UCLouvain

epl

École polytechnique de Louvain

Study of Convolutional Models for Semi-Supervised Classification on Graph-Structured Data

Author: **Pierre-François De Plaen**
Supervisors: **Bertrand Lebichot, Marco Saerens**
Readers: **Sylvain Courtain, Siegfried Nijssen**
Academic year 2020–2021
Master [120] in Mathematical Engineering

Abstract

Due to the rapid growth of the internet, there are more and more graph-structured datasets such as social networks, co-purchase graphs, citation networks, and so on. The combination of both the numerical features and the graph structure of these datasets can improve the classification abilities. In the last few years, convolutional graph Neural Networks have shown encouraging results for semi-supervised classification on graph-structured datasets. However, we show through a theoretical analysis and a series of experiments that part of these models are not able to correctly classify nodes when the graph structure does not meet the autocorrelation hypothesis and suggest this is due to the low-pass behavior of these models. Only the Graph Wavelet Neural Network successfully predicts the class of the graph nodes on both feature-driven and graph-driven datasets. We present a novel approach that further improves the model by working at different wavelet scales. We demonstrate in a number of experiments that our approach outperforms the one scale approach on challenging datasets by a significant margin. Our work also shows that model regularization does in general not improve the generalization capabilities of convolutional graph Neural Networks, despite being heavily used in practice. We exhibit, however, that graph based regularization might help.

Additionally, we provide an open-source toolbox to help implement and compare classification models on graph-structured data. The toolbox is available at: `https://github.com/pfdp0/graph-classification-TF2-toolbox`.

Contents

List of Symbols

Graph Theory

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ Graph defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$

A Adjacency matrix of a graph

D Diagonal degree matrix defined by $D_{ii} = \sum_{j=1}^{N} A_{ij}$

L Graph Laplacian matrix defined by $L = D - A$

L^{norm} Normalized Graph Laplacian matrix defined by $L^{norm} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$

Matrix theory

$\odot$ Element-wise matrix multiplication

$\det(X)$ Determinant of matrix X

$\text{diag}(x)$ Diagonal matrix formed by the entries of vector x

I_N Identity matrix of size N

U, λ Eigenvectors and eigenvalues of a squared matrix X, such that $X = U diag(\lambda) U^T$

$X_{:,j}$ j-th column of matrix X

$X_{i,:}$ i-th row of matrix X

$X_{i,j}$ j-th value of the i-th row of matrix X

Neural Networks

$\hat{Y}$ Output of a Neural Network for a given entry

$\mathcal{L}(\cdot)$ Loss function

$\sigma(\cdot)$ Activation function

H_l Values of the layer l of a Neural Network

$N_{(\cdot)}$ Number of elements of set $(\cdot)$

Other Symbols

$\langle \cdot, \cdot \rangle$ Scalar product

$\mathbb{E}$ Mathematical expectation

$\mathcal{O}$ Big O notation

Number Sets

$\mathbb{R}$ Set of real Numbers

Chapter 1

Introduction

With the emergence of the world wide web, more and more data/knowledge has been collected and made easily available: millions of images, text messages, videos and articles are uploaded on the web every day. One of the challenges coming with such large sets of data is the task of classifying them. That is, to give label to each element to tell what it is about. As an example, suppose we want to identify what each new image uploaded on social media represents. A straightforward way of doing it would be to ask someone to look at the images one by one and to tell what each one is about. This approach could maybe work for about a thousand of images but would be infeasible for trillions of them. Therefore, many researchers have tried to build computer programs able to perform classification tasks. Starting in 2005, an image recognition competition called Pascal VOC and later ImageNet [31] was organized every year to reward the best improvements in that field. In 2012, a huge improvement has been made by a model called AlexNet [49]. The model used a Machine Learning model called Convolutional Neural Network (CNN) to classify the images. At that time, the model outperformed the next best model by a margin of 40%, being able to correctly classify about 85% of the images. Since then, research in Machine Learning and in particular in deep networks such as CNNs has gained interest and evolved quickly.

On the internet, many elements can be modelled by pairwise relations. Take for example a scientific publication: in addition to its text, its author and its publication date, the publication is also linked to other ones through citations. The normal way to represent such pairwise relations is to use a mathematical structure called a graph [84]. Consider now the task of classifying the publications by subject. That is, build a model which predicts the subject of the articles. The model uses both the articles' content and the graph structure. The motivation behind the use of the graph structure is that it could provide some useful additional information about the publications. In particular, it is assumed that linked articles are more likely to belong to the same class [27]. This hypothesis is called autocorrelation [88] and plays an important role in the design of the studied models.

In this work, we investigate the task of classifying nodes of a graph using models built using CNN-like structures since these models have shown outstanding results in many different Machine Learning tasks. Furthermore, we will see that the convolution operation of CNNs has the effect of spreading information to neighbouring pixels of an

image, which is a desired effect under the autocorrelation hypothesis. Unfortunately, the extension of convolution to graph-structured data is not straightforward. Indeed, the convolution operation of CNNs works only on grid-like structures, whereas graphs typically have way more complex structures on which the convolution operation is not applicable. Therefore, models like the Graph Convolutional Network, the Wavelet Graph Neural Network and the Diffusion Convolutional Neural Network have taken different approaches to extend the notion of convolution to graphs.

We compare state of the art graph node classification models by using a rigorous methodology and provide an easy to use open source toolbox to compare them with any other model. By doing so, we get a good overview of which model performs best, whatever the size and the nature of the graph. From there, we highlight a necessary property for models to systematically achieve a good accuracy. Also, we expect to help researchers by providing an efficient and easy to use comparison tool in which multiple models are already implemented.
We then investigate different ways of improving the models using regularization and other loss functions. With these experiments, we show that the frequently used cross-entropy is a reasonable choice but that other choices may lead to better performances for the Graph Diffusional Neural Network. We also show that model regularization of CNN-like models on graphs do not improve the models, whereas a well chosen smoothing regularization based on the underlying structure of the graph can help reaching better performance.

From the conclusions drawn from the different experiments, we suggest an improved version of the Graph Wavelet Neural Network that further improves the accuracy on the most challenging dataset of this work and give directions to improve it further.

This work fulfills the following goals:

- Provide a fair comparison and clear insights about convolutional models on the task of semi-supervised classification of graph-structured data. This comparison answers the following questions: which convolutional model performs the best? What is the complexity of these models? How do these models compare to simpler models? What is the influence of the nature of the graph on the classification accuracy? Can we highlight desired properties for convolutional models on graph-structured data?

- Answer to research questions that are too often left aside: is the softmax-cross entropy loss always the best choice? Do model and graph regularization improve the generalization capabilities of convolutional models? We then improve each of the three studied convolutional models using the answers to the previously stated questions and answer the question: which improved convolutional model performs the best?

- Introduce a novel model that gives encouraging results on challenging classification tasks and suggest further research directions. The model is referred as Multiscale Graph Wavelet Neural Network (M-GWNN).

- Provide an open-source toolbox to help researchers fairly compare models on a large set of datasets with multiple already implemented models and an easy guide to implement new ones.

The document is structured as follows: in the first chapter, we formally introduce the notion of classification using Machine Learning as well as three of the most used models, namely the Support Vector Machine, the Multilayer Perceptron and the Convolutional Neural Network. The second chapter is dedicated to the theory behind training a Neural Network. Different loss functions are studied along with their main properties. The end of the chapter introduces the main model regularization techniques proposed in the literature. From there, we introduce the main question that this work addresses and present the investigated models. From these models, we also introduce a new model, namely the Multiscale Graph Wavelet Neural Network. This model provides both a new structure and new basis to perform convolutions. From there, the fourth chapter answers the main question of this work: which model performs the best and why ? The question is answered through a set of experiments comparing the models and we draw several conclusions on the models performances. These conclusions motivate the interest of the Multiscale Wavelet Neural Network (M-GWNN). The M-GWNN is then compared to the one scale Graph Wavelet Neural Network. In the penultimate chapter, we motivate the utility of the toolbox to help researchers to compare efficiently different graph node classification models and provide useful information about the toolbox features. The last chapter concludes the work and provides further research directions.

Chapter 2

Basics of Machine Learning

In the computer science field [67, 72], artificial intelligence (AI), sometimes also referred as computational intelligence [67], is the study of the design of intelligent agents. An intelligent agent is a system that takes appropriate decisions in its environment to reach its goals. These choices are constrained by perceptual limitations and must be made within a finite computation time and memory. For example, an algorithm playing chess by comparing possible moves and choosing which one to play with pre-defined heuristics is an artificial intelligence [56, 41].

Machine Learning (ML) is a sub-field of AI in which the agents will improve a part of their components automatically by learning from data and from experience [60, 72]. A Machine Learning algorithm will thus learn to perform a task without being explicitly told how to do so. If we take again the example of an AI playing chess, the AI will now learn by looking at a large sample of games from which it will try to learn which moves are the most effective to reach its goal, winning the game [26, 78]. There are many different types of learning algorithms [62]. These can be arranged into different categories, depending on the task for which the models are built and the type of data they learn from. In this work, we will focus on the family of semi-supervised classification algorithms.

2.1 Supervised and Semi-supervised Learning

First, let us introduce supervised learning. A supervised learning algorithm [40, 72] builds a mathematical model from a set of training data which consists of data points and a desired output for each data point. In particular, supervised classification is a task in which the possible outputs, also called labels, are a finite set of classes. The goal is thus to build a model that best maps the data points to the corresponding classes. Ideally, the model should then be able to predict accurately the class of new data points for which the output class is unknown.

Semi-supervised classification algorithms [72] are, like for supervised learning, models that learn from a set of training examples but for which part of the desired outputs are missing. Typically, only a small part of the labels are available. The goal of semi-supervised classification is thus to build a model using all the training

data as well as the few known labels. Indeed, one could hope that the information added by the unlabelled data points could help improve the results.

More formally, the task of semi-supervised classification can be defined as follows [35, 43]: given a training set, which consists of a set of N data points or features of dimension N_F and a set of $L < N$ known labels assigned to a few features:

$$\{(x_i, y_i)\}_{i=1}^{N} \subset \mathbb{R}^{N_F} \times \{0,1\}^{N_C} \tag{2.1}$$

where N_C is the number of classes. Find a function $\phi : \mathbb{R}^{N_F} \to \{0,1\}^{N_C}$ that assigns a class to each data point and such that the classification accuracy is maximal[1]. The set of features is often represented by a so-called features matrix $X \in \mathbb{R}^{N \times N_F}$.

As an example, let us consider we have certain information about a group of people (e.g. their age, income, marital status and living place) as well as the voting intentions of some of them. A semi-supervised classification task could be to train a model that will aim to predict the voting intentions of the others, given all the information and the few known voting intentions.

There is a large variety of models that can be trained to accomplish classification tasks. In the following sections, we will first introduce the notion of Artificial Neural Network. We will then present two of them (the Multi-Layer Perceptron and the Convolutional Neural Network) as well as a third model, namely the Support Vector Machine. These models will be the basis for chapter 4, in which we will present algorithms that perform semi-supervised classification on graphs.

2.2 Neural Networks

An Artificial Neural Network (ANN) or simply a Neural Network, is a computing model inspired by the human brain [72, 55]. The ANN model is composed of a collection of neurons which are linked by connections. Like in the biological brain, the connections transmit the information between two neurons. An artificial neuron is thus a unit that receives a signal, processes it by doing basic computations and then sends it to other neurons through its connections.
In the following, we will present two ANN's in which the signal flows one-way from the input of the model to the output of the model. This family of ANN's is called feed-forward Artificial Neural Networks [60].

2.2.1 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is an ANN [55, 60, 40] composed of successive layers of neurons. The network has three types of layers: the input layer, the hidden layers and the output layer. The information flows one-way from the input layer of the network to the output layer. An example of the structure of a MLP with one hidden layer in shown on Figure 2.1.

[1]Note that getting a perfect accuracy for the known data points is not sufficient, since it could poorly generalize to the unlabelled data points. This will be further discussed in Section 3

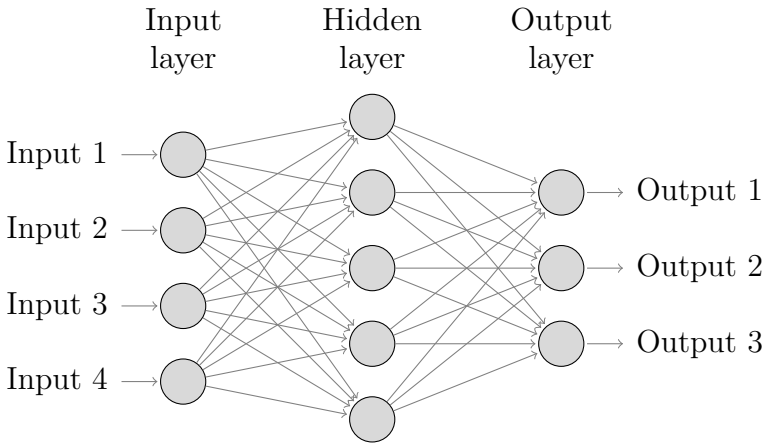


Figure 2.1: Example structure of a Multi-Layer Perceptron with one hidden layer. Illustration inspired from [38].

A neuron receives signals from the previous layer through the connections, sums the received values and then applies a function called activation function. Activation functions are presented in Section 2.2.3. Each connection of the network, also called edge, links two neurons of successive layers. Each edge has a weight. The values of the weights control the importance of the connection between two neurons.

More formally, each neuron of a MLP applies a nonlinear weighted combination of the values of the neurons in the previous layer [55]. Let $h_{l,i}$ be the output of the i^{th} neuron of the l^{th} layer, its value is then given by:

$$h_{l+1,i} = \sigma\left(\sum_{j=1}^{N_l} w_{l,j,i} h_{l,j} + b_{l,i}\right) \tag{2.2}$$

where $w_{l,j,i}$ is the weight of the edge between node j of layer l and node i of layer $l+1$, $b_{l,i}$ is the bias term of the neuron and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function. The Multi-Layer Perceptron is trained by changing the weights and the biases so that the output of each labelled data closely matches the desired values. The way the updates are done is discussed in Section 3.

2.2.2 Convolutional Neural Network (CNN)

In this section, we now introduce the Convolutional Neural Network. This model is the basis of the three main graph node classification methods that are introduced in Section 4.

So far, we have seen that a MLP neuron takes as input a weighted average of all the neurons of the previous layer and that the input of the model (i.e. the input layer) is a vector. Suppose now we want to classify images composed of a grid of $N \times M$ pixels. The input of a MLP is then a vector of size $N \times M$, which is a flattened version of the image. This approach requires many weights and thus a large amount of images to train the model as well as a very long training time since the more weights a model has, the more training data it needs to fit these weights [1]. Furthermore, this method does not take advantage of the image structure. To

overcome these issues, let us take inspiration from digital image processing. One of the main operations in digital image processing is the image convolution [81, 79]. The (2D) convolution of a $N \times N$ image f by another $K \times K$ image h, often called kernel, is defined by [79]:

$$(f * g)(i,j) = \sum_{i'=1}^{K} \sum_{j'=1}^{K} f\left(i - i', j - j'\right) g\left(i', j'\right) \quad \text{for all } i,j = 1,\ldots,N \tag{2.3}$$

Intuitively, this means that a new image is formed by shifting the kernel g all over the image f and performing an element-wise multiplication at each position. One step of the 2D convolution of an image with a 3×3 kernel is illustrated on Figure 2.2. By doing so with well chosen kernels, it is then possible to apply many transformations to the image, as for example blurring the image or detecting edges. Furthermore, the combination of such kernels could allow to detect more and more complex shapes or textures in the image [8].

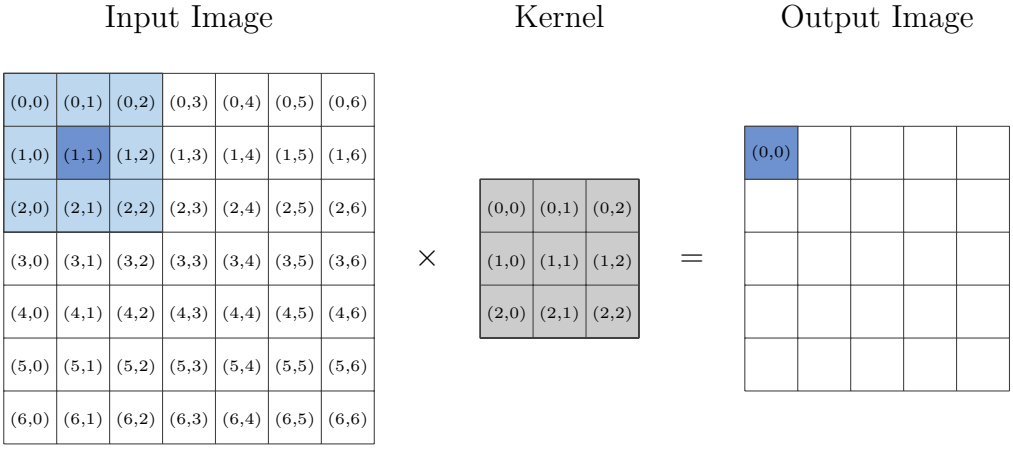


Figure 2.2: Application of one step of the image convolution. Illustration inspired from [8].

From there, one could build a model for image classification by using multiple convolution kernels. This is the main idea behind the Convolutional Neural Network (CNN). Let the neurons of the input layer be an image stored as a tensor of shape $N \times M \times D$, where $N \times M$ is the shape of the image (i.e. its number of pixels) and D are its different channels. An image has typically 3 channels (Red, Green and Blue). The CNN structure is composed of three main types of layers [3]:

1. **Convolutional layer**: this layer convolves the input tensor with a kernel of learnable weights. It then applies an eventual bias and passes the output of each neuron through an activation function. Formally, the operation of a convolutional layer is given by:

$$H_{l+1,k} = \sigma\left(H_{l,:} * W_{l,k} + B_{l,k}\right) \tag{2.4}$$

where $H_{l,k}$ are the neurons of channel k of layer l, $H_{l,:}$ are all the neurons of all channels of layer l, $W_{l,k}$ and $B_{l,k}$ are respectively the kernel and the bias

associated to the k^{th} channel of layer l and σ is an activation function applied separately on each neuron. Note that two consecutive layers might have a different number of channels and that kernels can be designed to take only a subset of the channels of the previous layer into account by fixing some of its weights to zero.
If we compare this operation to the one performed by a MLP, the input of the neuron is now dependent only of neighbouring neurons (i.e. neighbouring pixels) of the previous layer and the weights of the kernel are the same for all the neurons of a given layer. By doing so, the spatial relations between the features are taken into account and the number of learnable weights is reduced.

2. **Downsampling layer (or pooling layer)**: this layer reduces the size of the image by dividing the image into small clusters of neighbouring neurons (for example of size 2×2 each) and then combining the neurons into one neuron. Typical operations are max-pooling (taking the max value along each cluster) and average-pooling (taking the average value of each cluster). The operation can be denoted by: $H_{l+1,k} = down\,(H_{l,k})$.
Intuitively, the goal of the pooling layer is to summarize the image information into a smaller image by merging the information of neighbouring pixels.

3. **Fully connected layer**: in the last layers of the network, the neurons are flattened and fully connected together using typically one or two MLP layers. This is done to map the layers to the output classes.

A simple example of CNN structure for image classification is given on Figure 2.3. This example takes as input a vehicle image, applies two times a convolutional layer followed by a pooling layer and then one fully connected layer. The classes of the output layer then give the type of vehicle the image represents. The model uses two different activation functions, a Rectified Linear Unit (ReLU) and a Softmax. We will present these two activation functions as well as other common ones in the next section.

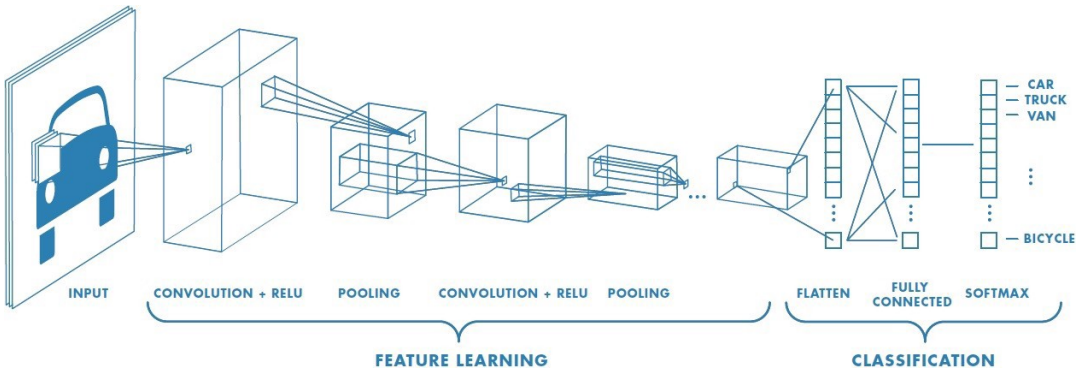


Figure 2.3: Typical CNN structure. Illustration from [82].

Images as graphs Intuitively, an image can be seen as a graph that has a grid-like structure in which neighbouring pixels are connected. The convolution with a 3×3

kernel is thus equivalent to performing a weighted linear combination of a node value with the ones of its neighbours, this is illustrated on Figure 2.4. Indeed, the next layer's value of a node at position i, j will be determined by the values of the nodes at positions $(i + i'), (j + j')$ with $i' \in \{-1, 0, 1\}$ and $j' \in \{-1, 0, 1\}$ in the case of a 3×3 kernel. The effect of the convolution operation is thus to spread the information of each node to its neighbouring nodes and the way the information is spread is controlled by the weights of the kernel. This observation is the fundamental for the building of convolutional methods on graphs that are presented in Section 4.3.

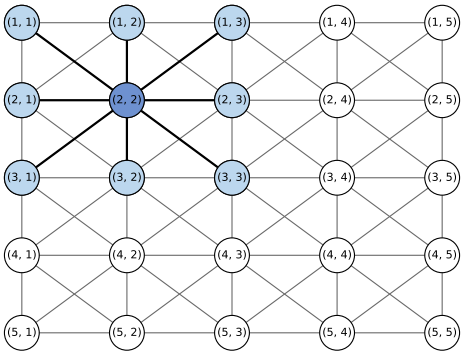


Figure 2.4: Application of one step of the image convolution with a 3×3 kernel. The image is represented as a graph in which the nodes are pixels and the edges are the neighbouring relations between the pixels.

2.2.3 Activation functions

In Neural Networks, an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is a function that applies a transformation to each node of a layer [72, 3]. The goal of most activation functions is to introduce non-linearities in the Neural Network so that it can approximate any function. Indeed, it can be proven that a two-layer neural network with non-linear activation functions can approximate arbitrary well any continuous function [19]. This property is commonly mentioned as Universal Approximation Theorem.

The most widely used activation functions in Machine Learning are the following:

- **Identity**: this activation function just takes the input as output. Its equation is given by: $\sigma_{id}(x) = x$. Note that this activation function has no effect on the node output, we define it so that any layer can be defined with an activation function at its output.

- **Logistic (sigmoid)**: the logistic is an activation function of the family of the sigmoids. According to [37], a sigmoid is a function that has the following properties: it is bounded, differentiable, has a non-negative derivative at each point and one inflexion point. Intuitively, sigmoids are functions that have a "s"-shape.
  The logistic activation function maps any real value to a $[0, 1]$ value and is defined by: $\sigma_{sigmoid}(x) = \frac{1}{1+e^{-x}}$.

- **Hyperbolic tangent (tanh)** the hyperbolic tangent is also a sigmoid function, it maps any real value to the $[-1, 1]$ interval and is defined by: $\sigma_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Rectified Linear Unit (ReLU)**: the ReLU is defined by: $\sigma_{relu}(x) = \max(0, x)$. It is mainly used in deep networks like CNNs for its ability to introduce non-linearities while being computationally cheaper to derive than a sigmoid. Indeed, its derivative is simply given by:

$$\frac{\partial \sigma_{relu}}{\partial x}(x) = \begin{cases} 1, & \text{for } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

These activation functions shapes are shown on Figure 2.5. The performances of the different activation functions will depend of the model and of the type of input data. Even if certain activation functions are known to perform better in some cases, their influence on the efficiency of a classification task is difficult to predict. For this reason, the activation function choice is one of the model hyper-parameters that will need to be tuned.

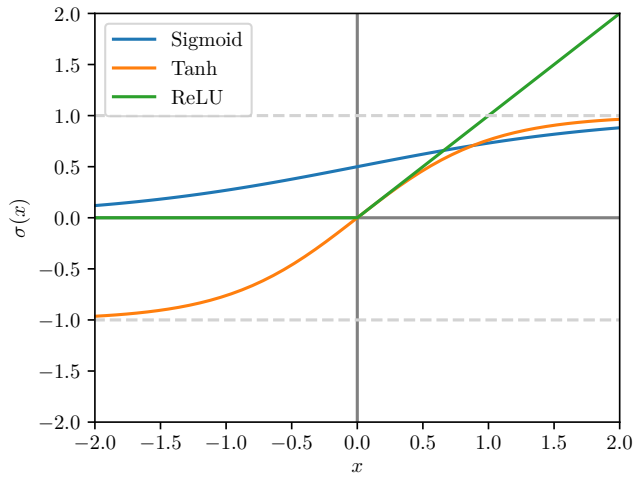


Figure 2.5: Shape of the different activation functions on interval $[-2, 2]$. Note that both sigmoid functions (blue and orange curves) are bounded.

