and further indicated that the generalization ability of such models improve as the number of connected nodes. After that, Garg et al. [GJJ20] studied the generalization ability of GNNs via Rademacher bounds. In comparison to the work of Scarselli et al., Garg et al. provided a tighter bound on the generalization error of GNNs.

Nevertheless, Morris et al. [Mor+24] pointed out that the generalization ability of GNNs is still not well understood. The bounds established in these studies typically rely solely on general graph parameters, such as the maximum degree or total number of nodes, often overlooking more expressive graph parameters.

2 Background on learning on graphs

The following sections provide an overview of the concepts and methods related to graph theory, machine learning, and graph neural networks.

2.1 Notion

At first, we need to define some terms with regard to the graph. In graph theory, we assume that $G = (V, E)$ is an undirected graph, where V is the set of nodes and $E \subseteq \{\{v, w\} \mid v, w \in V, v \neq w\}$ is the set of edges. We use $\mathbb{G}$ to represent the set of all possible graphs. The neighborhood of a node v is denoted as $N(v)$, defined as $N(v) = \{w \mid \{v, w\} \in E\}$. The degree of a node v, denoted as $d(v)$, is the number of neighbors of v, that is, $d(v) = |N(v)|$. In addition to the nodes and edges that comprise the graph, we can attach a more complex set of data to each node. A feature vector $x_n \in \mathbb{R}^k$, is attached to a node v_n, and in some cases, edge feature vectors are also included. These vectors can be used to characterize graphs or nodes in graphs. The adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ is a matrix that represents the connections between nodes in the graph. The element a_{ij} of the adjacency matrix is 1 if there is an edge between node v_i and node v_j, and 0 otherwise.

2.2 Machine learning

In the following section, we provide an overview of the concepts and methods related to machine learning.

2.2.1 Supervised learning

Machine learning can be roughly divided into three categories: supervised learning, unsupervised learning, and reinforcement learning [BN06].

Supervised learning is a type of machine learning that involves training a model on a labeled dataset. The model is trained to learn the relationship between the input features and the output labels. The goal is to learn a mapping from the input features to the output labels so that the model can make predictions on new, unseen data. One common task in supervised learning is classification, where the goal is to assign each input vector to one of a finite number of classes (categories). In the graph learning context, the predicted output can be a label for each node in the graph, a label for the entire graph, or a label for each edge in the graph, e.g. node classification, graph classification, and link prediction [Vel23].

2.2.2 Feed-forward neural networks

Feed-forward neural networks, also known as multilayer perceptrons (MLPs), are a type of neural network that consists of multiple layers of neurons. The neurons in each layer are connected to the neurons in the next layer, and the output of each neuron is computed as a weighted sum of the inputs followed by a non-linear activation function. The output of the network is computed by passing the input through the network and computing the output of the final layer. The weights of the network are learned by using an optimization algorithm such as gradient descent. Feed-forward neural networks are a powerful tool for learning complex patterns in data and have been successfully applied to a wide range of tasks, such as image recognition, speech recognition, and natural language processing.

The function of a feed-forward neural network can be written as

$$y = f(W^{(L)} \cdot f(W^{(L-1)} \cdot \ldots \cdot f(W^{(1)} \cdot x + b^{(1)}) + b^{(2)}) + \ldots + b^{(L)})$$

where x is the input vector, $W^{(i)}$ and $b^{(i)}$ are the weights and biases of the i-th layer, and f is the activation function. The output y is the predicted output of the network. L is denoted as the depth of the network.

2.2.3 Loss function

The loss function is a function that measures the difference between the predicted output of a model and the true output. The goal of training a machine learning model is to minimize the loss function, which is done by adjusting the parameters of the model using an optimization algorithm such as gradient descent. The choice of loss function depends on the task at hand. For classification tasks, the cross-entropy loss function is commonly used, while for regression tasks, the mean squared error loss function is often used.

The cross-entropy loss function is defined as

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

where y is the true output, $\hat{y}$ is the predicted output, and i is the index of the classes. The cross-entropy loss function is used to measure the difference between the true output and the predicted output of a classification model.

2.2.4 Activation function

An activation function is a non-linear function that is applied to the output of a neuron in a neural network. The activation function introduces non-linearity into the network, allowing it to learn complex patterns in the data. There are many activation functions that can be used in neural networks, such as the sigmoid function, the tanh function, and the ReLU function. The ReLU function is one of the most commonly used activation functions in deep learning because it is simple and computationally efficient. The ReLU function is defined as

$$\mathrm{ReLU}(\cdot) = \max(0, \cdot)$$

The ReLU function is used to introduce non-linearity into the network and has been shown to be effective in training deep neural networks.

2.2.5 Gradient Descent

Gradient descent is an optimization algorithm used to minimize a function by iteratively moving in the direction of the steepest decrease of the function. The algorithm works by computing the gradient of the function at a given point and then updating the point in the direction of the negative gradient. The update rule is given by

$$W^{(t+1)} = W^{(t)} - \alpha \nabla f(W^{(t)})$$

where $W^{(t)}$ is the parameter vector at iteration t, α is the learning rate, and $\nabla f(W^{(t)})$ is the gradient of the function f with respect to the parameters $W^{(t)}$. The learning rate α is a hyperparameter that controls the step size of the update. Gradient descent is a popular optimization algorithm used in machine learning to train neural networks and other models.

2.2.6 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a variant of gradient descent that uses a random sample of the training data to compute the gradient at each iteration. The algorithm works by randomly selecting a mini-batch of data points from the training dataset, computing the gradient of the loss function with respect to the mini-batch, and then updating the parameters based on the gradient. The update rule is given by

$$W^{(t+1)} = W^{(t)} - \alpha \nabla f(W^{(t)}, x^{(t)})$$

where $x^{(t)}$ is a mini-batch of data points from the training dataset. Stochastic gradient descent is more computationally efficient than standard gradient descent and can be used to train large-scale machine learning models.

2.2.7 Early Stopping

Early stopping is a technique used to prevent overfitting in machine learning models. The idea is to monitor the performance of the model on a validation dataset during training and stop training when the performance starts to degrade. This is done by keeping track

of the validation loss and stopping training when the validation loss stops decreasing or starts to increase. Early stopping is used to prevent the model from memorizing the training data and to obtain a more generalizable model.

2.2.8 Cross-validation

Cross-validation is a technique used to evaluate the performance of a machine learning model. The idea is to split the dataset into multiple subsets, or folds, and train the model on one subset while evaluating it on the remaining subsets. This process is repeated multiple times, with each subset used as the test set once. The performance of the model is then averaged over all the folds to obtain an estimate of the model's generalization performance. Cross-validation is used to prevent overfitting and to obtain a more accurate estimate of the model's performance.

2.2.9 Hyperparameter tuning

Hyperparameter tuning is the process of selecting the optimal hyperparameters for a machine learning model. Hyperparameters are parameters that are set before the training process begins and control the behavior of the model. Examples of hyperparameters include the learning rate, batch size, number of hidden layers, and number of epochs. Hyperparameter tuning is an important step in the machine learning pipeline and can have a significant impact on the performance of the model. There are many techniques for hyperparameter tuning, such as grid search, random search, and Bayesian optimization. In this thesis, Bayesian optimization is used to tune the hyperparameters of the models.

Bayesian optimization is more efficient than grid search and random search because it uses a probabilistic model to guide the search for the optimal hyperparameters. The idea is to model the objective function (e.g. the validation accuracy of the model) as a Gaussian process and use the model to select the next set of hyperparameters to evaluate. This process is repeated until the optimal hyperparameters are found.

2.2.10 Optimization algorithm

An optimization algorithm is an algorithm used to minimize a function by iteratively updating the parameters of the model. The goal is to find the set of parameters that minimizes the loss function. Gradient descent is a common optimization algorithm used in machine learning, but there are many variants of gradient descent that are designed to improve convergence speed and stability. Some popular optimization algorithms include Adam [Kin14], RMSprop [Gra13], and Adagrad [DHS11]. These algorithms use different strategies to update the parameters of the model and can be more effective than standard gradient descent in certain situations.

2.2.11 Generalization

A key concept in evaluating model performance or generalization ability in such task is the empirical generalization error E_{gen}. This is a measure of how well the model generalizes to unseen data, which can be calculated as the difference between the training accuracy and the test accuracy. It is defined as

$$E_{gen} = Acc_{train} - Acc_{test}$$

where Acc_{train} is the accuracy of the model on the training dataset, and Acc_{test} is the accuracy of the model on the test dataset. The accuracy Acc on a dataset is defined as the proportion of correctly classified graphs, that is, $Acc = \frac{\text{Number of correctly classified graphs}}{\text{Total number of data points}}$.

The empirical generalization error is also known as the generalization gap [GBC16]. Typically, the learning on the training dataset and test dataset behaves differently. The model may perform well on the training dataset but poorly on the test dataset. By observing the generalization error, we can decide on a stopping criterion for the training process. And the generalization error can be used to determine whether the model is overfitting or underfitting. Furthermore, the generalization error can be influenced by many factors, such as the model architecture, the hyperparameters, and the data properties in the dataset (e.g. Graph parameter).

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural networks that can operate on graph-structured data. GNNs use the structure of a graph to perform "message passing", allowing information to propagate across nodes and edges. The key idea is to iteratively aggregate and update node features by exchanging information with local neighboring nodes, which allows GNNs to capture the structural information of the graph. This information is then used to make predictions. GNNs have been demonstrated to be effective in a wide range of tasks, such as node classification, edge prediction, and graph classification. Popular GNNs in recent years include Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and Message Passing Neural Networks (MPNN).

In the following sections, we provide an overview of the methods related to this thesis.