Finally, there is the softmax activation function. It is a generalization of the logistic activation. It takes as input a full layer and outputs a probability distribution where the outputs are proportional to the exponential of the inputs. This activation function is typically applied to the output of a Neural Network. The softmax activation function for a layer of N_l neurons is defined by $\sigma_{softmax} : \mathbb{R}^{N_l} \to [0, 1]^{N_l}$ with

$$\sigma_{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N_l} e^{x_j}} \tag{2.5}$$

Now that we introduced Neural Networks, let us present another type of model that will be used as a baseline in the experiments of Chapter 5.

2.3 Support Vector Machine (SVM)

A Support Vector Machine [40, 72, 60] is a supervised learning algorithm that can be used for classification. To introduce SVMs, let us begin with a very simple linearly separable set of 2 classes and then progressively generalize to any supervised classification task.

Linear SVM Suppose we have a set of L data points $X = \{x_i\}_{i=1}^{L} \subset \mathbb{R}^N$ which all belong to one of two classes: $Y = \{y_i\}_{i=1}^{L} \subset \{-1, 1\}$ and that these two classes are linearly separable. Which means we can find a hyperplane that perfectly separates the data points of the two classes. A linear SVM is an algorithm that separates the two classes by finding the hyperplane with a maximal margin [40]. In other words, a hyperplane such that its distance to the closest point of each class is maximal. The problem can be formulated as follows [2]:

$$\begin{aligned} \max_{w,b} \quad & \frac{2|k|}{\|w\|} \\ \text{s.t.} \quad & y_i \left(w^T x_i + b \right) \geq k \quad \text{for } i = 1, 2, \ldots, L \end{aligned} \tag{2.6}$$

where the equation of the hyperplane is given by $w^T x + b = 0$ and k is the margin. A two dimensional example is shown on Figure 2.6.

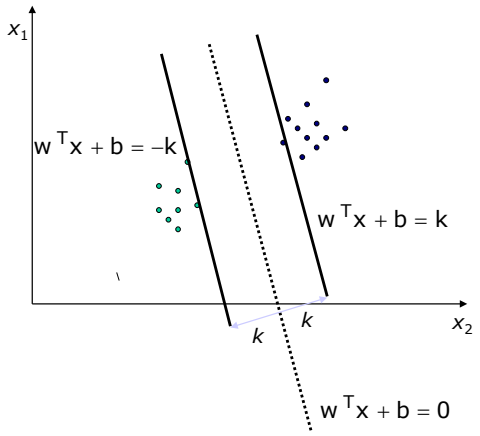


Figure 2.6: Linear SVM hyperplane for 2 dimensional linearly separable problem. Illustration from [2].

Without loss of generality, let us suppose the data is scaled such that $k = 1$, the problem is then equivalent to the following quadratic formulation [2]:

$$\begin{aligned} \min_{w,b} \quad & \|w\|^2 \\ \text{s.t.} \quad & y_i \left(w^T x_i + b \right) \geq 1 \quad \text{for } i = 1, 2, \ldots, L \end{aligned} \tag{2.7}$$

Let us now consider two classes that are not linearly separable because of a small number of outliers. The problem can be relaxed by allowing data points to fall

within the margin, but by penalizing them by a factor β in the objective function. The problem then becomes:

$$\begin{aligned} \min_{w,b} \quad & \|w\|^2 + \beta \sum_{i=1}^{L} \xi_i^2 \\ \text{s.t.} \quad & y_i \left(w^T x_i + b \right) \geq 1 - \xi_i \quad \text{for } i = 1, 2, \ldots, L, \\ & \xi_i \geq 0 \qquad\qquad \text{for } i = 1, 2, \ldots, L \end{aligned} \tag{2.8}$$

This formulation, often referred as soft-margin (linear) SVM, is convex. This implies that there is a unique global minimum and that it can be efficiently solved by a quadratic optimization solver [40].

Nonlinear SVM Consider now more complex sets of data points for which a linear decision surface would be inefficient. The nonlinear SVM solves this issue by mapping the data points into a high-dimensional space in which the points are linearly separable. The formulation of the optimization problem remains the same as in formulation 2.8 but with data points: $x_i' = \psi(x_i)$, where $\psi : \mathbb{R}^D \to \mathbb{R}^{D'}$ is a nonlinear mapping function.

Explicitly mapping each data point to the new space is computationally expensive when ψ is high dimensional. Fortunately, the Wolfe dual formulation of this problem gives a form in which the mapping only appears as an inner product $\Psi(x_i, x_j) = \psi(x_i)^T \psi(x_j)$:

$$\begin{aligned} \min_{\alpha_i} \quad & \frac{1}{2} \sum_{i,j} \left[\alpha_i \alpha_j y_i y_j \Psi(x_i, x_j) \right] - \sum_i \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq \beta \quad \text{for } i = 1, 2, \ldots, L, \\ & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{2.9}$$

Therefore, the problem reduces to finding a function $\Psi(x_i, x_j)$ which gives the inner product of two points in the new space. Furthermore, it turns out that $sign\left(w^T \psi(x) + b\right) = sign\left(\sum_i \alpha_i y_i \Psi(x_i, x) + b\right)$ and b can thus be found by solving $\alpha_j \left(y_j \sum_i \alpha_i y_i \Psi(x_i, x_j) + b - 1 \right) = 0$ for any j such that $\alpha_j \neq 0$. This trick thus allows to classify without explicitly mapping the data points into the high-dimensional space when training the SVM, which highly reduces the computation cost.

Multiclass SVM An approach to extend binary SVMs to multi-class problems is to decompose the problem into multiple binary problems using a one-vs-one approach, i.e. build a binary classifier for every pair of classes [22]. The assigned class for a data point is then the most selected one.

Chapter 3

Training a Neural Network

In the previous section, we have seen that the classification task in Machine Learning consists of building a model that best predicts the output class of a set of data points. The model is trained by passing multiple times through at a set of data points, of which a few or all of them are labelled. In particular, we presented two Neural Networks which can learn by updating their weights. There still remains a question: how are the weights updated? The weights of a Neural Network are updated so that these minimize the error between the true class of each data point and the value predicted by the output of the network. In short, the idea is to feed one by one each training example into the Neural Network. The output is then compared with the true label using a so called loss function. Then, the weights are updated by propagating the gradient of the loss backwards into the network using a chain rule called backpropagation of the gradients. Intuitively, backprogapation aims to tell in which direction each weight has to be updated so that the model better predicts the training examples. The training phase ends when a minimum is reached. Unfortunately, there is no guarantee of reaching a global minimum and to have built a model that generalizes well to new data points.

This chapter is organized as follows. Firstly, we present the most common loss functions and their properties. Then, we see how to update the weights of a model using the gradient of the loss. Finally, we present ways to asses how a model will generalize to new data points and how to avoid learning too much by using different regularization techniques.

3.1 Loss Functions

In Neural Networks, a loss function is a function that measures the difference between the desired output Y and the output of the network $\hat{Y}$. In that sense, the prediction error can be measured by any function that maps these two values onto a real number, which value is minimal when these two values are equal. Such a function is commonly named loss function and can be defined by:

$$\mathcal{L} : \mathbb{R}^{N_C} \times \mathbb{R}^{N_C} \to \mathbb{R} \tag{3.1}$$

The goal of a loss function is to give information about how to update the weights of the network. Like we will see in Section 3.2, the weights are updated using

back-propagation of the gradients. Therefore, a loss function needs to be at least one time differentiable. Moreover, some loss functions have other interesting properties that will be presented in this section.

Let a set of N_S training examples $S = \{(Y_i, \hat{Y}_i)\}_{i=1}^{N_S}$ where $Y_i \in \{0,1\}^{N_C}$ is the target class as binary indicator vector and $\hat{Y}_i \in \{0,1\}^{N_C}$ is the output of the Neural Network. The loss over the set S is computed as the sum of the individual losses:

$$\mathcal{L}(S) = \sum_{i=1}^{N_S} \mathcal{L}(\hat{Y}_i, Y_i)$$

In the following, we present different loss functions as well as their main properties.

Notations

For the rest of this section, we will use the following notations:

- $Y \in \{0,1\}^{N_C}$ is the target class as binary indicator vector. Its i-th element is denoted by y_i.
- $\tilde{Y} \in \{-1,1\}^{N_C}$ is the target class as $-1/1$ encoding.
- $\hat{Y} \in \mathbb{R}^{N_C}$ is the output of the Neural Network. Note that in general, the elements of this vector can take any real value. Its i-th element is denoted by $\hat{y}_i$.

As an example, if the item belongs to class number two, its binary indicator encoding is given by: $Y = [0,1,0,0,\cdots,0]^T$, its $-1/1$ encoding is: $\tilde{Y} = [-1,1,-1,-1,\cdots,-1]^T$.

3.1.1 L^p losses

The L^p-losses are loss functions that come from the family of p-norms. For any $p \geq 1$, the L^p-loss of vectors Y and $\hat{Y}$ is given by:

$$\mathcal{L}_{L^p}(\hat{Y}, Y) := \|\hat{Y} - Y\|_p^p = \sum_{i=1}^{N_C} |\hat{y}_i - y_i|^p \tag{3.2}$$

Note that the L^1-loss and the L^2-loss are respectively the mean absolute error (MAE) and the mean squared error (MSE).

When p approaches ∞, the p-norm approaches the maximum norm. Therefore, we define the L^∞-loss as follows:

$$\mathcal{L}_{L^\infty}(\hat{Y}, Y) := \|\hat{Y} - Y\|_\infty^1 = \max_i |\hat{y}_i - y_i| \tag{3.3}$$

Main properties

L^P-losses, despite being used almost only for regularization, present interesting properties for classification purposes. The main property is the following:

Proposition 3.1.1 *L^1-loss applied to the probability estimates $\hat{p}(\mathbf{y} \mid \mathbf{x})$ leads to minimisation of expected misclassification probability. Similarly L^2 minimises the same factor, but regularised with a half of expected squared L^2 norm of the predictions probability estimates.*

Another property of L^1 and L^2 losses might cause slow convergence rate of such losses:

Proposition 3.1.2 *L^1 and L^2 losses applied to probabilities estimates coming from sigmoid (or softmax) have non-monotonic partial derivatives wrt. the outputs of the final layer (and the loss is not convex nor concave wrt. last layer weights). Furthermore, they vanish in both infinities, which slows down learning of heavily misclassified examples.*

Proofs for propositions 3.1.1 and 3.1.2 can be found in [43].

Proposition 3.1.1 shows that L_1 and L_2 losses have interesting probabilistic interpretations for classification and can therefore be reasonable loss function choices, despite usually being used almost only for regularization purposes [43]. One could expect that this should lead to higher robustness to outliers, as we try to maximise the expected probability of good classification as opposed to the probability of perfect prediction. Unfortunately, Proposition 3.1.2 shows that these losses vanish in both infinities, which slows down learning of heavily misclassified data points. This slow convergence might be one of the reasons for the unpopularity of these losses as main objective function.

3.1.2 Cross-entropy loss

Cross-entropy is a measure of the difference between two probability distributions. It comes from information theory, in which measures the the average number of bits needed to encode data coming from a source with distribution p when we use a model q [63].

Formally, cross-entropy loss is defined as the opposite of the log-likelihood of conditionally independent samples [24]:

$$\mathcal{L}_{CE}(\hat{Y}, Y) := -\sum_{i}^{N_C} y_i \log(\hat{y}_i) \tag{3.4}$$

$$= -\log(\hat{y}_j) \text{ where } j \text{ is the true label} \tag{3.5}$$

Cross-entropy loss is commonly preceded by a softmax activation function, which maps values a probability estimate. Cross-entropy loss for two classes, as a function of the probability estimate, is drawn on Figure 3.1. The illustration shows that cross-entropy loss highly penalizes data points that are heavily misclassified.

Cross-entropy loss is by far the most common choice for both image classification [4, 45] and classification of graph-structured data [47, 87, 5]. We will check in Section 5.4.1 whether or not this choice is reasonable.

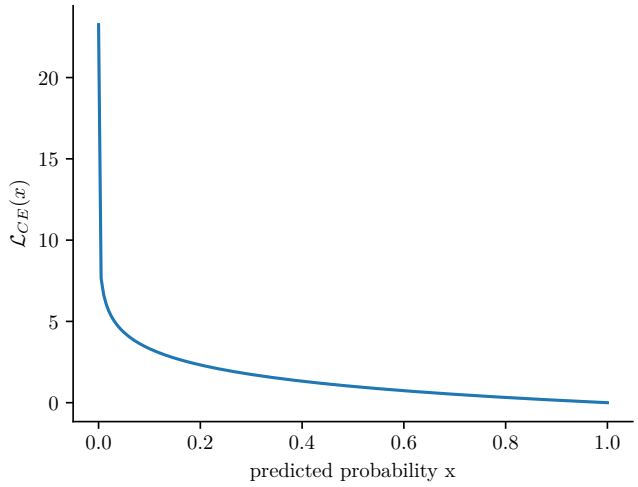


Figure 3.1: 2-classes cross entropy loss of probability estimate x.

Focal loss

Experiments [54] have shown that large class imbalance can be an issue when training using the cross-entropy loss. Focal loss is a solution to this issue. Focal loss reshapes the cross-entropy loss function to down-weight well-classified examples and thus focus training on hard negatives. Formally, focal loss is defined by:

$$\mathcal{L}_{focal_\gamma}(\hat{Y}, Y) := (1 - \hat{y}_j)^\gamma \log(\hat{y}_j) \text{ where } j \text{ is the observed label} \tag{3.6}$$

where $\gamma \geq 0$ is the focusing parameter. Higher values of γ will increase the down-weighting effect, whereas $\gamma = 0$ is equivalent to the cross-entropy loss.

3.1.3 Multiclass hinge loss

Recall the definition of a linear SVM presented in Section 2.3: a linear SVM is an algorithm that separates two classes by finding the hyperplane with a maximal margin. An alternative formulation to the optimization problem 2.7 for one point is the following:

$$\mathcal{L}_{SVM}(t, \hat{y}) = \max(0, 1 - t\hat{y}) \tag{3.7}$$

where $t \in \{-1, 1\}$ is the true label of point x and $\hat{y} = w^T x + b$ is the distance between point x and the hyperplane defined by equation $w^T x + b = 0$.
This loss will be zero when the predicted label $\hat{y}$ has the same sign as the true label t and has a value of at least 1. On the other side, the loss will grow linearly when it is within the hard margin or on the wrong side of the margin. Minimizing this loss will therefore lead to a margin such that the point x is correctly classified.

This alternative formulation leads to the definition of the multiclass hinge loss for Neural Networks. Let the true class be represented by a vector $\tilde{Y}$ in $-1/1$ encoding. The multiclass hinge loss for a predicted output $\hat{Y}$ can then be defined as:

$$\mathcal{L}_{hinge}(\hat{Y}, \tilde{Y}) := \sum_{i=1}^{N_C} (max(0, 1 - \tilde{y}_i \hat{y}_i) \tag{3.8}$$

The loss is commonly preceded by a softmax activation when used for Neural Networks. Note that such a choice restricts the values of the outputs $\hat{y}_i$ to the interval $[0, 1]$, which has for consequence that the loss is still minimal at $\hat{Y} = \tilde{Y}$, but that this minimum is no more equal to 0. Therefore, one might want to redefine the loss using the transformation $\hat{z}_i = 2\sigma(\hat{y}_i) - 1$:

$$\begin{aligned}\mathcal{L}_{\sigma hinge}(\hat{Y}, \tilde{Y}) &:= \sum_{i=1}^{N_C} (\max(0, 1 - \tilde{y}_i(2\sigma(\hat{y}_i) - 1)) \\ &= \sum_{i=1}^{N_C} (\max(0, 1 - \tilde{y}_i \hat{z}_i))\end{aligned}$$

Squared and Cubed Hinge Losses The multiclass hinge loss is not differentiable around $\hat{z}_i = 1$. Therefore, one could take the square of each term of the sum, which gives the following loss:

$$\mathcal{L}_{hinge^2}(\hat{Y}, \tilde{Y}) := \sum_{i=1}^{N_C} (\max(0, 1 - \tilde{y}_i \hat{z}_i))^2 \tag{3.9}$$

In the same way, one can also define the cubed hinge loss.

As an illustration, the contribution of the term i of the sum that corresponds to the true label $\tilde{y}_i = 1$ for the hinge, squared hinge and cubed hinge losses are shown on Figure 3.2.

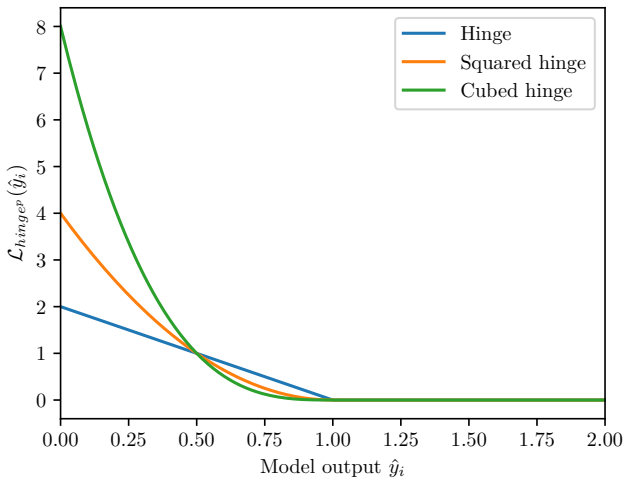


Figure 3.2: Contribution of the i-th term of the loss that corresponds to the true label $\tilde{y}_i = 1$ for the hinge, squared hinge and cubed hinge losses.

3.2 Backpropagation

The objective is now to minimize the output of the loss function by updating the weights of the neural network. This task is achieved by using backward propagation of the errors, or in short backpropagation.

Backpropagation is one of the most efficient learning procedures for multi-layer neural networks [52]. It uses a well chosen chain rule and a gradient descent to update the weights of the network in order to minimize the loss function. Like detailed in [71], the algorithm works by first computing the gradient of the loss function with respect to each weight of the last layer and then by iterating the same process backward layer-wise.

There are many ways to derive backpropagation. In the following, we will present an approach first introduced by [52] in which backpropagation is seen as an optimal control problem, using Lagrangian formalism.

Let us use the following notation: the state of layer l for an input feature vector x_p is denoted $H_l^{(p)} := [h_{l,1}^{(p)}, h_{l,2}^{(p)}, \dots, h_{l,N_l}^{(p)}]^T$ and the corresponding weights of the layer are denoted by $W_l := [w_{l,1}, w_{l,2}, \dots, w_{l,N_l}]^T$. The global state of the network (i.e. in all layers) for feature vector x_p is $H^{(p)} := [H_1^{(p)}, H_2^{(p)}, \dots, H_L^{(p)}]^T$ for $p = 1, \cdots, P$. We denote by H the concatenated $H^{(p)}$ for all feature vectors $p = 1, \cdots, P$. The inputs of layer l are denoted by $A_l^{(p)}$. Using these notations, we rewrite the forward propagation rule of a Multi-Layer Perceptron (recall Equation 2.2) as follows:

$$H_l^{(p)} = \sigma\left(W_l H_{l-1}^{(p)}\right) = \sigma\left(A_l^{(p)}\right) \tag{3.10}$$

From there, we show that backpropagation can be viewed as a constrained minimization problem involving not only the trainable weights W, but also the state H of the network. The Lagrange function [44] corresponding to the problem is the sum of the loss function and a constrained term which describes the structure of the Neural Network. The constrained term is multiplied by a so called Lagrange multiplier. The Lagrange function of the problem for a single feature vector x_p is given by:

$$L^{(p)}(W, H^{(p)}, B^{(p)}) = \mathcal{L}(Y^{(p)}, H_L^{(p)}) + \sum_{l=1}^{L} (B_l^{(p)})^T \left(H_l^{(p)} - \sigma\left(W_l H_{l-1}^{(p)}\right)\right) \tag{3.11}$$

where $B_l^{(p)}$ is a Lagrangian multiplier and $\mathcal{L}(Y^{(p)}, H_L^{(p)})$ is the loss of the output of the last layer L of the neural network. In the following, let us take the squared output error as loss function, i.e. $\mathcal{L}(Y^{(p)}, H_L^{(p)}) = (Y^{(p)} - H_L^{(p)})^T (Y^{(p)} - H_L^{(p)})$. Note similar results can be obtained for any diferentiable loss function [52].
When the constraints are met (i.e. when the constraint term is equal to zero), we have: $H_l = \sigma(W_l H_{l-1})$ for all $l \in \{1, \cdots, L\}$, which is nothing more than the layer propagation rule. Thus, the constraint term just defines the MLP forward dynamics. In that sense, the constraint term only defines the dependencies between the state of the network H for all input data-points and the network weights W.

The constrained minimization problem is to find weight values $W_1, W_2, \cdots, W_L$ that minimize the loss function $\mathcal{L}$ while satisfying the constraints. A necessary

condition for reaching a local minimum is $\nabla L(W, X, B) = 0$ or equivalently [52]:

$$\begin{cases} \frac{\partial L(W,H,B)}{\partial B} = 0 \\ \frac{\partial L(W,H,B)}{\partial X} = 0 \\ \frac{\partial L(W,H,B)}{\partial W} = 0 \end{cases} \tag{3.12}$$

Each condition can be decomposed into one condition per layer. Furthermore, after some developments, system 3.12 can be rewritten as:

$$\begin{cases} H_l^{(p)} = \sigma_l\left(W_l H_{l-1}^{(p)}\right) & \forall l \in \{1, \cdots, L\} \text{ and } \forall p \in \{1, \cdots, P\} \\ Z_l^{(p)} = \nabla\sigma_l\left(A_l^{(p)}\right) W_{l+1}^T Z_{l+1}^{(p)} & \forall l \in \{1, \cdots, (L-1)\} \text{ and } \forall p \in \{1, \cdots, P\} \\ \sum_{p=1}^{P} Z_l^{(p)} (H_{l-1}^{(p)})^T = 0 & \forall l \in \{1, \cdots, L\} \end{cases} \tag{3.13}$$

with boundary conditions:

$$\begin{aligned} H_1^{(p)} &= X^{(p)} \\ Z_L^{(p)} &= 2\nabla\sigma_L\left(A_L^{(p)}\right)\left(Y^{(p)} - H_L^{(p)}\right) \end{aligned}$$

where $Z_l^{(p)} = \nabla\sigma_l\left(A_l^{(p)}\right) B_l^{(p)}$ is a gradient variable and $X^{(p)}$ is the input vector of pattern p. The full development can be found in [52].

The equations of System 3.13 can be understood as follows:

- the first condition corresponds to the forward pass of the input vectors through the Neural Network
- the second condition describes the backward pass, i.e. the backpropagation of the gradient of the inputs. Indeed, it provides the chain rule for computing the gradient values $Z_l^{(p)}$
- the last condition gives an optimality condition on the weights' values W but does unfortunately not give a direct method for computing them. However, it can be shown [52] that minimizing the output loss function with respect to the weights W is equivalent to finding a minimum of the Lagrange function $L(W, H, B)$ while satisfying the constraints. If we do so with the method of steepest descent, the procedure takes the form:

  $$W_l \leftarrow W_l - \lambda \frac{\partial L(W, H, B)}{\partial W}$$

  Or equivalently:

  $$W_l \leftarrow W_l + \lambda \sum_{p=1}^{P} Z_l^{(p)} (H_l^{(p)})^T \tag{3.14}$$

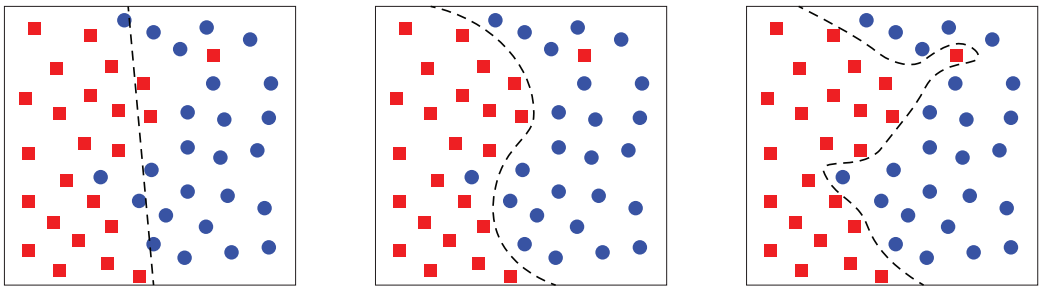
  where λ is the step size. The formulation of Equation 3.14 gives a weights update formula for backpropagation.

In that sense, we derived backpropagation of the gradients and an update rule for the weights as a constrained optimization problem in the Lagrange formalism.

In practice, the weights are updated using a stochastic gradient descent [29] (SGD), which can be regarded as a stochastic approximation of gradient descent method [86], as it only considers a random subset of the data points at a time. This allows for a quicker computation for large datasets since the gradients are computed over smaller samples.

3.3 Evaluation and Model Regularization

Up to now, we have seen how to train a Neural Network by optimizing a criterion called the loss function. In an ideal world, the trained model should perform very well both on the examples is has been trained with and on new examples. In practice however, it is difficult to train the model such that it performs well in both cases. Indeed, training data often includes noisy or misclassified points. Trying to perfectly fit these points would therefore poorly generalize for new data points. This is called overfitting. On the other side, when a model performs badly on both training and new data, the model is told to underfit. Figure 3.3 illustrates these two concepts on a single example of two classes classification. Figure 3.3a is underfitting since it tries to explain a complex boundary by a linear model whereas Figure 3.3c fits perfectly each point of the training set, which is likely to generalize poorly to new points.



(a) Underfitting **(b)** Compromise **(c)** Overfitting

Figure 3.3: Example of underfitting and overfitting for classification of two dimensional data points. The classes are labelled by red squares and blue circles and the decision boundary is represented by a dotted line.

To avoid such situations, it is therefore a good practice to test the Neural Network on labelled data points on which it has not been trained. This set of data points is called the test set. By doing so, one can hope to fairly evaluate the generalization capabilities of the trained model and verify if it is overfitting.

Furthermore, most models have a set of parameters called hyper-parameters that can not be tweaked during the training phase. Take a MLP as example. One needs to choose how many hidden layers it will have as well as the number of neurons per layer. Obviously, it seems difficult to change these parameters dynamically during the training phase. Therefore, the optimal value for these hyper-parameters is chosen by trying different values and evaluating the accuracy for each value. The

hyper-parameters leading to the highest accuracy are then taken. Like for the model evaluation, the accuracy has to be computed on data on which the model has not been trained. Indeed, the set of hyper-parameters values that will lead to the best generalization capabilities is often different from the one leading to the highest training accuracy. This set of labelled examples is called the validation set.

Suppose we have one set of labelled examples. A common practice to train, validate and evaluate a model on these examples is to use k-fold validation: first, shuffle the examples and then divide them into a training and a test set. Then, further divide the train set into k equally sized subsets. Take one of the k subsets as validation set and the $k-1$ remaining subsets as train set. Then, train the Neural Network for all combinations of these k equally sized subsets. The hyper-parameters are then fixed by taking the set of hyper-parameters that led to the best validation accuracy. The model is finally evaluated on the test set. The division of the labelled data points for 5-fold validation is illustrated on Figure 3.4. In this example, the data is divided in 10 equally sized sets, of which 5 are for testing and all combinations of the 5 others are used for the 5-fold validation of the hyper-parameters.

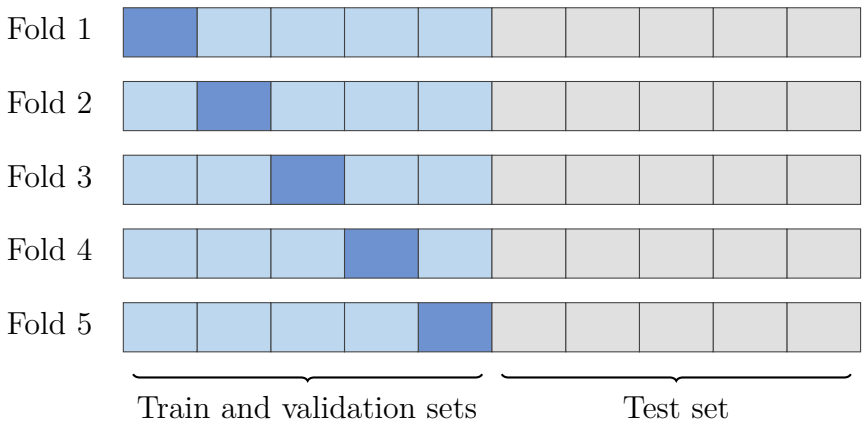


Figure 3.4: Example of 5-fold cross-validation. Each rectangle respresents a set of labelled samples. The training set is in light blue, the validation test in blue and the test set in grey.

From there, we are now able to check whether or not a model is overfitting. In the following sections, we will now present three techniques to avoid a model from fitting too much during the training phase.

3.3.1 Early stopping

Early stopping [68] works by stopping the training phase when the validation accuracy or validation loss begins respectively to decrease or to increase. The validation set is thus used to detect when the model begins to overfit. One way of implementing early stopping is the following. Fix a stopping window size S. After each full pass through the training data, compute the validation accuracy. If the mean validation accuracy is worse or the same as for the last S passes, stop the training phase. The evolution of the train and validation accuracy as well as the stopping epoch for a

MLP and a stopping window of size $S = 10$ is illustrated on Figure 3.5. The example suggests that after 125 epoch, the validation loss begins to increase while the train loss still decreases. This suggests that the model begins to overfit after around 125 epoch and that early stopping is in this case a good way to avoid overfitting.

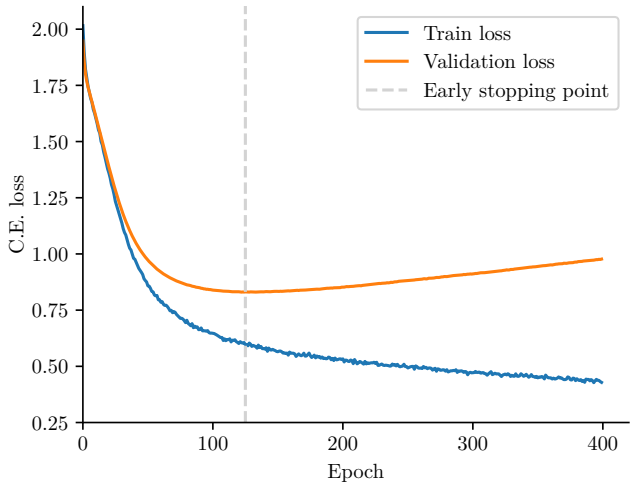


Figure 3.5: Evolution of the train and validation loss through 400 epoch of the training phase. The model is a MLP with one hidden layer and an early stopping window size 10, trained on Cora dataset using only the binary word vectors. More details about the dataset can be found in Section 5.1.

3.3.2 Weight Decay

Weight decay [30] is another regularization technique. The idea is to penalize the weights of the Neural Network that have large values by introducing a regularization cost to the loss function. Typical regularization techniques take the L1 or L2 norm of the NN weights. The loss for L^p regularization of the set of model weights $\mathcal{W}$ is given by:

$$\begin{aligned}\mathcal{L}(Y, \hat{Y}; \mathcal{W}) &= \mathcal{L}_{base}(Y, \hat{Y}) + \gamma_{\mathcal{W}} \mathcal{L}_{regu}(\mathcal{W}) \\ &= \mathcal{L}_{base}(Y, \hat{Y}) + \alpha \|\mathcal{W}\|_p^p\end{aligned}$$

where $\gamma_{\mathcal{W}}$ is the weight decay parameter.

The motivation behind weight decay is to decrease the complexity of the Neural Network by limiting the growth of its weights. Indeed, it should prevent the weights from growing too large unless it is really necessary. Paper [50] suggests it has two effects. On the one hand, it suppresses any irrelevant components of the weights by constraining the choice to the smallest vector that solves the learning problem. On the other hand, it could suppress part of the effect of statistic noise on the model, which will improve the generalization capabilities of the Neural Network.

3.3.3 Dropout

Deep Neural Networks like CNNs often suffer from overfitting. A common regularization technique on such types of networks is to randomly drop a given proportion of the neurons during the training phase. According to [80], this prevents the weights from co-adapting too much since dropout has the effect of bypassing part of the weights during the learning phase. Technically speaking, dropout is applied as follows. Drop randomly a proportion p of the weights of the network at each forward pass of the learning phase. Typically, inputs not dropped are scaled up by a factor of $\frac{1}{1-p}$ such that the sum over the layer input remains unchanged. All the neurons are used during the evaluation phases. Dropout with a rate of $p = 4/9$ for the MLP of Figure 2.1 is illustrated on Figure 3.6.

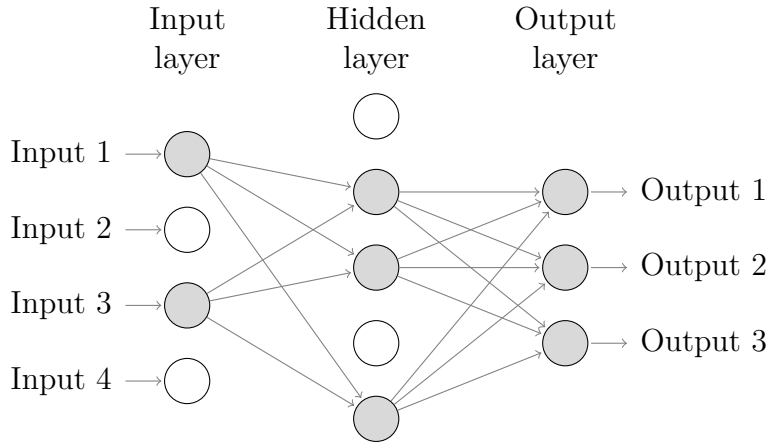


Figure 3.6: Dropout with a rate of $p = 4/9$ on a MLP with one hidden layer. The neurons with a white background have been dropped.

Chapter 4

Graph Node Classification

Now that we have defined what a Neural Network is and how to train such a model, let us introduce the notion of semi-supervised classification on graph-structured data and the models to perform this task. First, we define the notion of graph and present basic tools and properties of graphs. Then, we present four models based on the convolution operation as well as a model derived from the Support Vector Machine. Finally, we will investigate how graph regularization can help in the learning process of these Graph Neural Networks [74].

4.1 Graph Theory and Fourier Transform

In the nature, humans or objects are sometimes linked by pairwise relations. A convenient way to represent these relations is to draw links between the objects. Such a mathematical structure is called a graph, and is studied by the field of graph theory [66, 34, 84]

In this section, we will first see the definition of a graph and the different ways to represent it. Then, we will investigate important properties of graphs. Formally, an undirected graph, or simply a graph [84], is defined as an ordered set $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where:

- $\mathcal{V}$ is a set of nodes
- $\mathcal{E} \subseteq \{\{v_i, v_j\} \mid v_i, v_j \in \mathcal{V}\}$ is a set of edges

The degree of the node is the number of edges connected to the node. A self-loop counts for two in the degree. The definition of a graph can be extended by adding a weight to each edge. Such weights might for example represent a distance or model the strength of the relation between two connected edges. The degree of a node is then the sum of the weights of its edges.

Let us take an example to illustrate this definition. Consider the cities of Belgium and the roads between these cities. This could be represented as a graph in which the cities are the nodes and each road between two of these cities is represented as an edge [10]. A weight can be added to the edges to model the duration it takes to travel from one city to another.

Note that the edges are defined by an unordered set of two nodes, which means that the node relations are symmetrical. A graph in which the edges have a direction is called a directed graph and the set of edges is then defined by: $\mathcal{E} \subseteq \{(v_i, v_j) \mid v_i, v_j \in \mathcal{V}\}$, where (v_i, v_j) is an ordered pair.

In the following sections, consider a graph that is undirected, has no self-loops and no multiple edges. Which means that the set of edges is unordered, with no duplicates and with no edge that go from one node to the same node. Such a graph is called a simple graph. Furthermore, consider the graph to be connected, which means there exists a path between any pair of nodes. The graph of Belgian cities is a connected simple graph if there exists at most one direct road between any pair of cities, there are no one-way roads and if one could always find a path of roads from one city to another.

A graph can be fully defined by a matrix called adjacency matrix [84]. An adjacency matrix A is a square matrix of size $N_\mathcal{V} \times N_\mathcal{V}$ in which the value of element ij is the number of edges between node $v_i \in \mathcal{V}$ and node $v_j \in \mathcal{V}$. It is a mathematically convenient way to represent the graph structure. Note that adjacency matrix of an undirected graph is symmetrical since if there exists a path between node v_i and node v_j then there also exists a path between node v_j and v_i. Another convenient matrix representations of the graph is the graph Laplacian matrix [84]. For a simple graph, the Laplacian matrix is defined by $L := D - A$ where A is the adjacency matrix and the degree matrix D is a diagonal matrix such that $D_{ii} = \deg(v_i) = \sum_j a_{ij}$ for all $v_i \in \mathcal{V}$.

The following example shows the different but equivalent representations of a simple graph: let the graph defined by: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{v_i | i = 1, 2, \cdots, 6\}$ and $\mathcal{E} = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}\}$. Its adjacency and Laplacian matrices are given by:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } L = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

Graph $\mathcal{G}$ is illustrated on Figure 4.1. This graph is simple but not connected, since there are no paths to node v_6. If the graph is weighted, the adjacency matrix takes then the form $A \in \mathbb{R}^{N_\mathcal{V} \times N_\mathcal{V}}$ where each entry is either a real-valued weight or zero if there is no edge. The degree of a node i is then the sum of the weights of its edges: $\deg(i) = \sum_{j \in \mathcal{V}} A_{ij}$.

Graph node classification models will rely on properties of the adjacency matrix and the graph Laplacian matrix. Let us introduce the main properties.
Recall the definition of the unweighted adjacency matrix: entry a_{ij} encodes the number of paths of length one between node v_i and node v_j. One can show that the powers of the adjacency matrix share a similar property:

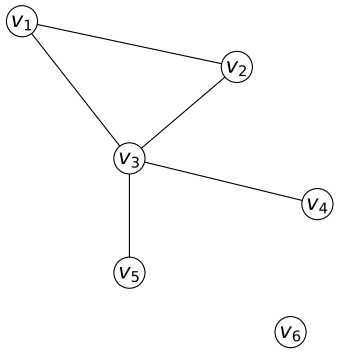


Figure 4.1: Representation of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{v_i | i = 1, 2, \cdots, 6\}$ and $\mathcal{E} = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}\}$.

Theorem 4.1.1 *The $(i, j)^{th}$ entry of the k-th power of the adjacency matrix A^k counts the number of walks of length k having start and end vertices i and j respectively. Proof in [23].*

This Theorem gives a straightforward way of transmitting information between nodes to a certain distance in the graph. It will be the foundation of the Diffusion Convolutional Neural Network that will be introduced in Section 4.3.3.

Consider now a simple graph $\mathcal{G}$ with n nodes on which we define a graph signal or feature $x : \mathcal{V} \to \mathbb{R}$. The graph signal is a function that maps every node of graph $\mathcal{G}$ to a real number. One will define the Graph Fourier Transform $\hat{x} = \mathcal{F}(x)$ as the projection of a signal on the eigenvectors of the Laplacian matrix L. Similar to the classical Fourier transform, the graph Fourier transform provides an equivalent representation of the signal in a graph spectral domain [15].

Take the eigenvalue decomposition of the Laplacian matrix. Since the matrix is symmetrical, it can be expressed by:

$$L = U \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) U^T$$

where U is a unitary matrix with the eigenvectors of L and $\text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix with the eigenvalues ordered in ascending order: $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

From this decomposition, the graph Fourier transform [73] of a signal f on graph $\mathcal{G}$ is defined as the projection of a signal on the eigenvectors of the Laplacian matrix:

$$\hat{x}(\lambda_i) = \mathcal{F}(x(\lambda_i)) = \langle x, u_i \rangle = \sum_{k=1}^{n} x(v_k) u_i(k)$$

where $u_i(k)$ is the k-th entry of the i-th eigenvector of the Laplacian matrix L. The inverse graph Fourier transform is defined by:

$$x(v_i) = \mathcal{F}^{-1}(\hat{x}(v_i)) = \sum_{k=1}^{n} \hat{x}(\lambda_k) u_k(i)$$

As previously stated, the graph Fourier transform provides an equivalent representation of the signal in a so called graph spectral domain. In that sense, the

eigenvalues of the Laplacian matrix can be seen as the frequencies of the graph. For seek of intuition about the graph Laplacian matrix, let us write $x_i = x(v_i)$ and consider the following quantity:

$$\begin{aligned} x^T L x &= \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} A_{ij} (x_i - x_j)^2 \\ &= \frac{1}{2} \sum_{\{i,j\} \in \mathcal{E}} A_{ij} (x_i - x_j)^2 \end{aligned}$$

The development highlights that $x^T L x$ quantifies the local variations of signal x on the edges of the graph and that the Laplacian matrix L is positive semi-definite for any $x \neq 0$, which means all the eigenvalues of the Laplacian matrix are nonnegative. By a similar development, we have $\sum_{j \in \mathcal{V}} L_{ij} x_j = \sum_{j \in \mathcal{V}} A_{ij}(x_i - x_j)$ for all $i \in \mathcal{V}$. Since an eigenvector of L is a vector for which the quantity $y_i = \sum_{j \in \mathcal{V}} L_{ij} x_j$ is the same for any node i, one can interpret the eigenvectors of the Laplacian matrix as values for which the variation is constant on all nodes of the graph, which can therefore be interpreted as graph frequencies.

Finally, the constant vector $\mathbb{1}$ is an eigenvector of the Laplacian matrix associated with eigenvalue 0 since it is a signal for which there are no variations between edges[1]. From there, the smallest eigenvalue is $\lambda_1 = 0$ (since we have shown that the eigenvalues are non-negative) and its multiplicity is the number of connected components of the graph [73].

Now that we have defined graphs and that we showed some of their properties, we can introduce the task of semi-supervised classification on graph-structured data.

4.2 Problem statement and autocorrelation

Semi-supervised graph node classification is defined as follows: let a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N_\mathcal{V}$ nodes and a set of feature vectors X_i of size N_F defined on the nodes of the graph. The features are stored in a $N_\mathcal{V} \times N_F$ matrix denoted by X. Typically, the features will be binary or real-valued features. The task of semi-supervised node classification consists of predicting the labels Y of all the nodes of the graph, knowing a small subset $Y_{labelled} \subset Y$ of labels as well as the full graph and set of features. That is, finding a function $\hat{f} : \mathcal{V} \to \{1, ..., N_{labels}\}$ that maps each node of the graph to a label, with the highest possible accuracy.

The motivation behind the use of the graph structure information is that it could improve the accuracy of the classification models by providing useful information in addition to the one provided by the node features. From this idea, many models [47, 87, 5] rely on the assumption that neighbouring nodes are more likely to belong to the same class [27]. This hypothesis is often referred as autocorrelation [88]. Autocorrelation is high when nodes with similar labels are clustered together.

Moran's I [61] is a statistical test for which the null-hypothesis states there is no autocorrelation or homophily in a given graph. Moran's I for a given quantity x on

[1]Proof: $[L\mathbb{1}]_i = \sum_{j \in \mathcal{V}} A_{ij}(1 - 1) = 0$ for any $i \in \mathcal{V}$

the graph is a value in the interval $[-1, 1]$ defined as [61]:

$$\frac{N_{\mathcal{V}}}{\sum_{i=1}^{N_{\mathcal{V}}}\sum_{j=1}^{N_{\mathcal{V}}} A_{ij}} \frac{\sum_{i=1}^{N_{\mathcal{V}}}\sum_{j=1}^{N_{\mathcal{V}}} A_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_{i=1}^{N_{\mathcal{V}}}(x_i - \bar{x})^2} \tag{4.2}$$

where $\bar{x}$ is the mean of quantity x. Moran I expected value under the null-hypothesis is: $\mathbb{E}[I] = \frac{-1}{N_{\mathcal{V}}-1} \approx 0$, which is very small. The null hypothesis states that the data is randomly distributed, that is when $I(x) \neq \mathbb{E}[I] \approx 0$. Larger values means autocorrelation is high and lower values mean there is negative autocorrelation. In this work, we compute autocorrelation on the binary class indicators as the average Moran I over the different classes:

$$\bar{I} = \frac{1}{N_C}\sum_{c=1}^{N_C} I(y_c)$$

In the following sections, we will first present three models that propagate the node features to neighbouring nodes through the use of well chosen generalized convolution operations. Then, we will suggest an improvement of one of the three models, namely the Graph Wavelet Neural Network. We will also present an adaptation of the SVM that takes graph information into account. Finally, we will present a regularization technique that aims to improve the learning process by penalizing predictions for which neighbouring nodes are predicted to have different labels.

4.3 CNN-Based Models

In the last few years, Convolutional Neural Networks (CNNs) have outperformed the other methods on the task of image classification [31, 49]. Like we have seen in Section 2.2.2, the strength of a CNN is that the convolution operation allows to learn patterns while reusing the same kernel weights for all the nodes of a given channel, which highly reduces the complexity of the model. Furthermore, we have shown that the convolution operation can be seen as a diffusion of node information to neighbouring nodes. In that sense, convolution seems well suited for graph node classification under the autocorrelation hypothesis since neighbouring nodes are more likely to belong to the same class when autocorrelation is high [27]. Spreading their information to neighbours in a trainable way could therefore improve the learning abilities of the models.

Multiple attempts have been made to build CNN-like models on graphs by generalizing the convolution operation to any graph. This generalization step is not straightforward since graphs in general do not have a fixed number of neighbouring nodes, which makes the use of a unique kernel of fixed size not possible. Two families of approaches exist to overcome this issue: the spatial and the spectral methods. Spatial methods try to still work in the graph domain by rewriting the convolution operation in a way that allows to keep the weight sharing property. Whereas spectral methods define the convolution operation in the spectral domain of the graph, which solves the neighbours issue since the Fourier domain of a graph is continuous. These

models will also take advantage of the convolution theorem [59], which states that a convolution of two signals in the spatial domain is equivalent to a product of the signals in the Fourier domain.

Let us present three of these models: one is spatial, one is spectral and one is in-between.

4.3.1 Graph Convolutional Network (GCN)

The propagation rule of the GCN model is based on a 1st-order approximation of the spectral graph convolution [47]. Working in the spectral domain instead of the spatial domain allows to define a kernel that maintains the weight-sharing property of CNNs as it is no more defined on the nodes of the graphs (i.e. graph spatial domain).

Approximation of the spectral graph convolution

The spectral graph convolution, or graph Fourier convolution, of a signal $x \in \mathbb{R}^{N_V}$ with a diagonal kernel $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^{N_\mathcal{V}}$ in the Fourier domain is given by:

$$g_\theta * x = U(g_\theta(U^T x)) \tag{4.3}$$

where U an Λ are respectively the eigenvectors and eigenvalues of the normalized Laplacian matrix $L' = I_N - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. Using the properties of the Laplacian matrix of a graph (recall Section 4.1), $U^T x$ is the graph Fourier transform of x and g_θ can be understood as a function of the eigenvalues of L'. The spectral domain is thus the space spanned by the eigenvectors of the graph Laplacian matrix L'.

Unfortunately, the computation of the eigendecomposition of L' and the application of matrix multiplications with U are expensive operations. An approximation of the kernel in the spatial domain, suggested by [36] is to use a truncated expansion in terms of Chebyshev polynomials[2] $T_k(x)$:

$$g_{\theta'} \approx \sum_{k=0}^{K} \theta'_k T_k\left(\widetilde{\Lambda}\right) \tag{4.4}$$

where $\theta' \in \mathbb{R}^{N_\mathcal{V}}$ is a vector of Chebyshev coefficients, K is the order of approximation and with $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$ where λ_{max} is the largest eigenvalue of L'.

By further limiting the approximation to $K = 1$ and by taking $\lambda_{max} \approx 2$ (since the neural network parameters will adapt to this change of scale during the training phase), equation 4.3 can be approximated by

$$\begin{aligned} g_{\theta'} * x &\approx \sum_{k=0}^{K} \theta'_k T_k\left(\tilde{L'}\right) x \\ &\approx \theta'_0 x + \theta'_1 \left(L' - I_N\right) x \\ &= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x \end{aligned}$$

[2]Chebyshev polynomials $T_k(x)$ are defined by the recursion: $T_k(x) = 2xT_{k-1} - T_{k-2}(x)$ with initial conditions $T_0(x) = 1$ and $T_1(x) = x$.

Note that this expression depends on nodes that are K steps away from the central node (since it is a K^{th}-order polynomial in the Laplacian). Taking $K = 1$ does therefore limit the approximation to the 1st-order neighbours of the central node and gives an operation that is linear w.r.t. the normalized Laplacian L'. This 1st-order approximation still allows to build a rich class of convolutional filter functions by stacking multiple layers of this kind with an added nonlinear activation function at the output of the layer. The authors of [47] justify this choice by the fact that it could alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions and by that it allows to build deeper networks for a same computation cost, which commonly improves performance on CNN models.

To further address overfitting issues, the model is simplified by limiting the number of weights using $\theta = \theta'_0 = -\theta'_1$, which finally leads to:

$$g_{\theta'} * x \approx \theta \left[I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right] x \tag{4.5}$$

A final issue is that the matrix $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ has eigenvalues in $[0, 2]$, which could lead to numerical instabilities when using a deep NN. This issue is solved by the following renormalization trick: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ with $\tilde{A}$ is the adjacency matrix with added self loops and $\tilde{D}$ is the degree matrix of $\tilde{A}$. The final operation is then defined by:

$$g_{\theta'} * x \approx \theta \left[\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right] x \tag{4.6}$$

The GCN model

The form given in Equation 4.6 can be generalized to the full set of feature vectors $X \in \mathbb{R}^{N_\mathcal{V} \times N_\mathcal{F}}$ by rewriting it: $X_{new} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$ where $\Theta \in \mathbb{R}^{N_\mathcal{F} \times N_{new}}$ are the filter parameters. This operation will be the basis for the information propagation. From there, the GCN layer is defined by:

$$H_{l+1} = \sigma \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H_l W_l \right) \tag{4.7}$$

where $W_l \in \mathbb{R}^{N_l \times N_{l+1}}$ are the filter parameters of layer l, σ is an activation function and the input of the first layer is the features matrix: $H_0 = X$. An illustration of the network structure is given on Figure 4.2.