2.3.1 Graph Convolutional Networks (GCN)

Graph Convolutional Networks (GCNs) are a class of neural networks that can operate on graph-structured data. The GCN model $f(X, A)$, as proposed by Kipf and Welling [KW16], is defined as

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

where $H^{(l)} \in \mathbb{R}^{N\times D}$ is the node feature matrix at layer l; $H^{(0)} = X$. $\tilde{A} = A + I$ is the adjacency matrix of the graph with self-connections added, $\tilde{D}$ is the degree matrix of $\tilde{A}$, $W^{(l)}$ is the weight matrix of the layer, and σ is the activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. The GCN layer aggregates information from neighboring nodes and updates the node features by applying a linear transformation followed by a non-linear activation function. The output of the GCN layer is the node feature matrix at the next layer. The GCN layer can be stacked to create deep GCN models, which have been shown to be effective in a wide range of tasks, such as node classification, edge prediction, and graph classification.

2.3.2 Simplified Graph Convolution (SGC)

Simplified Graph Convolution (SGC) [Wu+19] is a simplified version of the GCN model that removes the non-linear activation function and the normalization of the adjacency matrix. The SGC model is defined as

$$X' = (\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})^K XW$$

where X is the node feature matrix, W is the weight matrix, $\tilde{A}$ is the adjacency matrix with self-connections added, $\tilde{D}$ is the degree matrix of $\tilde{A}$, and K is the number of layers.

2.3.3 Graph Attention Networks (GAT)

Graph Attention Networks (GAT) are a class of neural networks that use attention mechanisms to aggregate information from neighboring nodes. The GAT model updates the feature representation of a node by assigning different importance scores, known as attention coefficients, to its neighbors. The updated node feature $\mathbf{h}_i^{(l+1)}$ at layer $l+1$ is given by:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right)$$

where $\mathbf{h}_i^{(l)}$ is the feature vector of node i at layer l, $\mathbf{W}^{(l)}$ is a learnable weight matrix, σ is an activation function, and $\alpha_{ij}^{(l)}$ is the attention coefficient between nodes i and j. These coefficients are computed as:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l)\top}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l)} \| \mathbf{W}^{(l)}\mathbf{h}_j^{(l)}]\right)\right)}{\sum_{k\in\mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l)\top}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l)} \| \mathbf{W}^{(l)}\mathbf{h}_k^{(l)}]\right)\right)}$$

where $\|$ denotes the concatenation of node features, and $\mathbf{a}^{(l)}$ is a shared attention vector. The softmax function ensures the attention coefficients sum to one for a node's neighbors.

GAT is particularly effective for tasks where different neighbors contribute unequally to a node's representation. By using attention mechanisms, GAT dynamically weighs neighbor contributions, enabling more expressive feature aggregation compared to earlier methods like GCNs.

2.3.4 Graph Attention Networks v2 (GATv2)

Graph Attention Networks v2 (GATv2) [BAY21] is an improved version of the GAT model that introduces residual connections and multi-head attention. The GATv2 model is defined as

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{h=1}^{H}\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l,h)}\mathbf{W}^{(l,h)}\mathbf{h}_j^{(l)}\right) + \mathbf{h}_i^{(l)}$$

where H is the number of attention heads, $\alpha_{ij}^{(l,h)}$ is the attention coefficient between nodes i and j for head h, and $\mathbf{W}^{(l,h)}$ is the weight matrix for head h. The attention coefficients are computed as

$$\alpha_{ij}^{(l,h)} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l,h)^\top}[\mathbf{W}^{(l,h)}\mathbf{h}_i^{(l)} \| \mathbf{W}^{(l,h)}\mathbf{h}_j^{(l)}]\right)\right)}{\sum_{k\in\mathcal{N}(i)}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^{(l,h)^\top}[\mathbf{W}^{(l,h)}\mathbf{h}_i^{(l)} \| \mathbf{W}^{(l,h)}\mathbf{h}_k^{(l)}]\right)\right)}$$

The GATv2 model uses multi-head attention to allow the model to attend to different parts of the input space. The model also introduces residual connections to improve the flow of information through the network. The GATv2 model has been shown to outperform the original GAT model on a wide range of tasks.

2.3.5 Message Passing Neural Networks (MPNN)

Message Passing Neural Networks (MPNNs) [Gil+17] have been the leading model for graph learning. The MPNN model operates by iteratively passing messages between nodes and updating their representations. At each iteration t, the message vector $m_i^{(t+1)}$ for node i is computed as:

$$m_i^{(t+1)} = \sum_{j \in N(i)} M^{(t)}(h_i^{(t)}, h_j^{(t)}, e_{ij})$$

where $m_i^{(t)}$ is the message vector of node i at iteration t, $M^{(t)}$ is a learnable message function (typically implemented as a neural network, such as an MLP), $h_i^{(t)}$ is the node feature vector of node i at iteration t, $h_j^{(t)}$ is the node feature vector of node j at iteration t, e_{ij} is the edge feature vector between node i and node j, and $N(i)$ is the neighborhood of node i. The message function $M^{(t)}$ is a neural network that takes as input the node feature vectors of node i and node j, and the edge feature vector between them, and outputs the message vector $m_i^{(t)}$.

2.4 Graph parameters

Now we can introduce the concept of a graph parameter. A graph parameter is a function $f : \mathbb{G} \to \mathbb{R}^d$ for $d \in \mathbb{N} > 0$, which maps a graph to a d-dimensional vector over the reals. Furthermore, such functions need to be permutation invariant, i.e., $f(G) = f(\pi(G))$ for any permutation π of the nodes of G, where a permutation is a function $\pi : V \to V$ and $\pi(G)$ denotes the graph obtained by permuting the nodes of G according to π. A graph parameter allows us to quantify certain properties of a graph, from the basic structural numbers (number of nodes, number of edges, average degree, etc.) to distance properties (average shortest path length, diameter, etc.) to more complex properties (graph clustering coefficient, average number of coloring in the 1-WL algorithm, etc.). This thesis will focus on seven graph parameters, with definitions provided in section 2.4.

3 Experimental study

3.1 Experimental setup

The following section outlines the planned methods and materials to be employed in the experiment, including the training framework, the datasets, the models, and the graph parameters.

3.2 Experimental procedures

The Experimental procedures are as follows:

1. Obtain the dataset using the setup outlined in Section 3.2.2.
2. Tune the hyperparameters using Bayesian optimization, as detailed in Section 3.2.3.
3. Train the models using the optimized hyperparameters and compute their generalization errors.
4. In the meanwile, calculate the graph parameters for each dataset, as detailed in Section 3.3.
5. Finally, evaluate the impact of the graph parameters on the generalization error of the models by calculating the correlation between the graph parameters and the generalization error.

3.2.1 Training Framework

The training framework is based on PyTorch Geometric [FL19]. PyTorch Geometric is a library for deep learning on graphs and other irregular structures. It consists of various methods and utilities to ease the implementation of GNNs.

3.2.2 Datasets

The dataset used for the classification is TUDataset [Mor+20]. TUDataset is often used for the GNN evaluation. It consists of data from different domains, including small molecules, bioinformatics, social networks and synthetic. Since the size of some datasets is quite small (less than 500 data points/graphs), I use 10-fold cross validation in the training process, in order to fully utilize the data. A dataset is split into 1:1:8, where one of the folds is treated as the test dataset and another is treated as the validation dataset. The remaining folds are used for training. The training is repeated 10 times, each time with a different test fold. The average of the generalization error over the 10 runs is calculated as the final generalization error of the model.

For the dataset with large size, the first 4000 data points are selected. The remaining data points are discarded. The reason is that some datasets are too large to be trained in a reasonable amount of time.

I list information about the 50 datasets used in the experiment in the appendix **??**.

3.2.3 Hyperparameters tuning

Before training the models, for each pair of dataset and model, the hyperparameters of the model are tuned. In the experiment, 5-fold cross-validation is used to tune the hyperparameters instead of 10-fold cross validation in the final experiment. The reason is that the 10-fold cross-validation is computationally expensive and time-consuming.

The hyperparameters include the learning rate, batch size, hidden dimension, number of epochs, patience in early stopping, number of hidden layers, type of normalization, and patience of plateau scheduler. Additionally, for the GATv2 model, the number of heads, the dropout rate, and residual are also tuned.

The hyperparameters are tuned using the Bayesian optimization method[Fra18]. The hyperparameters that result in the highest validation accuracy are selected as the optimal hyperparameters for the model.

There are five models and 57 datasets, resulting in 285 pairs of dataset and model. For each pair, the hyperparameters are tuned using the Bayesian optimization method. The hyperparameters are tuned using the validation dataset. The validation dataset is a subset of the training dataset that is used to tune the hyperparameters of the model. The hyperparameters that result in the highest validation accuracy are selected as the optimal hyperparameters for the model.

3.2.4 Models

In the experiment, I plan to use different GNN layers in the model, including Graph Convolutional Networks(GCN) [KW16], Graph Attention Networks(GAT) [Vel+20], Graph Attention Networks(GATv2) [BAY21], Simplified Graph Convolution(SGC) [Wu+19], and Message Passing Neural Networks(MPNN) [Gil+17]. The GCN model is shown in the appendix. The GCN model consists of four GCN layers, each followed by a ReLU activation function. The output of the last GCN layer is passed through a global mean pooling layer, followed by two fully connected layers.

3.2.5 Experimental details

The Adam optimizer [Kin14] is considered to be employed in the model. To prevent overfitting and to optimize the use of time, the early stopping is employed. The loss function is defined as the cross-entropy loss function. The hyperparameters of the model are considered to be tuned. The hyperparameters include the learning rate, batch size, hidden dimension, number of epochs, and patience in early stopping. The generalization error is calculated as the difference between the training accuracy and the test accuracy.

3.3 Graph Parameters Under Investigation

As graph parameters defined abstractly in the section 2.4, I give here the concrete definition of the graph parameters that are used in the experiment. The graph parameters are calculated for each dataset and used as input features for the model. The graph parameters are calculated using the NetworkX library [HSS08].