For seek of intuition, each GCN layer can be seen as a new representation of the node features. The new sets of features are built by spreading the features to the neighbouring nodes and then taking a well chosen transformation of the resulting set of features. The kernel size is thus independent of the number of nodes since is transforms the set of features of each node in the same way.

As a final point, let us discuss the nature of GCN. The Graph Convolutional Network is a model that is built on the idea of defining the convolution (propagation) operation in the graph spectral Fourier domain, it can therefore be seen as a spectral method. However, the Fourier and inverse Fourier transforms are two expensive operations. Therefore, the model relies on a first order approximation of the kernel

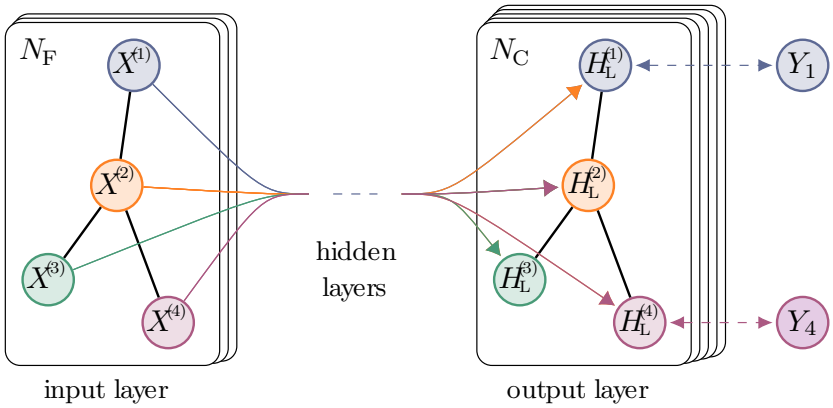


Figure 4.2: Illustration of the GCN model on a graph composed of four nodes, N_F features and N_C output classes. Figure adapted from [47].

directly in the spatial domain, which induces that GCN could also be seen as a spatial method, since the layer propagation rule of Equation 4.7 is defined only in the spatial domain.

Model Complexity

Scalability of Graph Neural Networks is an important factor to take into account when designing a model. Indeed, many graphs involve thousands of nodes. Any model of practical use should therefore have a reasonable computation cost even for large graphs as well as a number of weights that is smaller than the size of the adjacency matrix (i.e. than the square of the number of edges). The most expensive operations are often matrix multiplications with the adjacency matrix. Fortunately, most graphs have a low average node degree and the operations can therefore be implemented as a sparse matrix multiplication.

GCN layer propagation rule involves two operations: a matrix multiplication between the renormalized adjacency matrix and the features set and then the filtering operation. Since the renormalized matrix $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ has $N_\mathcal{V} + 2N_\mathcal{E}$ nonzero elements and since the filtering operation involves matrices of size $N_\mathcal{V} \times N_{l,in}$ and $N_{l,in} \times N_{l,out}$, the computation cost is:

$$\mathcal{O}\left((N_\mathcal{E} + N_\mathcal{V})N_{l,in}\right) + \mathcal{O}\left(N_\mathcal{V} N_{l,in} N_{l,out}\right) = \mathcal{O}\left((\bar{d}+1)N_\mathcal{V} N_{l,in} + N_\mathcal{V} N_{l,in} N_{l,out}\right)$$

where $\bar{d}$ is the average node degree. Since the layers sizes are typically decreasing: $N_f = N_{1,in} > N_{1,out} = N_{2,in} > \cdots > N_{L,out} = N_c$ and by assuming that the average node degree is smaller than the number of features, the time complexity of the layer propagation is finally reduced to:

$$\mathcal{O}\left(N_\mathcal{V} N_f^2\right)$$

Each layer has a filtering matrix W_l of size $N_l \times N_{l+1}$. By taking the same hypothesis as for the time complexity, the number of weights of a L layers GCN is

bounded by the quantity LN_f^2, which is independent of the number of nodes. This means that like for the convolution on images, the convolution kernel of GCN is independent of the number of input nodes.

4.3.2 Graph Wavelet Neural Network (GWNN)

The GWNN model does, like GCN, generalize the convolution operation by working in the spectral domain. However, unlike GCN, the convolution is this time performed in the wavelet domain [64, 69, 14]. This trick allows to drastically reduce the transform step since graph wavelet transform is sparse, whereas the Fourier transform is dense [87]. By doing so, it is possible to work in the spectral domain without needing to make huge approximations like GCN.

Graph Wavelet Transform

The graph wavelet transform employs a wavelet basis, defined by a set of wavelets: $\phi_s = \{\phi_{s1}, \phi_{s2}, \cdots, \phi_{sn}\}$. Wavelet element ϕ_{si} corresponds to a signal on the graph, diffused away from node i at a given scale s.

Because of the discrete nature of a graph, the set of possible scales would be restricted to the set of natural numbers. Indeed, the distance between two nodes of an unweighted graph only takes integer values. To overcome this limitation, Wavelets ϕ_s are defined in the Fourier domain [36]:

$$\phi_s = U G_s U^T \tag{4.8}$$

where U is the matrix formed by the eigenvectors of the graph Laplacian matrix and G_s is a diagonal scaling matrix, also called wavelet kernel matrix. In [87], the scaling matrix is chosen to be $G_s = \text{diag}\left(e^{\lambda_1 s}, e^{\lambda_2 s}, \cdots, e^{\lambda_{N_\mathcal{V}} s}\right)$. However, this is not the only possibility and other design choices will be discussed in Section 4.4.

Using this wavelet basis, the graph wavelet convolution of a signal x by a filter g_θ is defined by:

$$g_\theta * x = \phi_s(g_\theta(\phi_s^{-1}x)) \tag{4.9}$$

This operation is highly localized in the vertex domain since each wavelet corresponds to a signal on graph diffused away from a central node [87].

The GWNN model

Now that the graph wavelet convolution has been properly defined, it is possible to define the layer propagation rule. The output of layer $l+1$ is given by:

$$H_{l+1}^{[:,j]} = \sigma\left(\phi_s \sum_{i=1}^{N} W_l^{i,j} \phi_s^{-1} H_l^{[:,i]}\right) \quad j = 1, \cdots, N_l \tag{4.10}$$

where ϕ_s^{-1} is the graph wavelet transform matrix at scale s, $H_l^{[:,j]}$ is the j-th column of layer l (i.e. the j-th feature of each node at layer l), $W_l^{i,j}$ is a diagonal kernel matrix in spectral domain and σ is an activation function.

Each layer is thus composed of $N_l \times N_{l+1}$ diagonal kernel matrices to train, which leads to $N_\mathcal{V} \times N_l \times N_{l+1}$ weights per layer. Such a large number of parameters requires typically huge training data to learn the weights without overfitting. To avoid this issue, [87] suggests to divide the layers in two parts: the feature transformation and the graph convolution. This idea leads to the following formulation:

$$H_{l+1} = \sigma\left(\phi_s W_l^c \phi_s^{-1}(H_l W^d)\right) \tag{4.11}$$

where $W^c \in \mathbb{R}^{N_\mathcal{V} \times N_\mathcal{V}}$ and $W^d \in \mathbb{R}^{N_l \times N_{l+1}}$ are respectively the diagonal kernel and the feature transformation matrix. Indeed, W^d allows to map the number of features of layer l from the input size N_l to the output size N_{l+1}. By splitting the computation in two parts (see Equation 4.11), the total number of weights to train is reduced to $N_\mathcal{V} + N_l \times N_{l+1}$.

The successive layers of GWNN have the same structure as in the GCN model and can therefore again being illustrated by Figure 4.2. The Graph Wavelet Neural Network is similar to GCN in the sense both models build successive representations of the node features through the application of a convolution which is defined in the spectral domain. The model differs from GCN in the way it handles the computation cost. GCN reduces the computation cost by taking an approximation of the kernel in the spatial domain whereas GWNN kernel is truly defined in the wavelet domain while still having an affordable computation cost thanks to the high sparsity of the wavelet basis. Let us now take a closer look at the complexity of GWNN.

Model complexity

The layer propagation rule of GWNN involves four steps: a feature transformation, a diagonal filtering operation in the wavelet domain, the wavelet transform and inverse wavelet transform operations. Let us suppose the matrices of the wavelet transform and inverse transform have a number of nonzero elements bounded by a certain value $\gamma \in \mathbb{R}$ such that $\gamma N_\mathcal{V} \ll N_\mathcal{V}^2$, then the transforms can be implemented as sparse matrices multiplications and the total layer complexity is given by:

$$\mathcal{O}\left(N_\mathcal{V} N_f^2\right) + 2\mathcal{O}\left(\gamma N_\mathcal{V} N_f\right) + \mathcal{O}\left(N_\mathcal{V} N_f\right)$$

Which simplifies to:

$$\mathcal{O}\left(N_\mathcal{V} N_f (N_f + \gamma)\right)$$

This shows that the time complexity bounds of one layer of GCN and GWNN are the same when the sparsity level of the wavelet transform is such that $\gamma \approx N_f$. However, the computation time of GWNN is in general higher than GCN because it has higher hidden constants and because there are datasets for which the sparsity value γ is high.

The number of learnable weights of GWNN is bounded by $L(N_\mathcal{V} + N_f^2)$, which is higher than for GCN. Note that while not being independent of the number of nodes anymore, the kernel size grows only linearly with the number of nodes.

4.3.3 Diffusion Convolutional Neural Network (DCNN)

Unlike the two previous models, DCNN redefines the convolution operation directly in the graph (spatial) domain. The propagation rule of the DCNN model [5] is based on a so called "diffusion-convolution" operation using a diffusion tensor, which is built using power series of the probability transition matrix of the graph. Intuitively, this tensor can be thought as a measure of the connectivity between nodes when taking the paths of all lengths, but with shorter paths counting for more [25, 5].

The model works in two steps. Firstly, the diffusion-convolution operation builds multiple representations of the features by spreading the node features to neighbouring nodes at different distances. Then, a fully-connected layer classifies the nodes using the multiple feature representations as input.

Diffusion convolution

Let P be the degree-normalized transition matrix defined by [84]: $P_{ij} = \frac{A_{ij}}{D_{ii}}$. Entry ij gives the probability of jumping from node i to node j in one step. From there, let P^* be a $N_\mathcal{V} \times N_H \times N_\mathcal{V}$ tensor containing the power series of the degree-normalized transition matrix:

$$P^*_{ihj} = P^h_{ij} \tag{4.12}$$

Intuitively, the power series tensor P^* encodes the probabilities of jumping from a node to any other node in any number of steps between 1 and N_H steps (proof is straightforward from the definition of P and Theorem 4.1.1). In addition, the 0-th power $P^0 = I_{N_\mathcal{V}}$ will be encoded in the tensor so that node also takes its own features into account.
This tensor will be the basis for the diffusion in the graph domain.

The DCNN model

From this simple diffusion kernel, the diffusion convolutional layer is defined by [5]:

$$H_1 = \sigma\left(W^c \odot P^* X\right) \tag{4.13}$$

where the $\odot$ operator represents element-wise multiplication, W^c is a weight matrix of size $N_H \times N_F$ and σ is an activation function. Note that this diffusion kernel allows to maintain the weight-sharing property of CNNs. Indeed, the same weight matrix W^c of the layer propagation rule defined in Equation 4.13 is applied to each node of the graph. Its size is thus independent of the number of nodes in the graph.

This layer builds a latent representation H_2 of the graph of size $N_\mathcal{V} \times N_H \times N_F$. This representation can be seen as a new set of features for each node, inherited from its neighbouring nodes and weighted through weights W^c. Indeed, the representation gives a set of $\times N_H \times N_F$ features for each node. Then, the neurons of the representation are flattened and mapped to the output classes using a MLP layer with weights $W^d \in \mathbb{R}^{(N_H N_F) \times N_C}$:

$$\hat{Y} = \sigma\left(H_1 W^d\right) \tag{4.14}$$

The DCNN model structure is illustrated on Figure 4.3. In comparison with the two previously introduced models, DCNN builds a latent representation of the features that encodes the graph structure, whereas the successive layers of GCN and GWNN are composed of graph structured data. The power-series approach has the advantage of needing only one convolutional layer for the building of the latent representation since it is propagates the node features at different distances from 0 to H in just one convolution. In comparison, we have seen that GCN only propagates features to the one-hop neighbouring nodes and it therefore needs $N_L = H$ successive layers to propagate features to the same distance as DCNN.

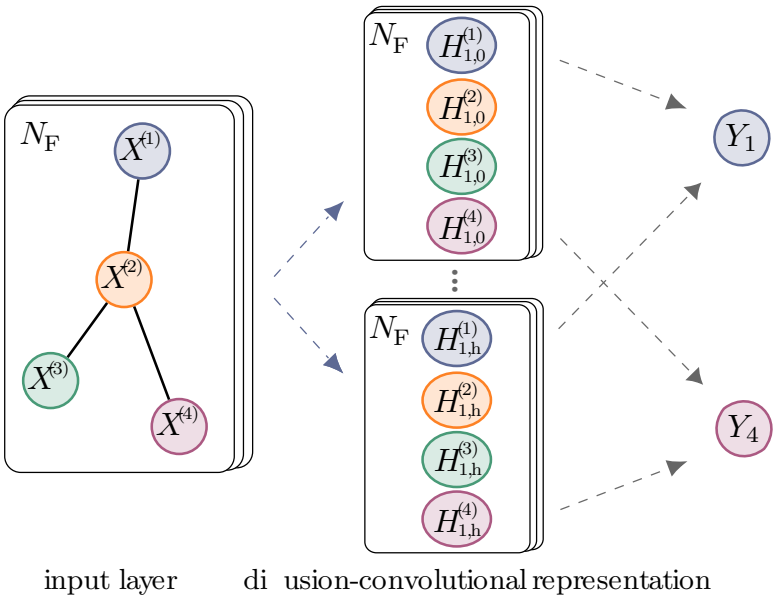


Figure 4.3: Illustration of the DCNN model on a graph composed of four nodes, N_F features and power series of P from 0 to H. Blue and grey arrows represent respectively the diffusion-convolution operation and the MLP operation.

Figure 4.3 shows that the convolution operation has a very similar structure to the one of a Convolutional Neural Network. Therefore, one could see the latent representation of the features as a layer with one channel per hop. From there, the diffusion convolution operation can be rewritten as:

$$H_{1,h} = \sigma\left(W_h^c \odot P^h X\right) \tag{4.15}$$

where $H_{2,h}$ is the channel h of the latent representation of the graph features. From there, the fully connected layer is:

$$\hat{Y} = \sigma\left(\sum_{h=0}^{N_H} H_{1,h} W_h^d\right) \tag{4.16}$$

Note that this formulation of the two layers is equivalent to the previously introduced one and will therefore give the same model output. However, this formulation allows to implement the matrix multiplications with the power series of the degree-normalized transition matrix in a sparse way, which reduces drastically the computation cost. Let us now derive the computation cost for this formulation.

Model Complexity

The Diffusion Convolutional Neural Network builds a latent representation of the graph features and then classifies using a fully connected layer. Which means that a difference with the previous models is that only one convolutional layer is needed.

Building the latent representation involves one sparse matrix multiplication and an element-wise matrix multiplication for each hop. If we further assume that the number of paths of length h is proportional to $N_{\mathcal{E}}$ for small values of h, then the total complexity of this layer is given by:

$$(N_H + 0)\mathcal{O}\left(N_{\mathcal{E}} N_f\right) + (N_H + 1)\mathcal{O}\left(N_{\mathcal{V}} N_f\right) \tag{4.17}$$

since the computation of $P^0 X = X$ is straightforward.

The fully connected layer performs $N_H + 1$ sparse matrix multiplications and then adds the resulting matrices. Its time complexity is therefore bounded by:

$$(N_H + 1)\mathcal{O}\left(N_{\mathcal{V}} N_f N_c\right) + \mathcal{O}((N_H + 1) N_{\mathcal{V}} N_c) = \mathcal{O}\left((N_H + 1) N_{\mathcal{V}} N_f^2\right) \tag{4.18}$$

From the previous results, one can bound total time complexity of the model by:

$$\mathcal{O}\left(N_H N_{\mathcal{V}} N_f^2\right) \tag{4.19}$$

It is interesting to notice that this upper bound is the same as the one of an L layers GCN.

The model has two weight matrices W^c and W^d of respective sizes $(N_H + 1)N_f$ and $(N_H + 1)N_f^2$, so that there are $(N_H + 1)(N_f + N_f^2)$ trainable weights in total.

4.4 Multiscale Graph Wavelet Neural Network (M-GWNN)

The Graph Wavelet Neural Network presented in Section 4.3.2 performs the convolution operation in a certain wavelet space, which depends on the choice of the diagonal scaling matrix G_s and on the scale s at which this transformation is defined. Paper [87] suggests using $g_s(\lambda) = e^{-\lambda s}$ as scaling function defined in the graph Fourier domain, which therefore acts as a low-pass filter on the signal and with a downsampling factor controlled by scale s. One could ask ourselves if combining multiple scales could improve the model and if a low-pass filter is the best choice for such a model or if other filters might be more suitable choices for the classification task. In the following, we first present a clever structure choice that allows to work at multiple scales. Then, we investigate other scaling function choices.

4.4.1 Multiscale Design

Taking inspiration from the Network structure of CNNs, the Multiscale Graph Wavelet Neural Network (M-GWNN) is designed to build successive layers with multiple channels. Each channel is associated with a different scale. In comparison

with classical CNNs, each channel encodes information depending on its associated kernel. Here, the kernels are each designed in a different space which depends on the scale associated with the channel. Which means each channel will encode different scale dependent information about the features. The general layer propagation rule of a M-GWNN with N_S scales is defined as follows:

$$H_{l+1,s} = \sigma\left(\sum_{s'=1}^{N_S} \phi_s W_{l,s}^c \phi_s^{-1}(H_{l,s'} W_{l,s'}^d)\right)$$

The computation cost and the number of weights can be drastically reduced by building successive channels that depend only on the channel of previous layer which is designed at the same scale. The model simplifies to:

$$H_{l+1,s} = \sigma\left(\phi_s W_{l,s}^c \phi_s^{-1}(H_{l,s} W_{l,s}^d)\right) \tag{4.20}$$

Like for GCN and GWNN, the layers build successive sets of features for each node, with a final layer of the size of the number of classes N_C. However, the Multiscale needs one more layer to combine the predictions of the different channels of the last layer. This layer takes the following form:

$$\hat{Y} = \sigma\left(\sum_{s=1}^{N_S} \alpha_s H_{L,s}\right)$$

where each $\alpha_s \in \mathbb{R}$ is a single combination weight. This formulation allows to diffuse the features at multiple scales and then combine the different class predictions through a small number of trainable weights.

4.4.2 Better Scaling Functions

The scaling functions suggested in [87] act like a low-pass filter since these scaling functions $g_s(\lambda) = e^{-\lambda s}$ are defined in the Fourier domain, which means they decrease the effect of high frequencies exponentially. The strength of this effect is controlled by the scaling parameter s, Therefore, high scales will act as a discriminant filter whereas the smallest scales will tend to act like no filter is applied (i.e. like a band-pass filter with the full spectrum in the pass frequencies). The effect of the scaling parameter for different scales ranging from 4^{-4} to 1 are displayed on Figure 4.4.

A recent study about the expressive power of Graph Neural Networks in a spectral perspective [7] suggests through a theoretical and experimental study that low-pass filters are efficient on citation networks but typically lead to bad performance on other types of graph-structured data. They suggest that Graph Neural Networks with band-pass frequency responses therefore behave better in these situations. Therefore, we suggest using band-pass scaling functions in addition to the low-pass scaling function. The initial paper which defined graph wavelets in the way they are used in this work [36] suggested a set of scaling functions designed so that they are invertible according to an admissibility condition and so that they are, like classical wavelets, localized both in spectral and spatial domain and with a density that depends on the frequency.

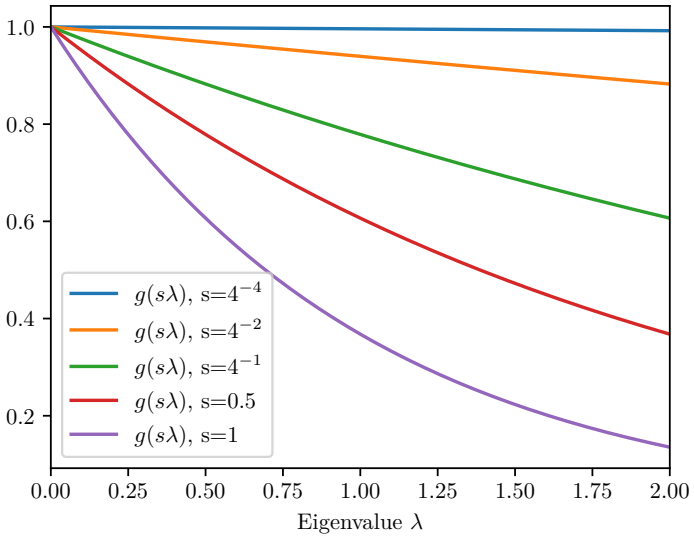


Figure 4.4: Values of scaling functions $g_s^{-1}(\lambda) = e^{-\lambda s}$ for scales $s = \{4^{-4}, 4^{-2}, 0.5, 1\}$ on eigenvalues λ ranging from 0 to 2. The scaling functions are defined in the Fourier domain and act therefore as low-pass filters on signals.

The design of the scaling functions g is motivated by the desire to achieve good localization while having a limited number of scales. They show it can be ensured if the functions behave as a monic power of the frequency near the origin. They suggest a power law decay for large frequencies so that the functions act like band-width filters. Finally, the values in between are set to be a cubic spline such that the functions and their derivatives are continuous. By finally normalizing the function since it can then be adapted by scale choices [36], the scaling function is:

$$g\left(x;\alpha,\beta,x_1,x_2\right) = \begin{cases} x_1^{-\alpha}x^{\alpha} & \text{for } x < x_1 \\ q(x) & \text{for } x_1 \leq x \leq x_2 \\ x_2^{\beta}x^{-\beta} & \text{for } x > x_2 \end{cases} \tag{4.21}$$

where the values of the cubic spline $q(x)$ are determined by $q(x_1) = q(x_2) = 1$, $q'(x_1) = \frac{\alpha}{x_1}$ and $q'(x_2) = \frac{-\beta}{x_2}$. They suggest to take logarithmically equispaced scales adapted to the upper eigenvalue λ_{max}: $s_{min} = \frac{x_2}{\lambda_{high}}$, $s_{max} = \frac{x_2}{\lambda_{small}}$ where $\lambda_{high} = \lambda_{max}$ and λ_{small} is proportional to the maximal eigenvalue: $\lambda_{small} = \frac{\lambda_{max}}{K}$. The other parameters are set to $K = 20$, $\alpha = \beta = 2$, $x_1 = 1$ and $x_2 = 2$.

In order to stably represent the low frequency content defined on the vertices of the graph, they introduce a second class of wavelets which acts like a low-pass filter. Its placement is determined by the low frequency parameter λ_{small} and is defined by:

$$h(\lambda) = \gamma \exp\left(-\left(\frac{\lambda}{0.6\lambda_{small}}\right)^4\right) \tag{4.22}$$

where γ is set such that $h(0)$ has the same value as the maximum of the scaling functions g. The full set of scaling functions is then composed of the low-pass filter h and the band-pass functions g. This set of functions for parameters $\alpha = \beta = 2$ is

illustrated on Figure 4.5. We see from definition of the scaling functions $g_s(\lambda)$ and the illustration that the sharpness of the band-pass filters increases when taking higher values for α and β.

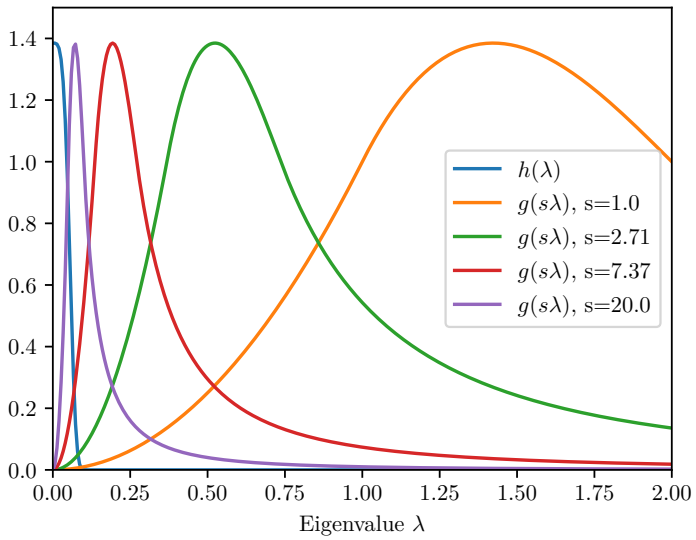


Figure 4.5: Values of scaling functions $g_s^(\lambda)$ and $h(\lambda)$ defined at Equation 4.21 and 4.22 respectively. The scaling functions are defined in the Fourier domain. One can see that functions g act like a smoothed band-pass filter, centered at different frequencies depending on the scale.

This family of scaling functions has been successfully applied to multiscale community mining [83]. This result suggests that these filters are able to detect communities at different scales in a graph, which could be a desired property to implement adaptive information propagation on the graph.

Unfortunately, the inverse transform of the individual filters is unstable due to the large amount of very small values outside the filtering region of each filter. Therefore, we suggest using shifted filters for better problem conditioning. Let $\nu > 0$ be a small shift, the shifted scaling functions are then defined by $g_{s,\nu}(\lambda) = \nu + g_s(\lambda)$ and $h_\nu(\lambda) = \nu + h(\lambda)$. We suggest using a value of $\nu = 0.05$, so that the filters are still highly discriminant.

In this work, we use three band-pass scaling functions and define the range of scales such that the smallest nonzero eigenvalue which corresponds to the smallest graph frequency is not discriminated by the low-pass filter. Indeed, the first nonzero eigenvalue and its respective eigenvector capture important information about the cheapest graph cut[3] and therefore about the graph main communities [77, 83]. We therefore take $\lambda_{small} = 5\lambda_2$ and set $\lambda_{high} = \rho\lambda_{max}$ where $\rho \in (0, 1]$ is a model hyperparameter.

Note that both the Laplacian and normalized Laplacian can be used as basis for the definition of the wavelet transforms matrices since both share similar properties

[3]The cheapest graph cut is defined here as the lowest number of edges that must be removed from the graph so that it is not connected anymore.

and are suitable to define a meaningful the wavelet transform [36]. We therefore take the normalized Laplacian matrix since its eigenvalues are within the range $\lambda \in [0, 2]$, see [15] for the proof. This fixed range allows to more easily tune the hypermarameter ρ.

We also investigate another choice of scaling functions in which the band-pass filters are defined in a way they decay less inside the frequency band. This choice is motivated by the fact the scaling functions proposed by [36] have the drawback of discriminating some frequencies whatever the scale, information that would reside in these frequencies could therefore be lost. Indeed, one can see on Figure 4.5 that for example frequencies $\lambda \approx 0.3$ or $\lambda \approx 0.9$ both lie between the peaks of two scaling functions, so that all the scaling functions discriminate these frequencies.

Butterworth band-pass filters [11] seem to be a reasonable choice. Indeed, these filters have a maximally flat frequency response within the low and high cut frequencies and no oscillations outside of the interval [85]. The same scales choices than before are taken: $\lambda_{small} = 5\lambda_2$ and $\lambda_{high} = \rho\lambda_{max}$. An additional low-pass Butterworth filter with a cut frequency at the first scale is added to take into account the low frequencies. This set of functions for a butterworth filter of order five is illustrated on Figure 4.6.

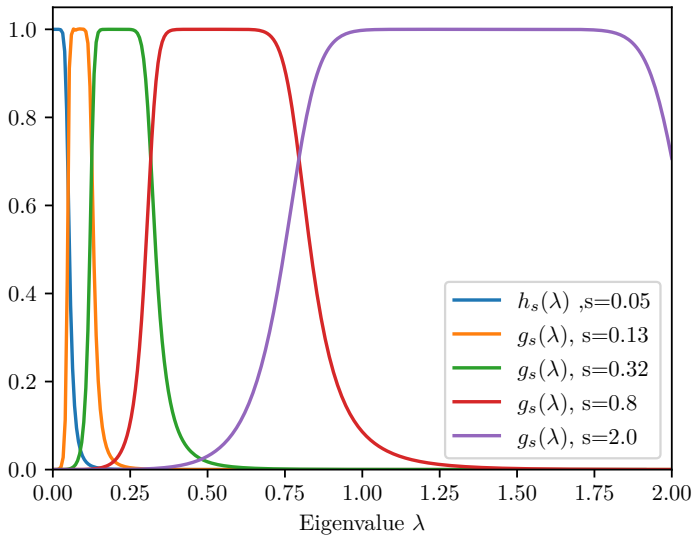


Figure 4.6: Values of scaling functions $g_s(\lambda)$ and $h_s(\lambda)$ which are respectively Butterworth bandpass and lowpass filters of degree five. The scaling functions are defined in the Fourier domain. The low cut and high cut frequencies are given by the successive scales.

Such a family of wavelet functions seems well suited for the task of semi-supervised graph node classification since it combines both low and higher frequencies information in a trainable way. Indeed, their influence can be controlled by the kernel weights $W_{l,s}^c$ and by the combination weights α_s. Results for these different models will be presented in Section 5.5.

4.5 Autologistic SVM (AutoSVM)

The Support Vector Machine presented in Section 2.3 is a supervised classifier, which means it is trained by only taking the features of the labelled nodes and their respective label into account. In this section, we will present the Autologistic SVM [6, 9, 32], an adapted SVM that takes the graph structure into account through the use of a quantity called the autocovariate. Like the previous models, the AutoSVM relies on the assumption that neighbouring nodes have a higher chance of belonging to the same class (autocorrelation). Therefore, the autocovariate is defined as a weighted average of the predicted class of the neighbouring nodes [51, 9]:

$$z_i^{(c)} = \sum_{j=1}^{N} P_{ij} \hat{y}_j^{(c)} \tag{4.23}$$

where P_{ij} is the degree-normalized transition matrix and $\hat{y}_j^{(c)}$ is the predicted class membership estimate for node j in class c.

The model is built as follows [27, 51, 9]: it first predicts the class of the unlabelled nodes using a simple SVM trained using the features X only. This gives a first estimate for $\hat{y}_j^{(c)}$ (the values for the labelled nodes are kept unchanged). Then, the model iteratively trains a new SVM which takes into account both the features X and the autocovariates Z that have been computed using the probabilities estimates. This procedure is repeated until convergence of the predicted membership values.

A variant of this method could be to use the weighted average of the neighbouring features into account instead of, or in addition to the class probability estimates. This approach would be then be close to the one of DCNN since it would build a set of features that takes the features of the neighbouring nodes into account.

4.6 Graph based regularization

Semi-supervised classification on graphs involves learning the labels knowing only a small portion of the labels. One could aim to find a regularization term for the loss function that would help the model improve its weights in the right direction by providing some kind of prior knowledge about the unlabelled nodes. Recall the main hypothesis on graph structured data, which states that neighbouring nodes are more likely to share the same label. Using this hypothesis, we define a smoothness penalty on the predicted labels [27], so that the loss favors predictions for which the predicted class vector $\hat{Y}$ of a node is closer to the ones of its neighbouring nodes. The smoothness penalty is added to the loss function with a smoothing parameter $\gamma_{\mathcal{G}} > 0$:

$$\mathcal{L}(Y, \hat{Y}; M) = \mathcal{L}_{base}(Y, \hat{Y}) + \gamma_{\mathcal{G}} \mathcal{L}_M(\hat{Y})$$

where M is a matrix representation of the graph and $\mathcal{L}_M(\hat{Y})$ is the smoothness penalty, which depends on the choice of M. There remains to find a matrix M that best fits to the goal of penalyzing abrupt class changes.

4.6.1 Laplacian matrix

Let us take as first choice the graph Laplacian matrix. Indeed, recall from Section 4.1 that the quantity Lx measures local variations of signal x between graph edges:

$$[Lx]_i = \sum_{j \in \mathcal{V}} A_{ij} \left(x_i - x_j\right)$$

And that the quantity $x^T Lx$ is a least square error cost function defined on the graph $\mathcal{G}$ [27]:

$$x^T Lx = \frac{1}{2} \sum_{\{i,j\} \in \mathcal{E}} A_{ij} (x_i - x_j)^2$$

If we take the predicted class vectors as signal on the graph: $x = \hat{Y}$, we get a regularization which penalizes abrupt class changes. The Laplacian regularization is defined by [27]:

$$\mathcal{L}_{L,p}(\hat{Y}) = \|L\hat{Y}\|_p^p \tag{4.24}$$

Which is for the L1-norm:

$$\begin{aligned} \|L\hat{Y}\|_1^1 &= \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} A_{ij} \left(x_i - x_j\right) \\ &= \sum_{i \in \mathcal{V}} \left[Lx\right]_i \end{aligned}$$

4.6.2 Probability transition matrix

Consider now taking the following smoothing matrix: $M = I - P$. We show this matrix is a degree normalized Laplacian matrix:

$$D^{-1}L = D^{-1}(D - A) = I - D^{-1}A = I - P$$

The smoothing penalization $\mathcal{L}_{I-P,p}(\hat{Y}) = \|(I - P)\hat{Y}\|_p^p$ will therefore have a similar effect to the Laplacian regularization [27], but with penalties inversely proportional to the node degree. This is suggested to make the regularization less sensitive to class imbalance [12].

Chapter 5

Experiments

The papers presenting the different models [47, 5, 87] all use a similar methodology to evaluate the models: they are typically compared with baseline models on three citation network datasets, namely Cora, Citeseer and Pubmed. Furthermore, these Graph Neural Networks are always compared using the same folds and with models already fine-tuned, without justifying all the hyperparameters choices: does the features pre-processing increase the accuracy ? does the regularization used help to avoid overfitting ? what is the influence of the learning rate ? and so on. It is therefore difficult to asses the true performance of the models.

In this work, we overcome these issues by first comparing the models on a more diverse set of datasets and by running each experiment 10 times with 5-fold cross-validation. We only fit the main hyperparameters and set all the other parameters to the same fixed values for all models. By doing so, we aim to provide a rigorous comparison of all the models and to separate the influence of the models' structures with the influence of the models' improvements.
Then, we investigate how to improve the models separately by conducting several experiments: analyse the influence of model based regularization, graph based regularization and better choices of loss functions. From there, the significative models improvements are kept and the fine-tuned models are compared.

This set of experiments allows to get a clear overview of what are the relative performances of each model depending on the nature of the dataset and to learn from there a deeper understanding of the qualities needed for a convolutional model to perform well, whatever the type of graph-structured data it learns from. By doing so, we show that low-pass filtering is not always sufficient and present a model that predicts the class through the use of convolutions performed at different well chosen band frequencies, namely the Multiscale Wavelet Convolutional Neural Network.

5.1 Datasets

The datasets are simple connected graphs. These are composed of an adjacency matrix $A \in \mathbb{R}^{N_\mathcal{V} \times N_\mathcal{V}}$ and a feature matrix $X \in \mathbb{R}^{N_\mathcal{V} \times N_F}$ which associates N_F features to each node of the graph. The datasets are the following:

- Cornell, Texas, Washington and Wisconsin datasets [18, 17]. Each of the four datasets consists of web pages from the computer science departments of the respective university. The nodes of the graph are the web pages and two nodes are linked when there is a hyperlink between the two web pages. Each node is labelled into one of the categories: course, faculty, student or project. Each web page is also characterized by a binary word vector indicating which words are present on the web page.

- Ego Facebook datasets [53]. Each of the three datasets consists of friend communities from Facebook. The datasets have each two communities, the classification task is thus to find the community of each user. In addition, each node is characterized by a set of features which are information about the Facebook user.

- CiteSeer dataset [70]. The dataset consists of scientific publications divided into six classes. Each node of the graph is a publication and two publications are linked when one cites the other. Such a graph is often referred as co-citation graph. In addition to the graph, each publication is also characterized by a binary word vector indicating which words are present on the article.

- Cora dataset [70]. Like CiteSeer, Cora is a co-citation graph with binary word vectors indicating which words are present on the article. The publications are divided into seven classes.

- Wikipedia dataset [75]. The dataset consists of Wikipedia articles that appeared in the featured list between the 7th and 21st October of 2009. An edge between two articles represents a hyperlink. The node features are the words present in the article and are encoded as a tf/idf-weighted vector. Note that this dataset is the only one for which the features are non-binary.

- Amazon Photos and Amazon Computers. These datasets are segments of the Amazon co-purchase graphs [76, 58]. Nodes represent goods and their class is given by the product category. There are eight categories for Amazon Photos and ten for Amazon Computers. Products are linked when frequently bought together. The features are binary word vectors of the product reviews.

Details about the datasets can be found in Table 5.1.

For seek of more intuition about the underlying structure of the graphs, we present four of the datasets graphs on Figure 5.1. The visualizations for all datasets at a larger scale can be found in Appendix C.

Dataset	Nodes	Edges	Classes	Nonzero Features	Moran's $\bar{I}$
Cornell	183	277	4	18.43 %	0.0486
Texas	183	279	4	17.44 %	-0.0283
Washington	215	365	4	17.45 %	-0.0257
Wisconsin	251	450	4	18.90 %	0.0521
Facebook 1	756	30,780	2	7.24 %	0.9549
Facebook 2	793	14,816	2	4.69 %	0.8142
Facebook 3	1,045	27,791	2	4.89 %	0.4929
CiteSeer	1,392	2,302	6	4.66 %	0.1070
Cora	2,485	5,069	7	4.12 %	0.1168
Wikipedia	3,254	33,218	14	21.44 %	0.0082
Amazon 1	7,487	119,043	8	33.05 %	0.0750
Amazon 2	13,381	245,778	10	33.17 %	0.0630

Table 5.1: Datasets statistics.

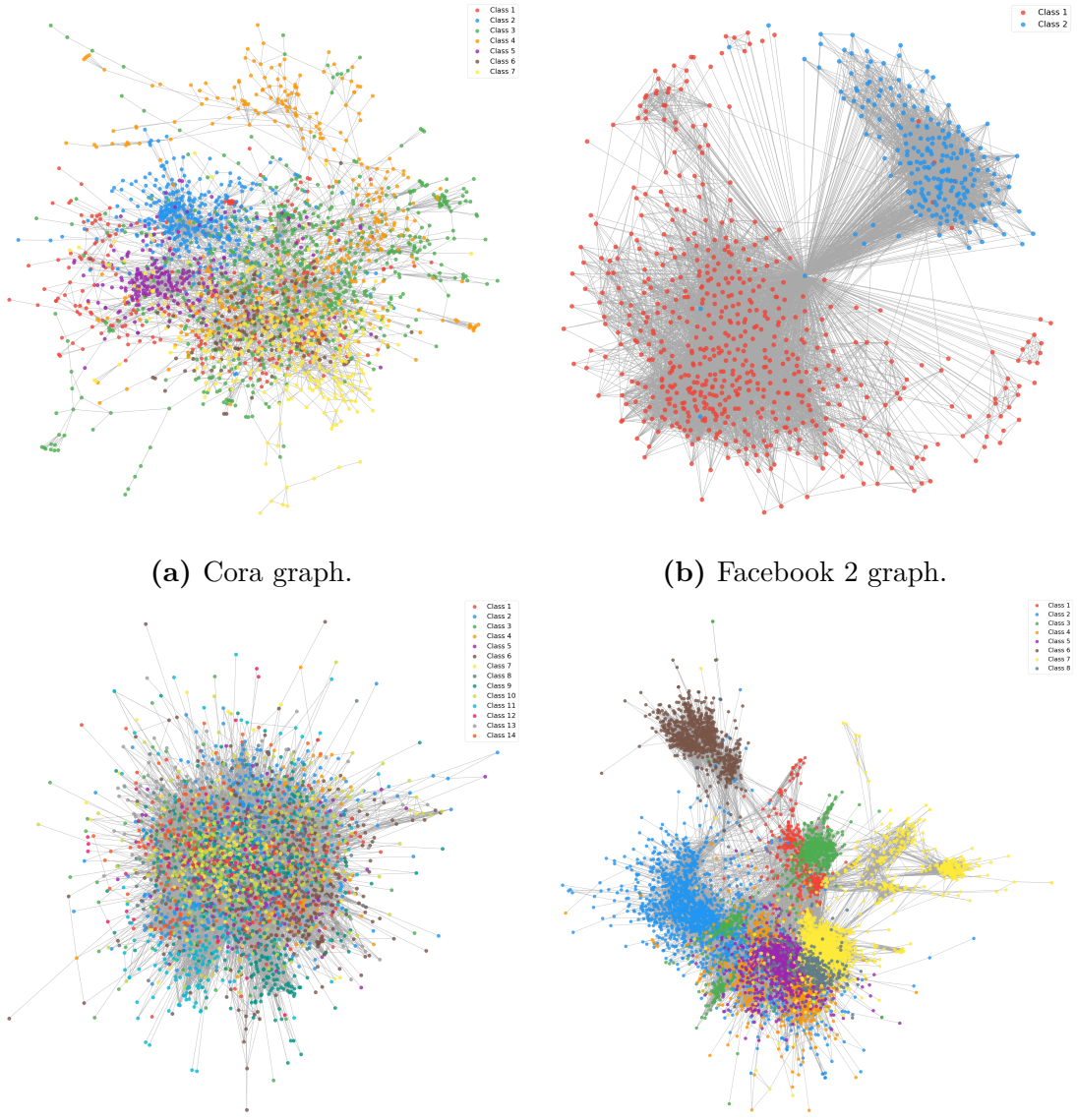
5.2 Experimental Setup

Pre-processing The following pre-processing steps are applied to each of the graphs:

- The graph is symmetrized by applying the following transformation to the adjacency matrix: $\frac{A+A^T}{2}$. This gives an undirected graph with weights two times stronger for nodes that were mutually pointing to each other.
- Self-loops are removed by setting the diagonal of the adjacency matrix to zero.

A feature selection is also applied on each dataset. Only the one-hundred most relevant features for classification are kept, according to a Chi-square statistical test [16]. The test is performed as follows: (1) Take each vector $X_{:,i}$ corresponding to the value of the i-th feature at each node. (2) Apply a chi-square test to each pair $(X_{:,i}, Y)$ where $Y \in \{0,1\}^{N_\mathcal{V} \times N_C}$ is the binary class matrix. The null-hyopthesis states that the variables are uncorrelated. (3) Select the one-hundred features with the lowest p-value, i.e. the features that are the most correlated to the class memberships.

Train-validation-test splits We assess the accuracy of the models with 5-fold cross-validation. The node labels are divided into two equally sized train and test sets. The train set is then further divided into five sub-sets of the same size. Four are used for training and the last one is the validation set. This methodology is the one described in Section 3.3 and is illustrated on Figure 3.4. Each experiment is run ten times with 5-fold cross-validation, the train-validation-test splits are different at each of the ten runs but are the same for the different models.



(a) Cora graph.

(b) Facebook 2 graph.

(c) Wikipedia graph.

(d) Amazon Photos graph.

Figure 5.1: Graphs of the different datasets, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey edges. On this illustration, one can see the datasets have different types of structures. The full-size illustrations are available in Appendix C.

5.3 Fair Model Comparison

We compare the CNN-based models against the baseline models SVM, MLP and AutoSVM on the task of graph node classification on the twelve datasets presented in previous section. The models are summarized in Table 5.2. Each experiment is run ten times with 5-fold cross-validation, following the methodology of Section 5.2. We select the set of best hyperparameters for each model on each dataset by taking the combination of hyperparameters that most frequently gives the best accuracy

across the ten runs.

Abbreviation	Model	Features	Papers
MLP	Multi-Layer Perceptron	X	[55, 60, 40]
SVM	Support Vector Machine	X	[40, 60, 72]
AutoSVM	Autologistic Support Vector Machine	X, P	[51, 9]
GCN	Graph Convolutional Network	X, $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$	[47]
GWNN	Graph Wavelet Neural Network	X, L	[87]
DCNN	Diffusion Convolutional Neural Network	X, P	[5]

Table 5.2: Summary of the compared models.

For a fair comparison, all the Neural Networks are trained using the same pre-processing steps (see Section 5.2) and without any further preprocessing or normalization. These models are trained for a maximum of 200 epochs using a softmax cross-entropy loss with Adam optimizer [46] and with an early stopping window of size ten. No other form of regularization is applied to the models.

For each Neural Network, the combinations of following hyperparameters are tested: the initial learning rate (0.01 and 0.001) and the activation function of all layers except the output layer (ReLU, Tanh and Sigmoid). In addition, some model specific parameters are tuned: for the Neural Networks with a variable number of layers (i.e. MLP, GCN and GWNN), a fixed number of two layers is taken and different number of hidden units are considered (16, 32 or 64 hidden units). For DCNN, we tested the model with one to three hops. For GWNN, a wide range of scaling factors are considered (ranging from 4^{-4} to 1). Indeed, contrary to what is suggested in [87] about the interesting range of values for the scaling parameter, way smaller values sometimes give outstanding results. The influence of this hyperparameter choice will be further discussed in Sections 4.4 and 5.5. For SVM and AutoSVM, two different kernels are tested (RBF and sigmoid).

5.3.1 Statistical Tests and Results

The mean accuracy of the ten runs is summarized in Table 5.3. From these results, some models seem to perform better in general. In particular, GWNN is the only Neural Network that performs well both on the small datasets and on the larger ones. Nevertheless, none of the models systematically outperforms the others. Therefore, let us present statistical tests to assess if some of the models are better than the others.

Friedman test [28] is a statistical test which assesses if all models are statistically equivalent without making the assumption of normality of the distribution. The null-hypothesis states that the average ranks of the models should be equal since they are equivalent [21]. The Friedman statistic is distributed according to the chi-square distribution χ^2_F with $M-1$ degrees of freedom (when N and M are large enough) [28]:

	SVM	AutoSVM	MLP	GCN	DCNN	GWNN
Cornell	81.8 ± 1.9	83.5 ± 2.3	**83.9** ± 1.8	65.8 ± 1.5	75.5 ± 4.2	82.8 ± 2.2
Texas	88.2 ± 1.6	**90.3** ± 1.8	88.3 ± 1.9	64.3 ± 2.2	84.4 ± 2.6	88.9 ± 1.9
Washington	81.3 ± 2.4	83.1 ± 2.1	**86.0** ± 1.2	65.2 ± 2.0	78.1 ± 2.1	84.8 ± 1.1
Wisconsin	87.7 ± 2.0	**88.9** ± 2.0	88.4 ± 1.6	59.7 ± 3.8	83.0 ± 3.6	87.1 ± 2.2
Facebook 1	92.5 ± 1.0	96.5 ± 0.7	92.2 ± 0.8	96.8 ± 0.5	96.8 ± 0.5	**96.9** ± 0.7
Facebook 2	92.6 ± 0.8	97.4 ± 0.4	91.9 ± 0.6	98.3 ± 0.4	**98.4** ± 0.6	**98.4** ± 0.4
Facebook 3	81.2 ± 0.8	83.8 ± 1.3	79.6 ± 1.1	**84.1** ± 1.0	83.7 ± 1.5	83.6 ± 1.5
CiteSeer	72.0 ± 1.0	74.7 ± 1.0	71.8 ± 0.9	77.3 ± 0.6	76.4 ± 1.1	**77.8** ± 0.8
Cora	72.2 ± 1.0	81.7 ± 0.7	72.7 ± 0.8	86.1 ± 0.7	86.5 ± 0.6	**87.2** ± 0.7
Wikipedia	54.1 ± 1.0	54.1 ± 1.0	60.4 ± 1.0	48.7 ± 0.6	**65.2** ± 0.8	60.4 ± 0.6
Amazon 1	90.3 ± 0.3	93.4 ± 0.2	89.6 ± 0.3	93.6 ± 0.3	**95.1** ± 0.2	93.9 ± 0.2
Amazon 2	83.9 ± 0.2	88.7 ± 0.2	82.2 ± 0.3	89.0 ± 0.5	89.1 ± 0.3	**89.7** ± 0.2

Table 5.3: Results of 10 runs with 5-fold cross-validation. Mean accuracy and standard deviation over the 10 runs are shown in percent. For each dataset, the best accuracy is shown in bold.

$$\chi_F^2 = \frac{12N}{M(M+1)} \left[\sum_j R_j^2 - \frac{M(M+1)^2}{4} \right] \tag{5.1}$$

where R_i is the average rank of model i, N is the number of datasets and M is the number of methods.

Iman and Davenport [42] showed that this statistical test is undesirably conservative and derived a corrected Friedman test, which is distributed according to the F-distribution with $M-1$ and $(M-1)(N-1)$ degrees of freedom:

$$F_F = \frac{(N-1)\chi_F^2}{N(M-1) - \chi_F^2} \tag{5.2}$$

Using the previously found accuracies, we compute a Friedman and corrected Friedman statistics of respectively 16.62 and 4.21. The critical values for $p < 0.05$ are respectively given by 11.07 and 2.38 and correspond to p-values of 0.53% and 0.26%, which means that both tests have a probability of rejecting the null hypothesis while it is true of less than 5%.

Since the null-hypothesis is rejected, let us conduct a post-hoc test which finds if groups of models are significantly different. Nemenyi's test [65] makes pairwise comparisons between all the models. Two models are considered statistically different if the difference between their average ranks is higher than the critical difference, defined by [21]:

$$\kappa_{CD} = q_\alpha \sqrt{\frac{M(M+1)}{12N}} \tag{5.3}$$

where q_α is the Studentized range statistic with a probability of α. The critical difference at $p < 0.1$ and $p < 0.05$ is respectively of 1.977 and 2.176. The results for $p < 0.1$ are illustrated on Figure 5.2.