Average degree The average degree of the graph d_{avg} is defined as the average of the degrees of all nodes in the graph, that is

$$d_{avg} = \frac{1}{|V|} \sum_{v \in V} d(v)$$

Average shortest path length The average shortest path length of the graph a is defined as the average of the shortest paths between all pairs of nodes in the graph. The shortest path $d(s,t)$ between two nodes v and w is the minimum number of edges that need to be traversed to go from v to w. The formal definition is

$$a = \frac{1}{|V|(|V|-1)} \sum_{v \in V} \sum_{w \in V, w \neq v} d(v,w)$$

Graph diameter The graph diameter d is the maximum of the shortest paths between all pairs of nodes in the graph, that is,

$$d = \max_{v \in V} \max_{w \in V, w \neq v} d(v,w)$$

Graph density The graph density p is the ratio of the number of edges in the graph to the number of the maximum possible edges in the graph, that is,

$$p = \frac{2|E|}{|V|(|V|-1)}$$

Graph clustering coefficient The graph clustering coefficient C is a measure of the degree to which nodes in a graph tend to cluster together. The clustering coefficient of a node v (as known as the local clustering coefficient) is defined as the fraction of the number of triangles that include node v to the maximum possible number of

triples centered on node v. The clustering coefficient of the graph (as known as the global clustering coefficient) is defined as the average of the clustering coefficients of all nodes in the graph. The formal definition is

$$C = \frac{1}{|V|} \sum_{v \in V} \frac{2 \cdot |\{(i,j) \in E \mid i,j \in N(v)\}|}{d(v)(d(v)-1)}$$

where the term $|\{(i,j) \in E \mid i,j \in N(v)\}|$ is the number of edges between the neighbors of node v, i.e. the number of triangles containing node v. Furthermore, the term $\sum_{v \in V} \frac{2 \cdot |\{(i,j) \in E | i,j \in N(v)\}|}{d(v)(d(v)-1)}$ is the sum of the local clustering coefficients of all nodes in the graph, in which the degree of node v is denoted as $d(v)$.

Centrality measure of graphs At first we have the definition of centrality measure in the **node level**. The centrality measure of a node is a measure of the importance of the node in the graph. There are many centrality measures, such as degree centrality, closeness centrality, betweenness centrality, and eigenvector centrality.

Degree centrality is identical to the average degree of the graph defined above.

Closeness centrality $C_C(v)$ is defined as the reciprocal of the sum of the length of the shortest paths from the node v to all other nodes in the graph, that is

$$C_C(v) = \frac{1}{\sum_{w \in V} d(v,w)}$$

where $d(v,w)$ is the shortest path between node v and node w.

Betweenness centrality $C_B(v)$ is the sum of the fraction of the shortest paths between all pairs of nodes that pass through node v, that is

$$C_B(v) = \sum_{s,t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of the shortest paths between nodes s and t, and $\sigma_{st}(v)$ is the number of the shortest paths between nodes s and t that pass through node v.

Eigenvector centrality is a measure of the influence of a node in a graph based on the concept that connections to high-scoring nodes contribute more to the score of the node in question. Formally, the eigenvector centrality $C_E(v)$ of a node v

is defined as the v-th component of the eigenvector corresponding to the largest eigenvalue of the adjacency matrix A of the graph. Mathematically, it can be expressed as:

$$C_E(v) = \frac{1}{\lambda} \sum_{u \in N(v)} a_{vu} C_E(u)$$

where λ is the largest eigenvalue of the adjacency matrix A. a_{vu} is the element of the adjacency matrix A corresponding to the edge between nodes v and u. $C_E(u)$ is the eigenvector centrality of node u.

Finally, in the **graph level**, the centrality measure of the graph is defined as the average of the centrality measures over all nodes in the graph.

Average number of coloring in the 1-WL algorithm The 1-dimensional Weisfeiler-Leman algorithm (1-WL) [WL68] is an algorithm that assigns a unique label(color) to each node in the graph, also known as color refinement. The average number of coloring in the 1-WL algorithm is defined as the average number of colors used to color the nodes in the graph. The 1-WL algorithm is a powerful graph isomorphism algorithm that can be used to determine if two graphs are isomorphic. I give the procedure of the 1-WL algorithm here:

1. Assign the same color to all nodes in the graph.
2. Two nodes v, u are assigned a different color if there is a color c such that the number of c-colored neighbors of u is not equal to the number of c-colored neighbors of v.
3. Repeat step 2 until the colors of all nodes do not change.

4 Results & Discussion

In this chapter, the results of the experiments are presented and discussed. The experiments were conducted to evaluate the impact of different neural network structures and parameters on image reconstruction. The experiments were conducted using the four neural network architectures described in Chapter 3. The neural networks were trained and tested using the datasets described in Section 3.2.2. The experiments were conducted using the training and testing procedures described in Section 3.2.

4.1 Graph parameters

The graph parameters calculated in the experiments are shown in Table A.1 A.2.

4.2 Graph Convolutional Networks

The results of the experiments using Graph Convolutional Networks are shown in Table 4.1.