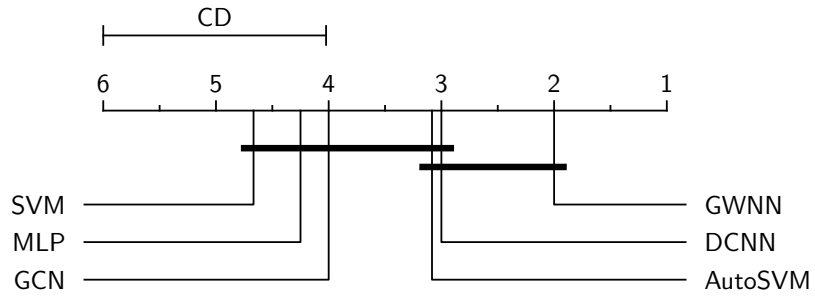


Figure 5.2: Results of Nemenyi's test on the six models for $p < 0.1$. Critical difference value of 1.98. Connected models means they are not statistically different according to Nemenyi's test.

5.3.2 Discussion of Results

From these first results, one can see that GCN and DCNN give poor results on the four university datasets and on Wikipedia dataset. We also note that the models using only the features perform systematically worse than the other models on the Facebook datasets, citation networks and Amazon datasets. These observations suggest that the university datasets are feature-driven datasets whereas the graphs of the other datasets seem to provide useful additional information [51]. This hypothesis is verified by the close to zero Moran's $\hat{I}$ statistics of the four university datasets (see Table 5.1). Still, AutoVSM outperforms SVM on all datasets except Wikipedia, which shows that all graphs seem to carry additional information. The Wikipedia dataset gives intriuging results: the best performing model is by far DCNN and then MLP and GWNN, whereas GCN performances are really poor on the dataset. One could try to explain these result by the fact that the Wikipedia graph provides additional information but that this information is difficult to correctly spread along the graph. Indeed, if we compare Wikipedia graph to the other main families of graphs (i.e. citation networks, friends networks and Amazon graphs) on Figure 5.1, one can observe that the classes are more mixed up. Still, by taking a closer look to the graph structure, one can see on Figure 5.3 that the dataset still provides useful information about the class memberships of neighbouring nodes. In that sense, Wikipedia dataset seems to be a tough but interesting dataset for model performance assesment.

Nemenyi's test suggests that GWNN, DCNN and AutoSVM are not statistically different one from another and that only GWNN is statistically different from the three worse performing models. Indeed, GWNN, DCNN and AutoSVM perform relatively well on most datasets, but GWNN is the only model that systematically ranks in the top three best accuracies, both on feature-driven datasets and on graph-driven datasets.

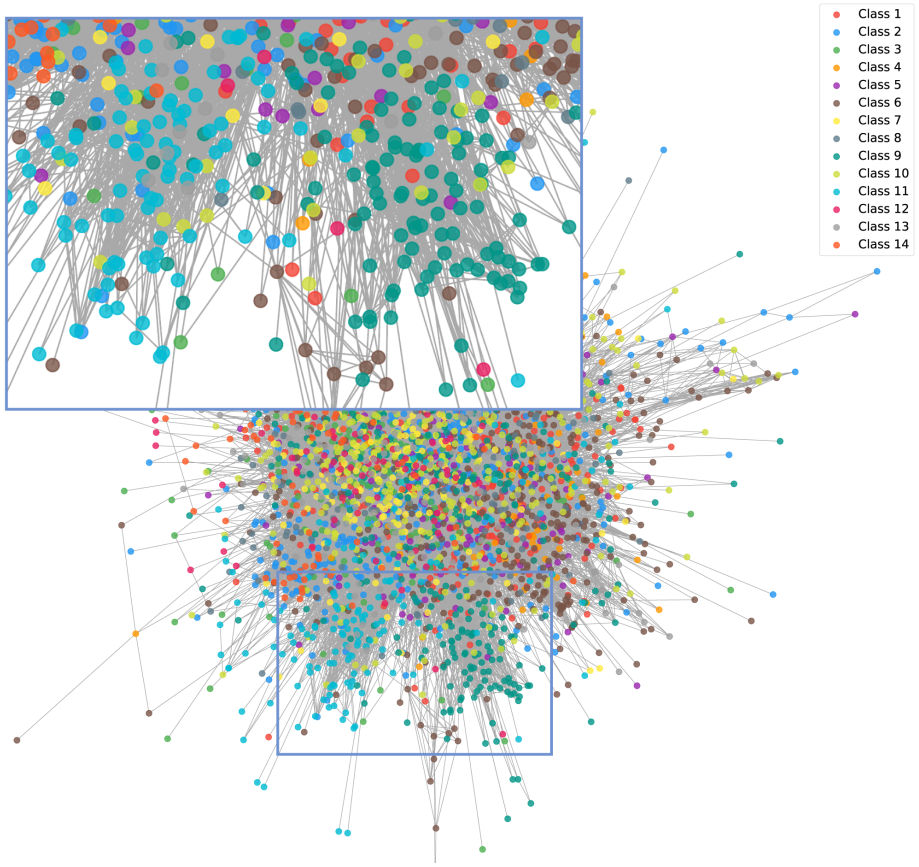


Figure 5.3: Closer look at Wikipedia graph structure. This illustration shows the graph structure seems to provide useful information about the class membership of the nodes.

5.4 Model Improvements

Many research papers [20, 47, 5, 87] do not discuss hyperparameters choices when testing a new model. In particular, the loss function choice and the regularization techniques are often fixed without any justification or experiments to assess their effect on the model accuracy. In the following, we investigate different loss functions and draw conclusions on which performs the best in general. Then, we assess the influence of model and graph regularization on the generalization capabilities of the models.

5.4.1 Is Softmax Cross-Entropy the Best Choice ?

The loss function of the previous experiments was the softmax cross-entropy. This loss is by far the most used in the literature of Graph Neural Networks [47, 5, 87], but is this a reasonable choice ? In the following, we will investigate the performance of the Neural Networks on semi-supervised graph node classification when using the different losses presented in Section 3.1.

The models are trained and evaluated using the same methodology as in the

previous section. The hyperparameters are set to the optimal values found during the initial models comparison, which can be found in Appendix A.

The Friedman and corrected Friedman critical statistics for the nine different loss functions on the twelve datasets are respectively 15.51 and 2.05. The p-values on each of the four Neural Networks are given in Table 5.4 and the full accuracy results are available in Appendix A.

	MLP	GCN	GWNN	DCNN
Friedman	$1.1 \cdot 10^{-6}$	$1.8 \cdot 10^{-3}$	$4.6 \cdot 10^{-6}$	$6.6 \cdot 10^{-3}$
Corrected Friedman	$1.0 \cdot 10^{-8}$	$7.1 \cdot 10^{-4}$	$1.2 \cdot 10^{-7}$	$3.7 \cdot 10^{-3}$

Table 5.4: The Friedman and corrected Friedman p-values for the different loss functions of Section 3.1 on the twelve datasets. The results suggest the null-hypothesis can be rejected for all models.

The values for both statistical tests reject the null-hypothesis for the different models, which means we can reject the hypothesis which states that the losses are all equally efficient for the learning process.

In these experiments about the loss functions, the improved losses are compared with a control loss, namely the softmax cross-entropy. In such a situation, we use the Bonferroni-Dunn test instead of the Nemenyi test. The test is generally more conservative than Nemenyi's test, but is more powerful in the specific case of pairwise comparisons [21]. The test statistics for comparing the i-th and j-th classifier using this method is given by:

$$z = (R_i - R_j) / \sqrt{\frac{M(M+1)}{6N}}$$

where R_i is the average rank of model i, N is the number of datasets and $M - 1$ the number of compared improvements. The z value is used to find the corresponding probability from the normal distribution, which is then compared with value α divided by the number of comparisons made [21]. An alternative way to compute the same test is to calculate the critical difference using the same equation as for the Nemenyi test, but using the critical values for $\frac{\alpha}{k-1}$.

The softmax cross-entropy loss systematically outperforms the other loss functions on MLP, GCN and GWNN. Also, despite its interesting properties, L1 losses are considered statistically worse than the baseline loss for all models except DCNN. This is illustrated for MLP on Figure 5.4. The Bonferroni-Dunn intervals for the other models can be found in Appendix B. This could be explained by the fact that the derivative of the L1 loss is not continuous, which therefore makes the learning process more difficult.

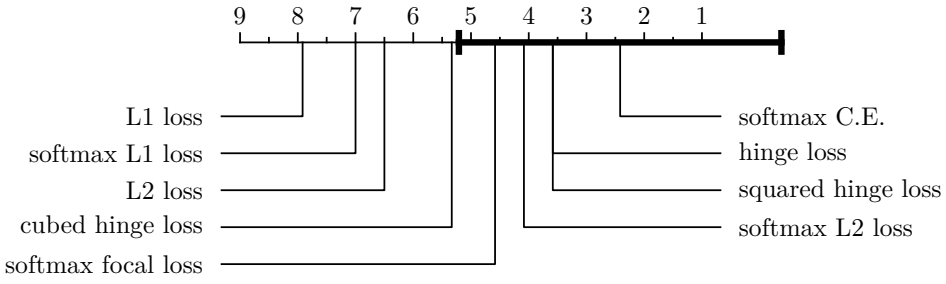


Figure 5.4: Results of Bonferroni-Dunn's test on MLP with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. MLP with Softmax Cross-Entropy loss (C.E.)).

On Figure 5.5, one can see that the squared and cubed hinge losses perform in general better than the cross-entropy for DCNN and that the hinge loss has a worse average rank. We assume that it is due to the non-differentiability of the hinge loss around $y = 1$ and due to its linear behavior elsewhere. The cubed hinge loss increases the accuracy by 2% to 5.5% on the university datasets and gives similar results to the cross-entropy loss on the other datasets. We therefore consider its improvements as interesting despite Bonferroni-Dunn's test. We further justify this decision by the fact this statistical test does not take the improvement magnitudes into account and because the statistical test is conservative [21].

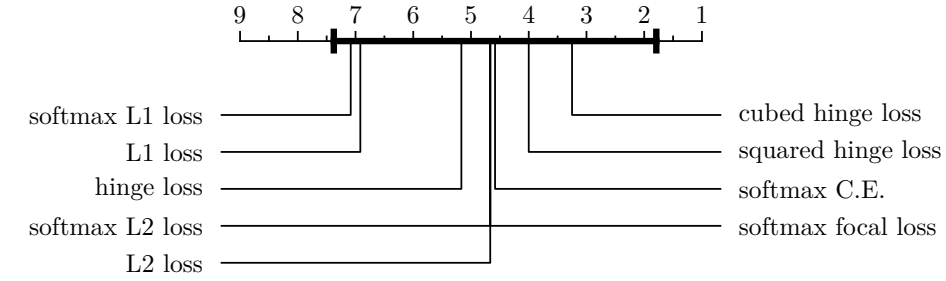


Figure 5.5: Results of Bonferroni-Dunn's test on DCNN with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. DCNN with Softmax Cross-Entropy loss (C.E.)).

5.4.2 Improving Generalization Through Regularization

Since the emergence of Machine Learning and in particular Deep Neural Networks, regularization has become an important technique to improve the ability of Neural Networks to generalize to unknown data points [33, 39, 80, 68]. In the following, dropout and weights regularization (with both L1 and L2 norms) will be investigated. By doing so, we asses the effect of these regularization techniques on semi-supervised classification of graph-structured data. This set of experiments is motivated by the fact that performance of most Graph Neurals Networks [47, 87, 5] are evaluated

using regularization but usually without assessing the effect of such techniques. We also investigate the effect of graph based regularization on the different Graph Neural Networks and on the MLP. Recall these techniques penalize non-smooth neighbouring class predictions, which is aimed to provide some kind of prior knowledge about the unlabelled nodes because of autocorrelation.

The models are trained and evaluated using the same methodology as in Section 5.3. Dropout is applied only during the training phase. The hyperparameters are set to the optimal values found during the initial models comparison, which can be found in Appendix A. The combinations of values that are investigated are given in Table 5.5.

Experiment	**Parameters**
Dropout	$p = \{0.2, 0.3, 0.4\}$
Weight regularization	$\gamma_{\mathcal{W}} = \{0.001, 0.01\}$ Norm $= \{1, 2\}$
Laplacian regularization	$\gamma_{\mathcal{G}} = \{0.001, 0.01\}$ Norm $= \{1, 2\}$
Transition probability regularization	$\gamma_{\mathcal{G}} = \{0.001, 0.01\}$ Norm $= \{1, 2\}$

Table 5.5: Range of investigated hyperparameters values for the regularization experiments.

Model improvements are again evaluated using Bonferroni-Dunn statistic test on pairwise comparisons with the baseline models. The experiments show both dropout and weight regularization do not significantly improve the test accuracy for the investigated models. The test accuracy confirms these results, since the variations are not greater than one and a half percent on the small datasets and one percent on the larger ones, which are therefore smaller variations than the standard deviation of the test accuracy. Full results including the test accuracy of each model improvement on each dataset and the related Bonferroni-Dunn test can be found in Appendix B.

Smoothing the predicted labels gives a better rank on all models with the regularization based on the row-normalized Laplacian matrix. This regularization always outperforms the regularization of the Laplacian matrix. For seek of an explanation, this could be due to the high variability in the average node degree of the dataset's graphs, which therefore makes the row-normalized Laplacian a smoothing that is proportional for all nodes whereas the Laplacian penalizes more nodes with a high degree. The most significant rank improvements occur on GWNN and DCNN, but the improvements are statistically significant only on DCNN according to Bonferroni-Dunn's test. This is illustrated on Figure 5.6. Finally, note that GCN is the only Graph Neural Network for which none of the regularization techniques seems to have an effect. This could be the consequence of the already included regularization and approximation in the model structure. Recall indeed

that GCN limits the information propagation of one layer to one hop neighbours and that the model further addresses overfitting by reducing the number of weight parameters[1].

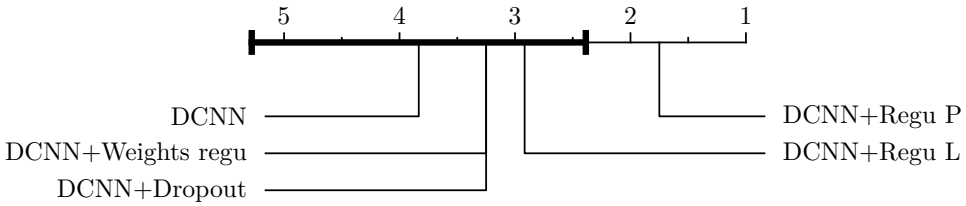


Figure 5.6: Results of Bonferroni-Dunn's test on DCNN with the different model improvements for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. DCNN without any improvements).

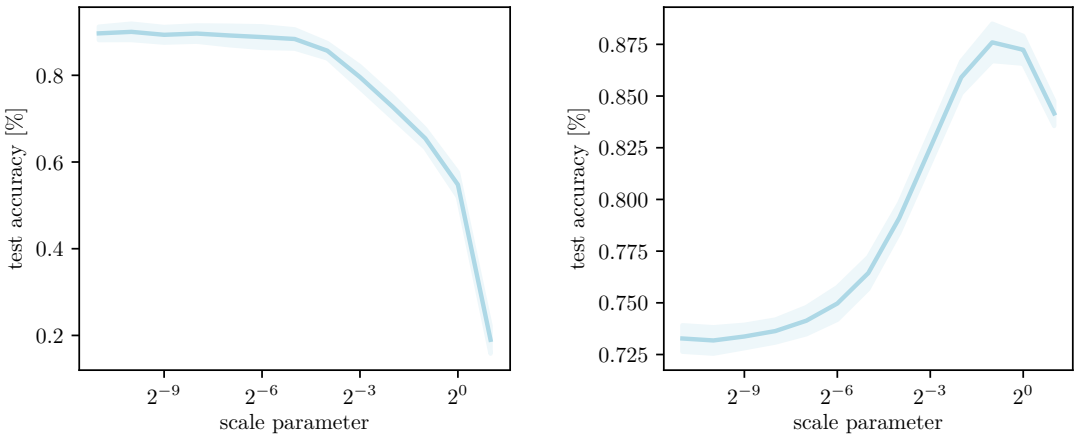
5.5 Multiscale Graph Wavelet Neural Network

The development of the Multiscale Graph Wavelet Neural Network is motivated by the conclusions drawn in [7] which state that low-pass filtering is not sufficient for classification of graph-structured data on graphs different from citation networks. We verify this idea and show the suggested scales in [87] are too narrow by drawing the mean test accuracy of GWNN as a function of the scale parameter. Results on Figure 5.7 for Texas and Cora dataset go in this direction. Indeed, scales close to one give very good results on Cora, which is a citation network. When the scales approach zero, the model accuracy falls to values close to the results of a MLP, which is explained by the fact that very low scales are close to no filtering. Results on Texas dataset are the best for small scales, which can be explained by the fact the dataset is features-driven (since its Moran $\hat{I}$ is close to zero).

From these observations, let us investigate a multiscale approach. In the first experiment, we will use the exponentially decreasing scaling functions at multiple scales: $S = \{0, 4^{-4}, 4^{-2}, 0\}$ using the multi-scale design presented in Section 4.4.1. Then, we reiterate the experiments with the two different band-pass filtering solutions that were presented in Section 4.4.2. Mean and standard deviation of the test accuracy for the first and third experiments are presented in Table 5.6.

The multi-scale approach with low-pass filters (M-GWNN Exp) shows very similar mean test accuracy as the initial GWNN with a well chosen scale (i.e. tuned using cross-validation). Nevertheless, the accuracy highly increases on the Wikipedia dataset. We suggested in Section 5.3 that Wikipedia is a challenging dataset in the sense it is mainly features-driven but still provides additional information in the graph, like illustrated on Figure 5.3. In that sense, the combination of a band-pass filter with less discriminant filters through the use of multiple scales seems to provide additional information to the model and therefore perform better on this challenging

[1]Indeed, the convolution operation is approximated by the simplification of the convolution parameters to: $\theta := \theta'_0 = -\theta'_1$.



(a) Texas dataset. **(b)** Cora dataset.

Figure 5.7: Influence of the scaling parameter of GWNN on the test accuracy for Texas and Cora datasets. The blue line represents the mean test accuracy and the light blue border is the standard deviation over ten runs. The maximal accuracy is at $s = 2^{-10}$ for the Texas danaset and at $s = 0.5$ for the Cora dataset.

	GWNN	**M-GWNN Exp**	**M-GWNN Butterworth**
Cornell	82.8 ± 2.2	82.5 ± 2.3	78.5 ± 1.9
Texas	88.9 ± 1.9	88.5 ± 1.9	84.4 ± 1.9
Washington	84.8 ± 1.1	84.3 ± 1.6	76.4 ± 2.7
Wisconsin	87.1 ± 2.2	87.2 ± 2.2	80.4 ± 3.1
Facebook	96.9 ± 0.7	96.5 ± 0.6	91.2 ± 2.1
Facebook	98.4 ± 0.4	97.9 ± 0.4	94.9 ± 1.6
Facebook	83.6 ± 1.5	82.9 ± 0.8	72.3 ± 2.7
Citeseer	77.8 ± 0.8	77.3 ± 0.5	74.7 ± 0.8
Cora	87.2 ± 0.7	87.7 ± 0.5	81.8 ± 0.8
Wikipedia	60.4 ± 0.6	**65.2** ± 0.6	56.1 ± 0.7
Amazon 1	93.9 ± 0.2	**94.5** ± 0.2	69.4 ± 3.7
Amazon 2	89.7 ± 0.2	**90.0** ± 0.2	67.8 ± 2.9

Table 5.6: Results of 10 runs with 5-fold cross-validation. Mean accuracy and standard deviation over the 10 runs are shown in percent. Improvements larger than the standard deviation are shown in bold. There is a notable improvement on Wikipedia dataset.

dataset. Also note that the accuracy increased by about 0.6% and 0.3% on Amazon 1 and Amazon 2, which are improvements higher than the standard deviation and suggests again some more information has been learned by the model.

The Multiscale Graph Wavelet Neural Network with Butterworth low-pass and band-pass filters gives decent results but below GWNN baselines. These decreased results are mostly explained by the model instability, as we can see in the high standard deviation the model suffers from. We therefore suggest that a better tuning of the hyperparameters could lead to better performance. In particular, the choice of the shift parameter ν and of the Butterworth filter order seem to be crucial in the model stability.

5.6 Fine Tuning and Discussion

In this section, we will compare the fine tuned models together and compare them with the Multiscale Graph Wavelet Neural Network. The results of Section 5.4 suggest that:

1. Other losses are statistically worse or equivalent to the softmax cross-entropy on all models except DCNN, on which a cubed hinge loss improves the accuracy (especially on the small datasets).

2. A graph-based regularization with the degree-normalized Laplacian matrix seems to improve the generalization capabilities of DCNN and GWNN.

3. None of the model regularizations show significative effects on the accuracy, which suggests these methods do not improve generalization capabilities of convolutional graph Neural Networks.

From these conclusions, we train the DCNN model again with a softmax cross-entropy loss and with a cubed hinge loss and apply in both cases a degree-normalized Laplacian regularization. The fine-tuned DCNN model hyper-parameters can be found in Table A.7. The same regularization is also used for GWNN and no improvement is applied to the other models since none of the experiments showed statistically significant improvements. The mean accuracy of the ten runs on the fine-tuned models and over M-GWNN (with no improvements) is summarized in Table 5.7. SVM is not shown since it is systematically outperformed by AutoSVM.

	AutoSVM	**MLP**	**GCN**	**DCNN**	**GWNN**	**M-GWNN**
Cornell	83.5 ± 2.3	**83.9** ± 1.8	65.8 ± 1.5	79.2 ± 3.7	82.8 ± 2.2	82.5 ± 2.3
Texas	**90.3** ± 1.8	88.3 ± 1.9	64.3 ± 2.2	87.5 ± 2.6	88.9 ± 1.9	88.5 ± 1.9
Washington	83.1 ± 2.1	**86.0** ± 1.2	65.2 ± 2.0	80.0 ± 1.7	84.8 ± 1.1	84.3 ± 1.6
Wisconsin	**88.9** ± 2.0	88.4 ± 1.6	59.7 ± 3.8	85.3 ± 1.5	87.1 ± 2.2	87.2 ± 2.2
Facebook 1	96.5 ± 0.7	92.2 ± 0.8	96.8 ± 0.5	**96.9** ± 0.5	**96.9** ± 0.7	96.5 ± 0.6
Facebook 2	97.4 ± 0.4	91.9 ± 0.6	98.3 ± 0.4	98.3 ± 0.4	**98.4** ± 0.4	97.9 ± 0.4
Facebook 3	83.8 ± 1.3	79.6 ± 1.1	**84.1** ± 1.0	83.8 ± 1.2	83.6 ± 1.5	82.9 ± 0.8
CiteSeer	74.7 ± 1.0	71.8 ± 0.9	77.3 ± 0.6	76.6 ± 1.0	**77.8** ± 0.8	77.3 ± 0.5
Cora	81.7 ± 0.7	72.7 ± 0.8	86.1 ± 0.7	86.3 ± 0.6	87.2 ± 0.7	**87.7** ± 0.5
Wikipedia	54.1 ± 1.0	60.4 ± 1.0	48.7 ± 0.6	**65.9** ± 0.7	60.4 ± 0.6	65.2 ± 0.6
Amazon 1	93.4 ± 0.2	89.6 ± 0.3	93.6 ± 0.3	**95.0** ± 0.2	93.9 ± 0.2	94.5 ± 0.2
Amazon 2	88.7 ± 0.2	82.2 ± 0.3	89.0 ± 0.5	89.2 ± 0.3	89.7 ± 0.2	**90.0** ± 0.2

Table 5.7: Results of 10 runs with 5-fold cross-validation on the fine-tuned models. Mean accuracy and standard deviation over the 10 runs are shown in percent. For each dataset, the best accuracy is shown in bold.

Nemenyi's test for $p < 0.1$ gives a critical difference of 1.77. Results of Nemenyi's test are shown on Figure 5.8.

The results show that after the models improvements, GWNN still has a better average rank than DCNN. Also, the average rank of M-GWNN is worse than the one of GWNN. This is mainly explained by a large rank differences on the Facebook datasets, where the models are all very close and a slightly worse accuracy therefore

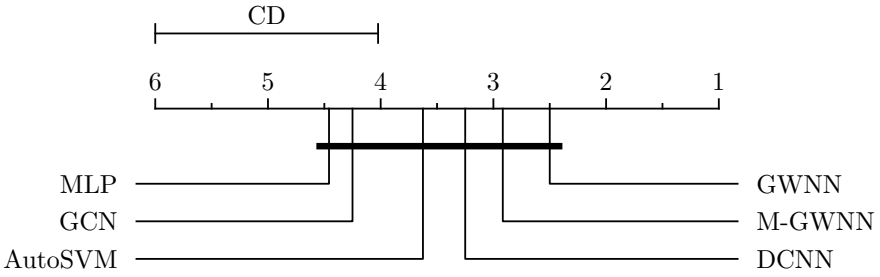


Figure 5.8: Results of Nemenyi's test on the fine-tuned models for $p < 0.1$. Critical difference value of 1.77. Connected models means they are not statistically different according to Nemenyi's test.

induces a huge penalty in the rank. Nevertheless, we observe that M-GWNN outperforms GWNN on the four largest datasets, whereas its accuracy is on average lower on smaller datasets. A possible explanation could be that M-GWNN is overfitting on small datasets, due to its way larger number of trainable weights. Therefore, we suggest that regularization could improve its results.