Name	Ave. generalization error	Standard deviation
MCF-7	0.002249991986900568	0.01604018546640873
MCF-7H	0.004124999046325684	0.02702830731868744
MOLT-4	0.0	0.014102176763117313
MOLT-4H	0.008624988608062267	0.01870434731245041
Mutagenicity	0.03765624016523361	0.024758048355579376
MUTAG	0.13552632927894592	0.10957122594118118
NCI1	0.06834375858306885	0.0339551605284214
NCI109	0.07212500274181366	0.030083369463682175
NCI-H23	0.003218746278434992	0.013211547397077084
NCI-H23H	0.0006250202422961593	0.011396590620279312
OVCAR-8	-0.0001249849738087505	0.015103230252861977
OVCAR-8H	0.0001562476099934429	0.013315632939338684
P388	0.008062511682510376	0.01043933816254139
P388H	0.006343752145767212	0.007148966193199158
PC-3	0.0008750140550546348	0.012619029730558395
PC-3H	0.0001250088243978098	0.00780930370092392
PTC_FM	0.12454469501972198	0.14246851205825806
PTC_FR	0.08159632235765457	0.09593729674816132
PTC_MM	0.12555554509162903	0.12687508761882782
PTC_MR	0.058610398322343826	0.11794959753751755
SF-295	0.0009687542915344238	0.013665653765201569
SF-295H	0.004500013776123524	0.018203353509306908
SN12C	0.0004999935626983643	0.017035124823451042
SN12CH	0.006718760821968317	0.016944456845521927
SW-620	5.9604645663569045e-09	0.006674798205494881
SW-620H	0.002499997615814209	0.008080806583166122
UACC257	0.0022812604438513517	0.013034629635512829
UACC257H	6.25073880655691e-05	0.015311303548514843
Yeast	0.0022187530994415283	0.01859860308468342
YeastH	0.00024999381275847554	0.015245940536260605
PROTEINS_full	0.11115662008523941	0.5160216689109802
DD	-0.1750326007604599	0.3490835428237915
ENZYMES	0.3110416531562805	0.06384395807981491
KKI	0.04776120185852051	0.2546080946922302
OHSU	0.39824172854423523	0.138640895485878
Peking_1	0.2980072498321533	0.2580110728740692
COLORS-3	0.07568749040365219	0.03585537523031235
SYNTHETIC	-0.41749995946884155	0.3929196000099182
SYNTHETICnew	-0.36250001192092896	0.45766785740852356
Synthie	0.14562499523162842	0.032582689076662064

Table 4.1: Generalization error of the GCN model on 57 datasets

5 Conclusion

A Graph Parameters

Name	DEG	ASP	DIA	DEN	GCC	ACC	ABC	AEC	WLC
COIL-DEL	4.6731	2.1288	4.0697	0.3275	0.5463	0.5122	0.0648	0.2431	21.1026
COIL-RAG	1.8277	1.0605	1.2422	0.9240	0.5904	0.9437	0.0463	0.5905	1.3115
Cuneiform	4.0796	2.2444	3.0000	0.2222	0.8634	0.4613	0.0723	0.1963	2.0000
Fingerprint	1.4869	2.1151	4.3444	0.4392	0.0017	0.5069	0.1840	0.4208	3.6566
FIRSTMM_DB	4.5024	12.3090	30.6761	0.0074	0.2632	0.0552	0.0188	0.0093	1357.8293
Letter-high	1.8897	1.4794	2.4445	0.5792	0.2976	0.6795	0.1658	0.4612	2.9942
Letter-low	1.3214	1.3336	1.9558	0.4165	0.0000	0.4955	0.1619	0.4407	2.4711
Letter-med	1.3527	1.3502	2.0066	0.4246	0.0139	0.5071	0.1683	0.4417	2.5173
MSRC_9	4.8153	3.1136	7.0045	0.1236	0.5413	0.3310	0.0552	0.1369	40.3620
MSRC_21	5.1024	4.0945	9.6217	0.0681	0.5147	0.2515	0.0415	0.0937	77.2274
MSRC_21C	4.7826	3.1138	6.9713	0.1240	0.5427	0.3315	0.0557	0.1372	40.0526
DD	4.9791	7.9199	19.7002	0.0278	0.4794	0.1402	0.0325	0.0342	278.2801
ENZYMES	3.8626	4.2811	10.4266	0.1599	0.4534	0.2636	0.1159	0.1471	28.5750
KKI	3.1857	3.5009	8.1446	0.1793	0.3526	0.3359	0.1298	0.1757	24.8193
OHSU	4.3277	4.9217	12.6962	0.0823	0.3808	0.2407	0.0656	0.0824	77.5190
Peking_1	3.5514	3.9681	9.6235	0.1276	0.3673	0.2889	0.0998	0.1285	36.7294
PROTEINS	3.7346	4.6022	11.2971	0.2122	0.5142	0.3023	0.1184	0.1695	34.4690
PROTEINS_full	3.7346	4.6022	11.2971	0.2122	0.5142	0.3023	0.1184	0.1695	34.4690
COLORS-3	2.9289	2.5810	5.2758	0.1696	0.1450	0.3503	0.0639	0.1737	56.9590
SYNTHETIC	3.9200	3.5065	7.0000	0.0396	0.0203	0.2893	0.0256	0.0832	100.0000
SYNTHETICnew	3.9250	3.4873	7.2748	0.0396	0.0229	0.2894	0.0255	0.0836	99.9633
Synthie	3.6210	2.6349	5.4948	0.0384	0.0937	0.2766	0.0232	0.0787	90.6125
TRIANGLES	3.1405	2.2634	4.5575	0.2416	0.2286	0.4374	0.0821	0.2262	19.8837
AIDS	2.0129	3.4905	7.7626	0.1935	0.0072	0.3501	0.1912	0.2628	11.8145
BZR	2.1467	5.1548	11.6568	0.0649	0.0001	0.2055	0.1245	0.1417	29.6074
BZR_MD	20.3039	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.2204	1.0000
COX2	2.1077	5.9364	13.7901	0.0529	0.0007	0.1757	0.1263	0.1288	28.0942
COX2_MD	25.2772	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.1957	1.0000
DHFR	2.1024	6.1447	14.6032	0.0537	0.0000	0.1734	0.1289	0.1272	32.2156
DHFR_MD	22.8677	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.2077	1.0000
ER_MD	20.3274	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.2245	1.0000