Which convolutional model performs the best? The experiments suggest that the Graph Wavelet Neural Network with a graph regularization using the degree-normalized Laplacian matrix is the best choice for semi-supervised classification on graph-structured data. Indeed, the model successfully predicts the class membership of the graph nodes, whatever the size and the structure of the graph, i.e. even on datasets on which the autocorrelation hypothesis is not met. Furthermore, it outperforms both DCNN and GCN on all the datasets except Wikipedia and Amazon 1. It also outperforms AutoSVM and MLP by a significant margin on the graph-driven datasets (i.e. datasets with a high Moran's $\hat{I}$).

The model still has limitations: it does significantly worse than DCNN on the challenging Wikipedia dataset. The multiscale approach of M-GWNN improves the accuracy on the Wikipedia dataset and on other large datasets, but comes with a higher risk of overfitting and a longer training time.

5.7 Limitations and Further Work

In the following, we describe several limitations of our experiments and of the multiscale approach. We also outline some future research directions.

Scalability of convolutional models Through the complexity analysis of the convolutional models, we have shown that the sparse implementations of GCN, GWNN and DCNN share similar time complexity bounds (see Table 5.8). However, the hidden constants are different. Therefore, DCNN and GWNN are slower and these models might therefore present limitations on very large datasets. Furthermore, GWNN involves an expensive precomputation step (eigendecomposition of the Laplacian matrix) and the sparsity (i.e. the number of nonzero elements) of the

wavelet transform might be high for some graphs, which then further increases the training time. An approximation of the graph wavelets using a truncated expansion in terms of Chebyshev polynomials highly decreases the precomputation cost of GWNN, one should investigate the accuracy trade-off of this approximation.

Limitations of the multiscale approach The Multiscale Graph Wavelet Neural Network comes with a higher time and memory complexity, which linearly increases with the number of considered scales. It has also more trainable weights and therefore a higher risk of overfitting. One could ask ourselves if the increased complexity is worth the accuracy improvements on the large datasets. Limiting the model to two scales might allow to get a both low-pass and band-pass filtering effect while reducing the complexity and the risk of overfitting.

One should also mention the Butterworth approach. Despite being theoretically promising, the model is very unstable (i.e. its accuracy has a large standard deviation) and performs worse than the approach with exponentially decreasing scaling functions. Further work could investigate how to improve the Butterworth approach by fine-tuning the scale choices and the model hyper-parameters.

Cost-benefit tradeoff In comparison with AutoSVM, the convolutional approaches increase the accuracy on most datasets and in particular on the citation networks and on Wikipedia dataset. This however comes with a larger complexity. For practical use, one could ask ourselves if the better accuracy of the convolutional models is worth the higher training time.

	Time complexity	**Number of weights**
GCN	$\mathcal{O}(LN_{\mathcal{V}}N_f^2)$	LN_f^2
GWNN	$\mathcal{O}(LN_{\mathcal{V}}N_f(N_f+\gamma))$	$L(N_{\mathcal{V}}+N_f^2)$
DCNN	$\mathcal{O}(N_H N_{\mathcal{V}} N_f^2)$	$(N_H+1)(N_f+N_f^2)$

Table 5.8: Time and memory complexities of the different models sparse implementations for a dataset with $N_{\mathcal{V}}$ nodes, N_f features and a Wavelet transform sparsity coefficient bounded by γ. L and N_H are respectively the number of layers (for GCN and GWNN) and the number of hops (for DCNN).

Chapter 6

Toolbox

We provide an open-source Tensorflow [57] toolbox for semi-supervised classification on graph-structured data. The toolbox includes four CNN-based models (GCN, GWNN, M-GWNN and DCNN) as well as three comparison models (SVM, AutoSVM and MLP). The goals of the toolbox are:

- Provide an automated script to fit the hyperparameters of a model and to evaluate its accuracy on a large number of datasets. The results of the experiments are automatically saved (accuracy on each run and set of best hyper-parameters).
- Allow to easily implement new models by providing an adapted Keras [13] model structure for the task of semi-supervised classification on graph-structured data.
- Quickly compare models. The implementations of the convolutional models are sparse and parallelizable, which allows to run the experiments on very large graphs. A summary of the time and memory complexities of the models' implementations is provided in Table 5.8
- Visualize the datasets. We provide a visualization of the graphs. These illustrations aim to help better understand the links between nodes and visualize the inter and intra class connections. Illustrations for the datasets used in this work can be found in Appendix C.

The toolbox is available at
`https://github.com/pfdp0/graph-classification-TF2-toolbox`.

Chapter 7

Conclusion

Through the analysis of the performance of convolution-based Graph Neural Networks on the task of semi-supervised classification on graph-structured datasets, we highlighted possible reasons for the good or bad performance of these Neural Networks and suggested that models with a band-pass frequency response in addition to low-pass frequency response might lead to an improved accuracy. From these observations, we suggested how to improve the Graph Wavelet Neural Network by working at different well chosen scales and with a set of scaling functions that behaves like a band-pass filter in the graph Fourier domain. Our model with four scales outperformed the GWNN by a significant margin on the challenging Wikipedia dataset. This result suggests the multiscale approach might help in the classification task on challenging datasets. Unfortunately, the approach using Butterworth filters as scaling functions showed encouraging but worse results. This provides however interesting directions to develop new state-of-the-art graph-structured classification models.

In addition, we showed that model regularization does not seem to be an important tuning aspect of Graph Neural Networks whereas graph based regularization might help. We also perform a series of experiments on loss functions and highlight that the softmax cross-entropy loss is in general a reasonable loss function choice, while the cubed hinge loss might be a better alternative for the Diffusional Neural Network. By a comparison of the fine-tuned models, we conclude that the Graph Wavelet Neural Network with graph regularization is the best choice for semi-supervised classification on graph-structured data, as it provides state-of-the-art predictions even on datasets on which the autocorrelation hypothesis is not met.

As a final contribution, we provide an open-source toolbox that allows to include any new semi-supervised classification model on graph-structured data and to easily compare and tune the models on a variety of datasets.

Bibliography

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] Constantin Aliferis and Ioannis Tsamardinos. "Machine Learning Methods for Decision Support and Discovery". In: Department of Biomedical Informatics, Vanderbilt University, 2004.

[3] Md. Zahangir Alom et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures". In: *Electronics* 8 (Mar. 2019), p. 292. DOI: 10.3390/electronics8030292.

[4] Valeria Andreieva and Nadiia Shvai. "Generalization of Cross-Entropy Loss Function for Image Classification". In: *Mohyla Mathematical Journal* 3 (Jan. 2021), pp. 3–10. DOI: 10.18523/2617-7080320203-10.

[5] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in neural information processing systems*. 2016, pp. 1993–2001.

[6] NH Augustin, Moira A Mugglestone, and Stephen T Buckland. "An autologistic model for the spatial distribution of wildlife". In: *Journal of Applied Ecology* (1996), pp. 339–347.

[7] Muhammet Balcilar et al. "Analyzing the Expressive Power of Graph Neural Networks in a Spectral Perspective". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Vienna, Austria, May 2021. URL: https://hal-normandie-univ.archives-ouvertes.fr/hal-03135633.

[8] Chaim Baskin et al. "Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform". In: (July 2017).

[9] Julian E Besag. "Nearest-neighbour systems and the auto-logistic model for binary data". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 34.1 (1972), pp. 75–83.

[10] Giovanni Buroni, Bertrand Lebichot, and Gianluca Bontempi. "AST-MTL: An Attention-based Multi-Task Learning Strategy for Traffic Forecasting". In: *IEEE Access* (2021).

[11] Stephen Butterworth et al. "On the theory of filter amplifiers". In: *Wireless Engineer* 7.6 (1930), pp. 536–541.

[12] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien, eds. *Semi-Supervised Learning*. London, England: MIT Press, 2006.

[13] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[14] Charles Chui. *An Introduction to Wavelets*. Vol. 1. Jan. 1992. DOI: 10.2307/2153134.

[15] Fan R K Chung. *Spectral Graph Theory*. Providence, RI: American Mathematical Society, 1996.

[16] William G. Cochran. "The χ^2 Test of Goodness of Fit". In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 315–345. DOI: 10.1214/aoms/1177729380. URL: https://doi.org/10.1214/aoms/1177729380.

[17] Mark Craven et al. "Learning to construct knowledge bases from the World Wide Web". In: *Artificial Intelligence* 118.1 (2000), pp. 69–113. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(00)00004-7. URL: https://www.sciencedirect.com/science/article/pii/S0004370200000047.

[18] Mark Craven et al. "Learning to Extract Symbolic Knowledge from the World Wide Web". In: AAAI '98/IAAI '98. Madison, Wisconsin, USA: American Association for Artificial Intelligence, 1998, pp. 509–516. ISBN: 0262510987.

[19] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

[20] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016.

[21] Janez Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets". In: *Journal of Machine Learning Research* 7.1 (2006), pp. 1–30. URL: http://jmlr.org/papers/v7/demsar06a.html.

[22] Kai-Bo Duan and S Sathiya Keerthi. "Which is the best multiclass SVM method? An empirical study". In: *International workshop on multiple classifier systems*. Springer. 2005, pp. 278–285.

[23] A. Duncan. "Powers of the adjacency matrix and the walk matrix". In: University of Malta. Department of Mathematics, 2004.

[24] Craig K. Enders. "Maximum Likelihood Estimation". In: *Encyclopedia of Statistics in Behavioral Science*. American Cancer Society, 2005. ISBN: 978-0470013199. DOI: https://doi.org/10.1002/0470013192.bsa200. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470013192.bsa200. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470013192.bsa200.

[25] Paul Erdos and Samuel James Taylor. "Some intersection properties of random walk paths". In: *Acta Math. Acad. Sci. Hungar* 11 (1960), pp. 231–248.

[26] D.B. Fogel et al. "A self-learning evolutionary chess program". In: *Proceedings of the IEEE* 92.12 (2004), pp. 1947–1954. DOI: 10.1109/JPROC.2004.837633.

[27] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and Models for Network Data and Link Analysis*. Cambridge University Press, 2016. DOI: 10.1017/CBO9781316418321.

[28] Milton Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance". In: *Journal of the American Statistical Association* 32.200 (1937), pp. 675–701. DOI: 10.1080/01621459.1937.10503522. URL: %5Curl%7B%20https://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522%7D.

[29] William A Gardner. "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique". In: *Signal processing* 6.2 (1984), pp. 113–133.

[30] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computation* 4 (Jan. 1992), pp. 1–58. DOI: 10.1162/neco.1992.4.1.1.

[31] Dave Gershgorn. *The data that transformed AI research-and possibly the world*. URL: https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/.

[32] Lise Getoor. "Link-based classification". In: *Advanced methods for knowledge discovery from complex data*. Springer, 2005, pp. 189–207.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[34] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. en. 2nd ed. Chapman and Hall/CRC, 2005.

[35] Mohamed Farouk Abdel Hady and Friedhelm Schwenker. "Semi-supervised learning". In: *Handbook on Neural Information Processing*. Springer, 2013, pp. 215–239.

[36] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. "Wavelets on graphs via spectral graph theory". In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[37] Jun Han and Claudio Moraga. "The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning." In: *IWANN*. Ed. by José Mira and Francisco Sandoval Hernández. Vol. 930. Lecture Notes in Computer Science. Springer, 1995, pp. 195–201. ISBN: 3-540-59497-3. URL: http://dblp.uni-trier.de/db/conf/iwann/iwann1995.html#HanM95.

[38] Hassan Hassan et al. "Assessment of Artificial Neural Network for bathymetry estimation using High Resolution Satellite imagery in Shallow Lakes: Case Study El Burullus Lake". In: *International Water Technology Journal* 5 (Dec. 2015).

[39] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. 2nd ed. Springer series in statistics. New York, NY: Springer, 2009, pp. 397–400. ISBN: 9780387848570 9780387848587.

[40] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[41] Feng-hsiung Hsu. "IBM's deep blue chess grandmaster chips". In: *IEEE Micro* 19.2 (1999), pp. 70–81.

[42] Ronald Iman and James Davenport. "Approximations of the critical region of the Friedman statistic". In: *Communications in Statistics-Theory and Methods* 9 (Jan. 1980), pp. 571–595.

[43] Katarzyna Janocha and Wojciech Marian Czarnecki. "On loss functions for deep neural networks in classification". In: *Schedae Inform.* 1/2016 (2017).

[44] Dan Kalman. "Leveling with Lagrange: An Alternate View of Constrained Optimization". In: *Mathematics Magazine* 82 (June 2009), pp. 186–196. DOI: 10.4169/193009809X468788.

[45] Le Kang et al. "Convolutional neural networks for document image classification". In: *2014 22nd International Conference on Pattern Recognition*. IEEE. 2014, pp. 3168–3172.

[46] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[47] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017.

[48] Stephen Kobourov. "Force-Directed Algorithms". In: Jan. 2013, pp. 383–408.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[50] Anders Krogh and John A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS'91. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, pp. 950–957. ISBN: 1558602224.

[51] Bertrand Lebichot and Marco Saerens. "An experimental study of graph-based semi-supervised classification with additional node information". In: *Knowledge and Information Systems* 62 (Nov. 2020). DOI: 10.1007/s10115-020-01500-0.

[52] Yann LeCun et al. "A theoretical framework for back-propagation". In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988, pp. 21–28.

[53] Jure Leskovec and Julian Mcauley. "Learning to Discover Social Circles in Ego Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/7a614fd06c325499f1680b9896beedeb-Paper.pdf.

[54] Tsung-Yi Lin et al. “Focal loss for dense object detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017.

[55] Christopher M. *Pattern Recognition and Machine Learning*. en. New York, NY: Springer, 2016.

[56] T Anthony Marsland and Jonathan Schaeffer. *Computers, chess, and cognition*. Springer, 1990.

[57] Martı´n Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[58] Julian McAuley et al. “Image-based recommendations on styles and substitutes”. In: *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 2015, pp. 43–52.

[59] Clare McGillem. *Continuous and discrete signal and system analysis*. New York: Holt, Rinehart, and Winston, 1984, p. 118. ISBN: 0-03-061703-0.

[60] Thomas M. Mitchell. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[61] P. A. P. Moran. “Notes on Continuous Stochastic Phenomena”. In: *Biometrika* 37.1/2 (1950), pp. 17–23. ISSN: 00063444.

[62] Ravil Muhamedyev. “Machine learning methods: An overview”. In: *Computer modelling & new technologies* 19.6 (2015), pp. 14–29.

[63] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[64] S. V. Narasimhan, Nandini Basumallick, and S. Veena. *Introduction to Wavelet Transform: A Signal Processing Approach*. 1st. Alpha Science International, Ltd, 2011. ISBN: 1842656295.

[65] P. Nemenyi. *Distribution-free Multiple Comparisons*. Princeton University, 1963. URL: `https://books.google.com.br/books?id=nhDMtgAACAAJ`.

[66] M. E. J. Newman. *Networks: an introduction*. Oxford; New York: Oxford University Press, 2010. ISBN: 9780199206650 0199206651.

[67] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Jan. 1998, p. 1. ISBN: 978-0-19-510270-3.

[68] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[69] Norman Ricker. “Wavelet Contraction, Wavelet Expansion, and the Control of Seismic Resolution”. In: *Geophysics* 18.4 (Oct. 1953), p. 769. DOI: `10.1190/1.1437927`.

[70] Ryan A. Rossi and Nesreen K. Ahmed. “The Network Data Repository with Interactive Graph Analytics and Visualization”. In: *AAAI*. 2015.

[71] Rumelhart et al. “Learning Internal Representations by Error Propagation”. In: *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations*. Jan. 1986. Chap. 8.

[72] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.

[73] Aliaksei Sandryhaila and José M. F. Moura. "Discrete Signal Processing on Graphs". In: *IEEE Transactions on Signal Processing* 61.7 (2013), pp. 1644–1656. DOI: 10.1109/TSP.2013.2238935.

[74] Franco Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[75] Prithviraj Sen et al. "Collective Classification in Network Data". In: *AI Magazine* 29.3 (Sept. 2008), p. 93. DOI: 10.1609/aimag.v29i3.2157. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/2157.

[76] Oleksandr Shchur et al. *Pitfalls of Graph Neural Network Evaluation*. Nov. 2019.

[77] Jianbo Shi and J. Malik. "Normalized cuts and image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (2000), pp. 888–905. DOI: 10.1109/34.868688.

[78] David Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[79] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. USA: California Technical Publishing, 1997. ISBN: 0966017633.

[80] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[81] Samuel D Stearns and Ruth A David. "Signal processing algorithms". In: *Englewood Cliffs* (1988).

[82] Saha Sumit. *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. Dec. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[83] Nicolas Tremblay and Pierre Borgnat. "Graph Wavelets for Multiscale Community Mining". In: *IEEE Transactions on Signal Processing* 62.20 (2014), pp. 5227–5239. DOI: 10.1109/TSP.2014.2345355.

[84] Douglas B. West. *Introduction to Graph Theory*. 2nd ed. Prentice Hall, Sept. 2000. ISBN: 0130144002.

[85] Wikipedia. *Butterworth filter — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Butterworth%20filter&oldid=1024621095. [Online; accessed 04-June-2021]. 2021.

[86] Wikipedia. *Stochastic gradient descent — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Stochastic%20gradient%20descent&oldid=1026371783. [Online; accessed 05-June-2021]. 2021.

[87] Bingbing Xu et al. “Graph Wavelet Neural Network”. In: (Apr. 2019).

[88] Dengyong Zhou et al. “Learning with Local and Global Consistency”. In: *Advances in Neural Information Processing Systems 16* 16 (Mar. 2004), pp. 237–244.

Appendices

Appendix A

Optimal Hyperparameters

The optimal hyperparameters for the parameters fitting of Section 5.3 and Section 5.6 are the following.

A.1 SVM

	Kernel
Cornell	rbf
Texas	rbf
Washington	rbf
Wisconsin	sigmoid
facebook 1	rbf
facebook 2	rbf
facebook 3	rbf
CiteSeer	rbf
Cora	rbf
Wikipedia	rbf
Amazon 1	rbf
Amazon 2	rbf

Table A.1: Best hyperparameters for SVM. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.2 AutoSVM

	Kernel
Cornell	rbf
Texas	rbf
Washington	rbf
Wisconsin	rbf
Facebook 1	rbf
Facebook 2	rbf
Facebook 3	rbf
CiteSeer	rbf
Cora	rbf
Wikipedia	rbf
Amazon 1	rbf
Amazon 2	rbf

Table A.2: Best hyperparameters for AutoSVM. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.3 MLP

	Activation	Learning Rate	Hidden
Cornell	Sigmoid	0.01	16
Texas	ReLU	0.01	16
Washington	tanh	0.01	16
Wisconsin	ReLU	0.01	32
Facebook 1	tanh	0.001	16
Facebook 2	Sigmoid	0.01	16
Facebook 3	tanh	0.001	16
CiteSeer	tanh	0.001	64
Cora	Sigmoid	0.001	32
Wikipedia	Sigmoid	0.001	64
Amazon 1	Sigmoid	0.001	32
Amazon 2	Sigmoid	0.001	32

Table A.3: Best hyperparameters for MLP. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.4 GCN

	Activation	Learning Rate	Hidden
Cornell	ReLU	0.01	32
Texas	ReLU	0.01	64
Washington	ReLU	0.01	16
Wisconsin	ReLU	0.01	64
Facebook 1	ReLU	0.001	16
Facebook 2	ReLU	0.01	16
Facebook 3	ReLU	0.01	64
CiteSeer	ReLU	0.001	64
Cora	ReLU	0.01	16
Wikipedia	ReLU	0.01	64
Amazon 1	ReLU	0.001	64
Amazon 2	ReLU	0.01	64

Table A.4: Best hyperparameters for GCN. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.5 GWNN

	Activation	Learning Rate	Hidden	Wavelet Scale
Cornell	ReLU	0.01	32	0.00390625
Texas	ReLU	0.01	64	0.015625
Washington	ReLU	0.01	32	0.015625
Wisconsin	ReLU	0.01	64	0.015625
Facebook 1	ReLU	0.001	64	0.0625
Facebook 2	ReLU	0.01	16	0.25
Facebook 3	ReLU	0.01	16	0.00390625
CiteSeer	ReLU	0.001	32	0.25
Cora	ReLU	0.001	64	0.5
Wikipedia	ReLU	0.001	64	0.015625
Amazon 1	ReLU	0.01	32	0.0625
Amazon 2	ReLU	0.01	64	0.0625

Table A.5: Best hyperparameters for GWNN. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.6 DCNN

	Activation	Learning Rate	Hops
Cornell	tanh	0.01	2
Texas	tanh	0.01	3
Washington	tanh	0.01	3
Wisconsin	tanh	0.01	3
Facebook 1	tanh	0.01	1
Facebook 2	tanh	0.01	3
Facebook 3	tanh	0.01	2
CiteSeer	tanh	0.01	2
Cora	tanh	0.01	3
Wikipedia	tanh	0.001	3
Amazon 1	tanh	0.001	3
Amazon 2	tanh	0.001	3

Table A.6: Best hyperparameters for DCNN. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs.

A.7 Fine-tuned DCNN

	Activation	LR	Hops	Loss Function	Regularization
Cornell	tanh	0.01	2	$(\text{hinge})^3$	Graph $I - P$
Texas	tanh	0.01	3	$(\text{hinge})^3$	Graph $I - P$
Washington	tanh	0.01	3	$(\text{hinge})^3$	Graph $I - P$
Wisconsin	tanh	0.01	3	$(\text{hinge})^3$	Graph $I - P$
Facebook 1	tanh	0.01	1	softmax C.E.	Graph $I - P$
Facebook 2	tanh	0.01	3	softmax C.E.	Graph $I - P$
Facebook 3	tanh	0.01	2	softmax C.E.	Graph $I - P$
CiteSeer	tanh	0.01	2	softmax C.E.	Graph $I - P$
Cora	tanh	0.01	3	softmax C.E.	Graph $I - P$
Wikipedia	tanh	0.001	3	$(\text{hinge})^3$	Graph $I - P$
Amazon 1	tanh	0.001	3	softmax C.E.	Graph $I - P$
Amazon 2	tanh	0.001	3	softmax C.E.	Graph $I - P$

Table A.7: Best hyperparameters, regularization and losses for DCNN. The best hyperparameters are the combination of hyperparameters that gave the most times the best accuracy on ten runs. "LR" and "C.E." refer respectively to Learning Rate Cross Entropy.

Appendix B

Full results for the models improvements

B.1 MLP

MLP	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	83.89	88.33	86.04	88.42	92.17	91.92	79.63	71.82	72.67	60.37	89.60	82.16
Dropout	81.80	87.71	85.08	88.42	92.12	92.38	78.66	72.23	72.29	61.17	88.63	81.42
Weights	82.68	88.20	85.44	88.22	91.88	91.96	79.64	71.86	72.56	60.59	89.67	82.23
Regu L	82.75	87.96	85.42	88.29	91.71	92.09	79.21	72.24	72.91	60.56	89.60	81.59
Regu P	82.79	87.85	85.59	87.87	91.94	92.12	79.85	71.66	72.97	60.68	89.61	82.26

Table B.1: Mean test accuracy for the different models improvements on MLP. Average is taken over ten runs.

MLP	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	83.89	88.33	86.04	88.42	92.17	91.92	79.63	71.82	72.67	60.37	89.60	82.16
Softmax focal loss	83.14	87.54	85.35	87.55	91.59	91.81	79.99	71.25	72.12	60.36	89.57	81.97
L1 loss	78.92	74.35	82.22	83.38	91.85	90.73	79.64	69.64	68.26	50.47	80.83	59.55
Softmax L1 loss	77.60	84.59	83.42	86.58	92.05	92.21	79.15	71.64	63.83	55.02	81.36	60.17
L2 loss	80.44	83.21	81.72	84.69	92.14	91.29	79.39	71.83	72.54	57.88	88.35	81.13
Softmax L2 loss	82.48	87.65	85.38	87.52	91.51	92.10	79.44	71.78	72.46	60.20	89.58	82.26
Hinge loss	79.78	88.00	85.03	88.06	91.64	92.33	79.88	72.34	73.19	59.19	89.00	82.05
$(\text{Hinge})^2$ loss	81.76	87.25	85.38	88.35	91.46	92.06	79.90	72.33	72.98	59.34	89.09	82.25
$(\text{Hinge})^3$ loss	81.71	86.46	85.42	87.82	91.46	92.01	79.54	71.76	72.79	58.63	88.83	81.76

Table B.2: Mean test accuracy for the different loss functions on MLP. Average is taken over ten runs.