Table A.1: Graph parameters of the datasets

Name	DEG	ASP	DIA	DEN	GCC	ACC	ABC	AEC	WLC
FRANKENSTEIN	2.0628	3.6743	8.4235	0.1705	0.0107	0.3148	0.1924	0.2432	13.4245
MCF-7	2.1516	5.0501	12.1412	0.0981	0.0020	0.2175	0.1724	0.1725	21.4181
MCF-7H	2.0889	5.8586	13.3062	0.0537	0.0006	0.1778	0.1198	0.1223	32.9756
MOLT-4	2.1461	4.9517	11.8746	0.0987	0.0019	0.2183	0.1714	0.1734	21.0290
MOLT-4H	2.0858	5.7381	12.9924	0.0539	0.0005	0.1782	0.1193	0.1229	32.2722
Mutagenicity	2.0380	4.4189	9.6370	0.0913	0.0031	0.2467	0.1372	0.1710	20.8746
MUTAG	2.1888	3.6262	8.2181	0.1385	0.0000	0.2938	0.1694	0.2197	14.8032
NCI1	2.1550	5.2349	12.6594	0.0889	0.0031	0.2026	0.1632	0.1607	24.7214
NCI109	2.1564	5.1840	12.4953	0.0894	0.0031	0.2041	0.1624	0.1618	24.5241
NCI-H23	2.1452	4.9509	11.8695	0.0988	0.0019	0.2184	0.1716	0.1735	20.9930
NCI-H23H	2.0853	5.7343	12.9837	0.0540	0.0005	0.1782	0.1193	0.1230	32.2258
OVCAR-8	2.1456	4.9452	11.8522	0.0988	0.0019	0.2184	0.1714	0.1735	20.9856
OVCAR-8H	2.0856	5.7287	12.9668	0.0540	0.0005	0.1782	0.1192	0.1230	32.2256
P388	2.1092	4.3558	10.2182	0.1191	0.0038	0.2460	0.1790	0.1958	17.1332
P388H	2.0639	5.2898	11.7657	0.0640	0.0010	0.1971	0.1240	0.1373	26.7110
PC-3	2.1521	5.0502	12.1409	0.0981	0.0020	0.2176	0.1725	0.1726	21.3734
PC-3H	2.0892	5.8578	13.3065	0.0538	0.0006	0.1779	0.1200	0.1225	32.8819
PTC_FM	1.9758	3.3116	7.3668	0.2155	0.0091	0.3693	0.2056	0.2786	9.9140
PTC_FR	1.9863	3.3877	7.5897	0.2086	0.0088	0.3613	0.2045	0.2734	10.2222
PTC_MM	1.9701	3.3014	7.3393	0.2223	0.0131	0.3745	0.2060	0.2823	9.8036
PTC_MR	1.9805	3.3578	7.5233	0.2138	0.0095	0.3666	0.2062	0.2778	10.0407
SF-295	2.1455	4.9446	11.8521	0.0988	0.0019	0.2184	0.1715	0.1736	20.9716
SF-295H	2.0855	5.7274	12.9664	0.0540	0.0005	0.1783	0.1192	0.1230	32.1961
SN12C	2.1450	4.9483	11.8640	0.0988	0.0019	0.2183	0.1715	0.1736	20.9917
SN12CH	2.0853	5.7301	12.9736	0.0540	0.0005	0.1782	0.1193	0.1230	32.2363
SW-620	2.1452	4.9439	11.8516	0.0989	0.0019	0.2185	0.1715	0.1736	20.9720
SW-620H	2.0854	5.7258	12.9629	0.0541	0.0005	0.1783	0.1193	0.1231	32.1965
UACC257	2.1457	4.9558	11.8857	0.0987	0.0019	0.2183	0.1716	0.1735	20.9942
UACC257H	2.0857	5.7405	13.0032	0.0540	0.0005	0.1782	0.1194	0.1230	32.2244
Yeast	2.0994	4.3643	10.2634	0.1211	0.0023	0.2477	0.1831	0.1986	16.6748
YeastH	2.0590	5.2454	11.6875	0.0643	0.0006	0.1984	0.1255	0.1385	26.0532
Sum	242.8348	258.6724	587.9183	12.7317	11.8459	20.6872	7.3848	11.6365	3245.5993

Table A.2: Graph parameters of the datasets

Bibliography

[AL19] Zeyuan Allen-Zhu and Yuanzhi Li. "Can sgd learn recurrent neural networks with provable generalization?" In: *Advances in Neural Information Processing Systems* 32 (2019) (cit. on p. 3).

[BO04] Albert-Laszlo Barabasi and Zoltan N Oltvai. "Network biology: understanding the cell's functional organization." In: *Nature reviews genetics* 5.2 (2004), pp. 101–113 (cit. on p. 1).

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. Springer, 2006 (cit. on p. 5).

[BAY21] Shaked Brody, Uri Alon, and Eran Yahav. "How attentive are graph attention networks?" In: *arXiv preprint arXiv:2105.14491* (2021) (cit. on pp. 2, 12, 16).

[CLZ19] Minshuo Chen, Xingguo Li, and Tuo Zhao. "On generalization bounds of a family of recurrent neural networks." In: *arXiv preprint arXiv:1910.12947* (2019) (cit. on p. 3).

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011) (cit. on p. 9).

[EK+10] David Easley, Jon Kleinberg, et al. *Networks, crowds, and markets: Reasoning about a highly connected world*. Vol. 1. Cambridge university press Cambridge, 2010 (cit. on p. 1).

[FL19] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric." In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019 (cit. on p. 15).

[Fra18] Peter I Frazier. "A tutorial on Bayesian optimization." In: *arXiv preprint arXiv:1807.02811* (2018) (cit. on p. 16).

[GJJ20] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. "Generalization and representational limits of graph neural networks." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3419–3430 (cit. on p. 3).

[Gil+17] Justin Gilmer et al. "Neural message passing for quantum chemistry." In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272 (cit. on pp. 1, 2, 12, 16).

[GRS18] Noah Golowich, Alexander Rakhlin, and Ohad Shamir. "Size-independent sample complexity of neural networks." In: *Conference On Learning Theory*. PMLR. 2018, pp. 297–299 (cit. on p. 3).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016 (cit. on p. 9).

[GMS05] Marco Gori, Gabriele Monfardini, and Franco Scarselli. "A new model for learning in graph domains." In: *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.* Vol. 2. IEEE. 2005, pp. 729–734 (cit. on pp. 1, 2).

[Gra13] Alex Graves. "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850* (2013) (cit. on p. 9).

[HSS08] Aric Hagberg, Pieter J Swart, and Daniel A Schult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 2008 (cit. on p. 17).

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 1).

[Kin14] Diederik P Kingma. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on pp. 9, 16).

[KW16] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." In: *arXiv preprint arXiv:1609.02907* (2016) (cit. on pp. 1, 2, 10, 16).

[Mor+20] Christopher Morris et al. "TUDataset: A collection of benchmark datasets for learning with graphs." In: *arXiv preprint arXiv:2007.08663* (2020) (cit. on pp. 1, 15).

[Mor+24] Christopher Morris et al. *Future Directions in the Theory of Graph Machine Learning*. 2024. arXiv: 2402.02287 [cs.LG]. URL: https://arxiv.org/abs/2402.02287 (cit. on p. 3).

[Mor+21] Deisy Morselli Gysi et al. "Network medicine framework for identifying drug-repurposing opportunities for COVID-19." In: *Proceedings of the National Academy of Sciences* 118.19 (2021), e2025581118 (cit. on p. 1).

[STH18] Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. "The vapnik–chervonenkis dimension of graph and recursive neural networks." In: *Neural Networks* 108 (2018), pp. 248–259 (cit. on p. 3).

[Sca+08] Franco Scarselli et al. "The graph neural network model." In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80 (cit. on p. 2).

[SK17] Martin Simonovsky and Nikos Komodakis. "Dynamic edge-conditioned filters in convolutional neural networks on graphs." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3693–3702 (cit. on p. 1).

[Vel+20] Petar Velickovic et al. "Pointer graph networks." In: *stat* 1050 (2020), p. 11 (cit. on pp. 1, 2, 16).

[Vel23] Petar Veličković. "Everything is connected: Graph neural networks." In: *Current Opinion in Structural Biology* 79 (2023), p. 102538 (cit. on p. 5).

[WL68] Boris Weisfeiler and Andrei Leman. "The reduction of a graph to canonical form and the algebra which appears therein." In: *nti, Series* 2.9 (1968), pp. 12–16 (cit. on p. 19).

[Wu+19] Felix Wu et al. "Simplifying graph convolutional networks." In: *International conference on machine learning*. PMLR. 2019, pp. 6861–6871 (cit. on pp. 11, 16).

[ZLD18] Jiong Zhang, Qi Lei, and Inderjit Dhillon. "Stabilizing gradients for deep neural networks via efficient svd parameterization." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5806–5814 (cit. on p. 3).