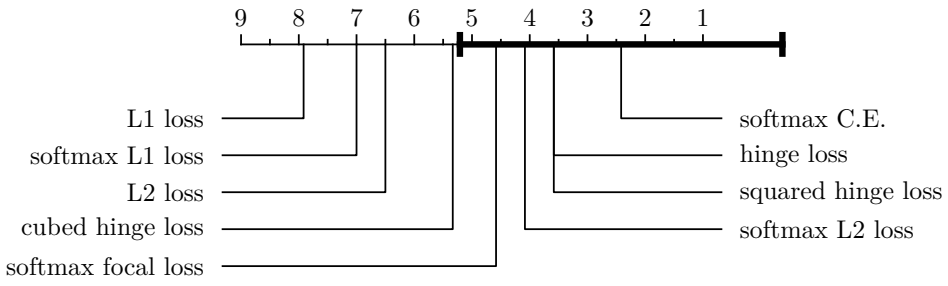


Figure B.1: Results of Bonferroni-Dunn's test on MLP with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. MLP with Softmax Cross-Entropy loss (C.E.)).

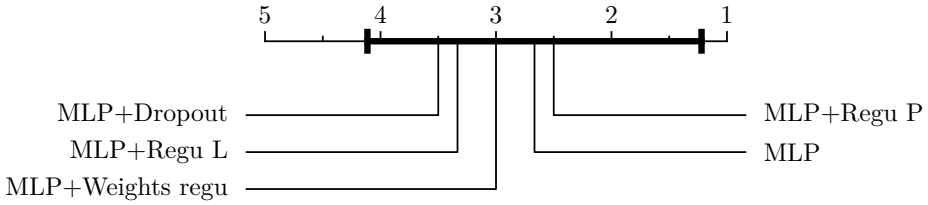


Figure B.2: Results of Bonferroni-Dunn's test on MLP with the different model improvements for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. MLP without any improvements.

B.2 GCN

GCN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	65.80	64.26	65.21	59.73	96.83	98.30	84.08	77.26	86.06	48.71	93.62	89.00
Dropout	66.04	65.14	64.73	58.16	96.26	98.30	84.23	77.30	86.48	49.02	93.62	89.44
Weights	65.93	64.04	64.86	59.98	96.79	98.22	84.34	77.21	85.96	48.94	93.64	89.10
Regu L	65.47	64.59	64.64	58.61	96.09	98.27	84.64	77.33	86.01	48.75	93.74	90.03
Regu P	65.56	63.91	64.37	60.18	96.30	98.40	84.39	77.30	85.98	48.80	93.82	90.12

Table B.3: Mean test accuracy for the different models improvements on GCN. Average is taken over ten runs.

GCN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	65.80	64.26	65.21	59.73	96.83	98.30	84.08	77.26	86.06	48.71	93.62	89.00
Softmax focal loss	65.23	61.82	64.71	59.14	96.83	98.24	84.58	77.34	85.48	45.76	93.55	57.54
L1 loss	57.69	61.27	59.87	57.04	96.86	98.30	82.99	77.52	51.23	17.16	91.62	85.45
Softmax L1 loss	55.67	63.60	62.50	55.31	96.84	98.36	83.06	77.37	85.19	31.96	79.44	59.98
L2 loss	61.76	61.52	63.16	59.33	96.83	98.24	84.17	77.61	85.83	35.10	93.93	90.03
Softmax L2 loss	63.01	65.65	63.91	59.33	96.81	98.29	84.16	77.10	86.03	48.01	93.20	89.12
Hinge loss	62.09	65.63	63.33	59.73	96.35	98.29	83.53	77.92	86.20	48.13	93.86	90.14
$(\text{Hinge})^2$ loss	63.89	62.66	64.80	59.02	96.93	98.24	84.30	77.63	86.04	46.11	93.82	89.64
$(\text{Hinge})^3$ loss	63.38	61.30	64.02	57.95	96.80	98.16	83.79	77.47	86.17	41.53	93.48	88.10

Table B.4: Mean test accuracy for the different losses on GCN. Average is taken over ten runs.

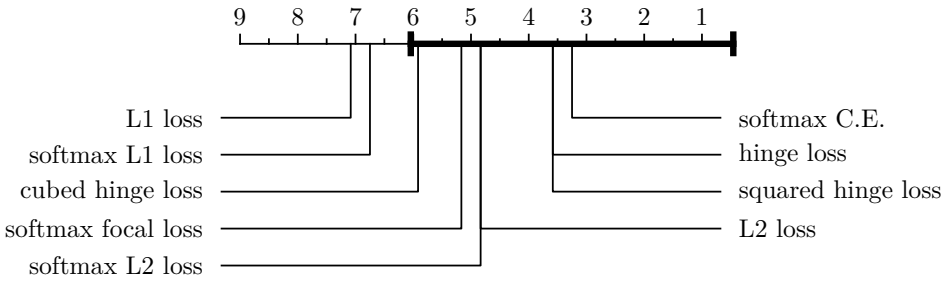


Figure B.3: Results of Bonferroni-Dunn's test on GCN with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. GCN with Softmax Cross-Entropy loss (C.E.)).

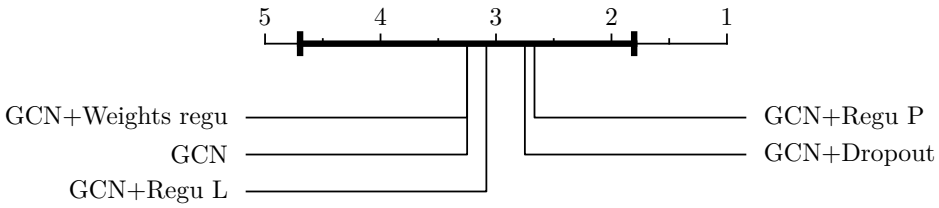


Figure B.4: Results of Bonferroni-Dunn's test on GCN with the different model improvements for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. GCN without any improvements.

B.3 GWNN

GWNN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	82.77	88.88	84.80	87.10	96.92	98.38	83.64	77.78	87.21	60.44	93.89	89.72
Dropout	76.31	81.14	75.57	84.77	96.43	98.29	82.34	76.47	86.63	47.65	93.44	87.89
Weights	83.25	88.35	84.02	87.66	97.00	98.33	83.29	77.87	87.57	60.75	93.80	89.70
Regu L	82.51	88.33	83.46	87.42	97.00	98.38	83.89	77.98	87.70	59.80	93.71	88.74
Regu P	82.99	88.84	83.70	87.30	97.07	98.33	83.75	77.99	87.74	60.44	94.03	90.34

Table B.5: Mean test accuracy for the different models improvements on GWNN. Average is taken over ten runs.

GWNN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	82.77	88.88	84.80	87.10	96.92	98.38	83.64	77.78	87.21	60.44	93.89	89.72
Softmax focal loss	81.98	88.13	83.83	86.67	97.12	98.40	82.73	77.38	87.38	7.62	93.55	87.79
L1 loss	76.00	84.75	81.31	84.42	97.20	97.24	82.51	77.01	87.29	12.31	65.08	78.95
Softmax L1 loss	79.89	84.22	80.04	84.67	97.19	95.69	82.89	77.23	87.01	42.33	80.83	61.52
L2 loss	78.97	83.96	82.84	84.86	97.14	98.30	83.02	78.09	87.53	13.18	93.57	89.20
Softmax L2 loss	83.01	88.48	82.92	86.75	97.06	98.42	83.40	77.76	87.62	58.88	93.77	89.57
Hinge loss	81.30	88.57	82.11	86.72	96.97	97.83	83.31	78.27	87.66	59.97	94.15	90.71
$(\text{Hinge})^2$ loss	81.87	88.59	82.56	87.30	97.06	97.85	83.45	78.22	87.80	57.60	94.21	90.78
$(\text{Hinge})^3$ loss	81.41	88.18	83.14	86.99	97.02	97.80	83.06	77.76	87.65	49.05	94.07	90.44

Table B.6: Mean test accuracy for the different losses on GWNN. Average is taken over ten runs.

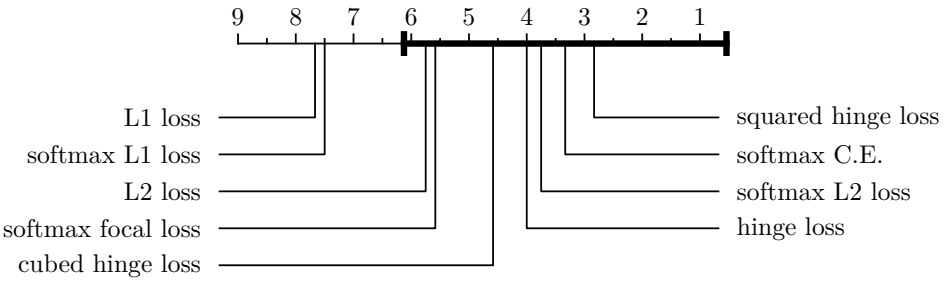


Figure B.5: Results of Bonferroni-Dunn's test on GWNN with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. GWNN with Softmax Cross-Entropy loss (C.E.)).

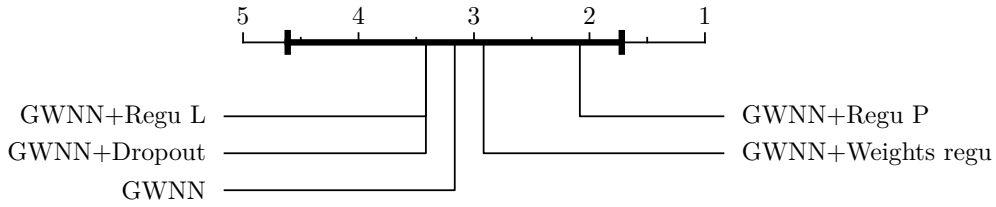


Figure B.6: Results of Bonferroni-Dunn's test on GWNN with the different model improvements for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. GWNN without any improvements.

B.4 DCNN

DCNN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	75.52	84.35	78.11	82.98	96.83	98.36	83.69	76.39	86.53	65.24	95.10	89.09
Dropout	75.41	81.30	75.72	83.50	97.01	98.39	83.57	76.86	87.01	65.02	94.86	89.24
Weights	76.53	85.71	77.85	83.30	96.98	98.31	83.62	76.52	86.59	65.54	95.12	89.10
Regu L	76.97	85.87	78.26	83.50	97.06	98.35	83.63	76.79	86.55	65.34	94.56	88.65
Regu P	77.67	85.19	79.01	84.56	97.05	98.44	83.64	76.59	86.59	65.78	95.12	89.21

Table B.7: Mean test accuracy for the different models improvements on DCNN. Average is taken over ten runs.

DCNN	Corn	Tex	Was	Wis	FB 1	FB 2	FB 3	Cite	Cora	Wiki	AM 1	AM 2
Baseline	75.52	84.35	78.11	82.98	96.83	98.36	83.69	76.39	86.53	65.24	95.10	89.09
Softmax focal loss	78.90	87.41	78.45	84.64	97.04	98.44	83.63	76.05	85.99	64.90	95.08	88.88
L1 loss	74.77	75.80	74.22	75.78	97.32	98.40	82.22	75.52	82.41	51.31	90.09	78.04
Softmax L1 loss	50.22	63.89	64.00	63.42	97.02	98.49	82.59	76.83	86.34	51.10	79.94	57.21
L2 loss	79.98	86.15	79.51	84.98	97.15	98.31	83.58	76.49	85.44	66.06	93.90	87.80
Softmax L2 loss	77.80	84.00	76.58	82.67	97.02	98.41	83.61	76.59	86.62	65.58	95.07	88.89
Hinge loss	69.12	71.01	72.13	73.02	97.11	98.29	82.77	77.21	86.82	66.19	94.90	89.01
$(\text{Hinge})^2$ loss	74.13	86.55	79.16	80.85	97.07	98.35	83.77	76.75	86.64	66.78	94.87	88.87
$(\text{Hinge})^3$ loss	81.01	87.52	80.19	85.89	97.03	98.41	83.78	76.23	86.25	66.16	94.92	88.82

Table B.8: Mean test accuracy for the different losses on DCNN. Average is taken over ten runs.

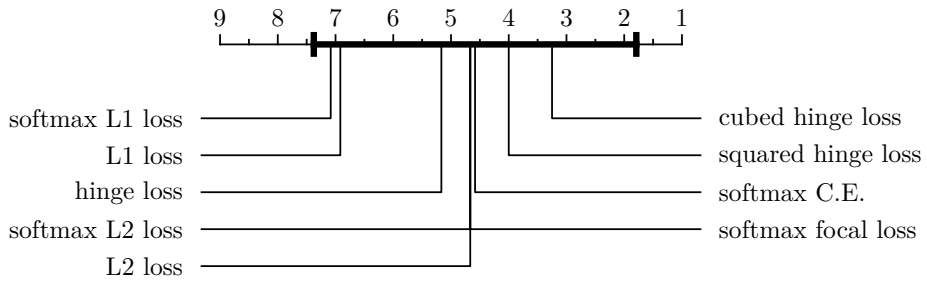


Figure B.7: Results of Bonferroni-Dunn's test on DCNN with the different loss functions for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. DCNN with Softmax Cross-Entropy loss (C.E.)).

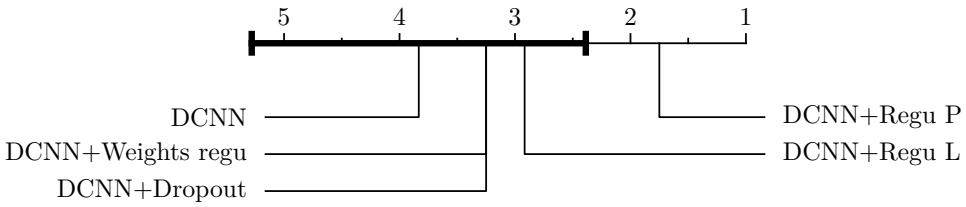


Figure B.8: Results of Bonferroni-Dunn's test on DCNN with the different model improvements for $p < 0.1$. Connected models are not statistically different to the baseline model (i.e. DCNN without any improvements.

Appendix C

Structure of the datasets

This appendix shows visualisations of the graph datasets' structures. Such illustrations aim to help better understand the links between nodes and visualize the inter and intra class connections. The nodes are positioned using Fruchterman-Reingold force-directed algorithm [48], which is an heuristic that does not take the class information into account.

C.1 Cornell

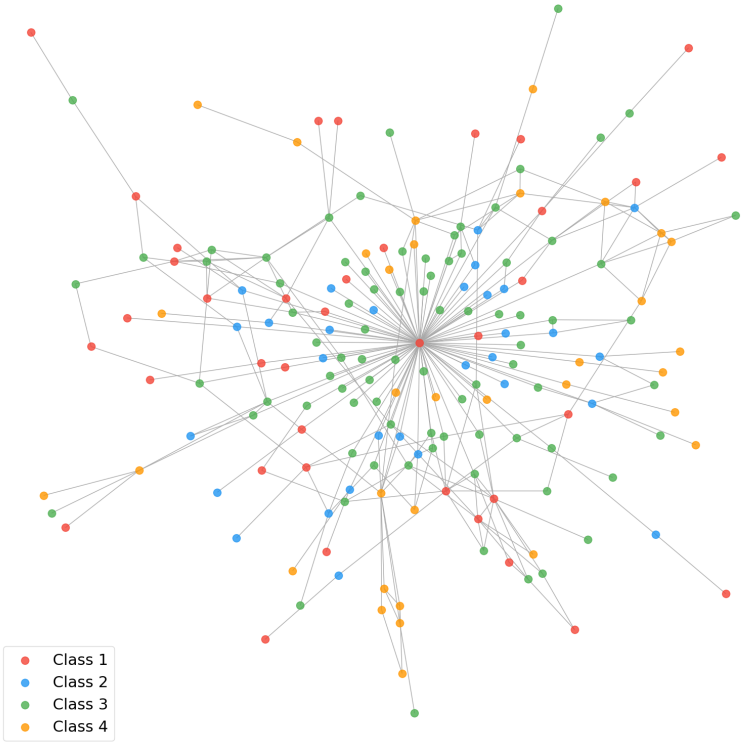


Figure C.1: Graph of Cornell dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.2 Texas

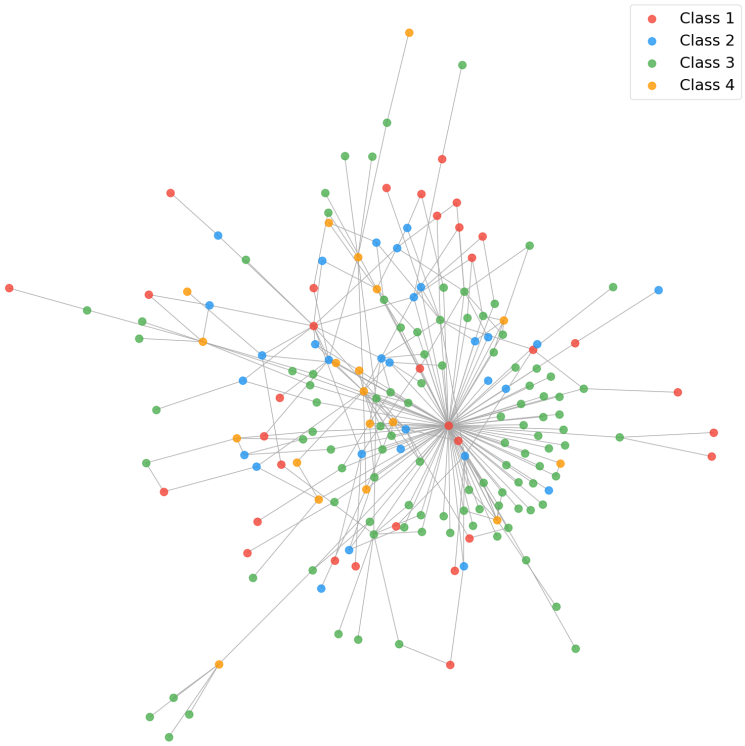


Figure C.2: Graph of Texas dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.3 Washington

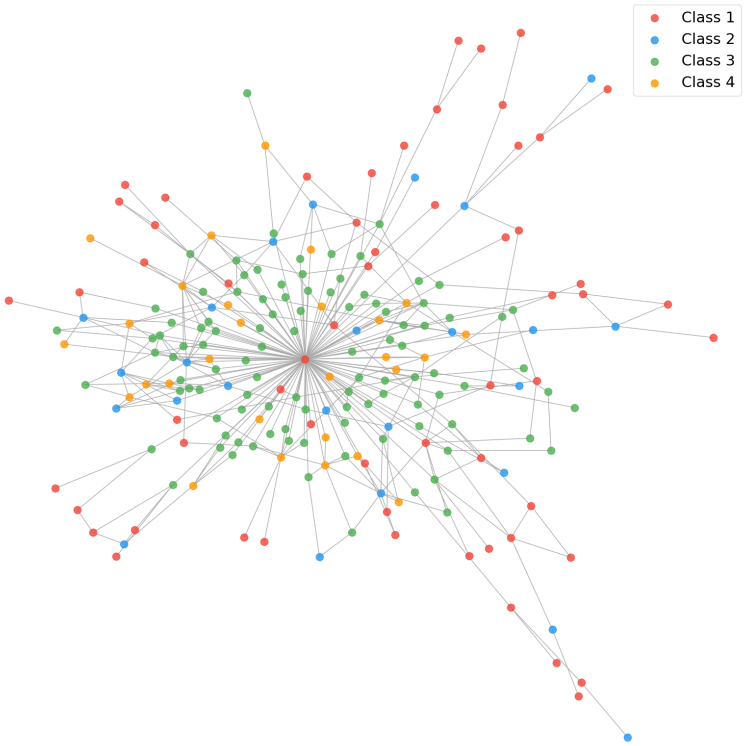


Figure C.3: Graph of Washington dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.4 Wisconsin

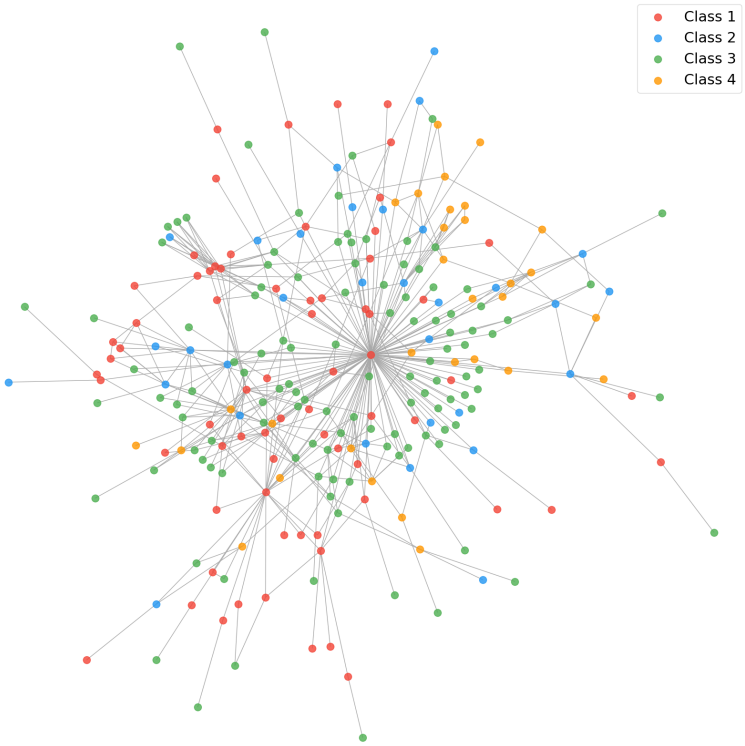


Figure C.4: Graph of Wisconsin dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.5 Facebook 1

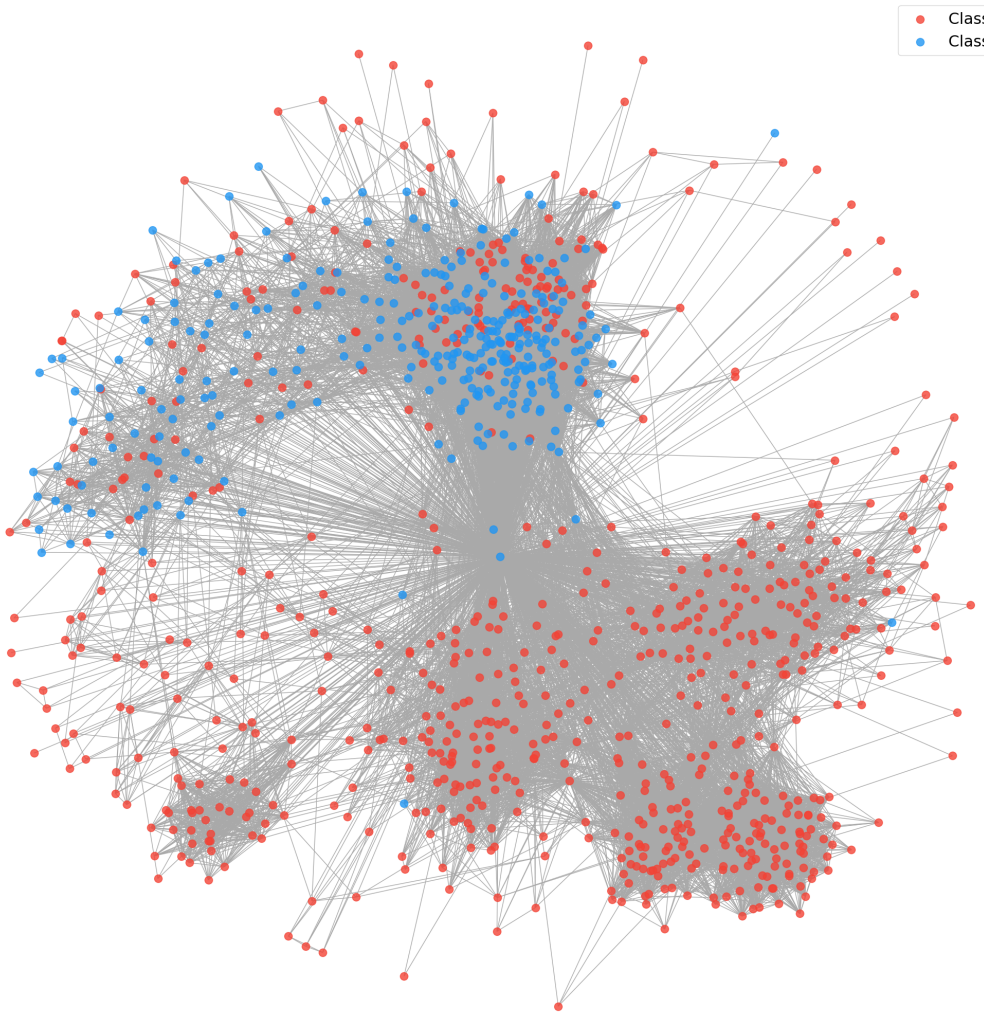


Figure C.5: Graph of Facebook 1 dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.6 Facebook 2

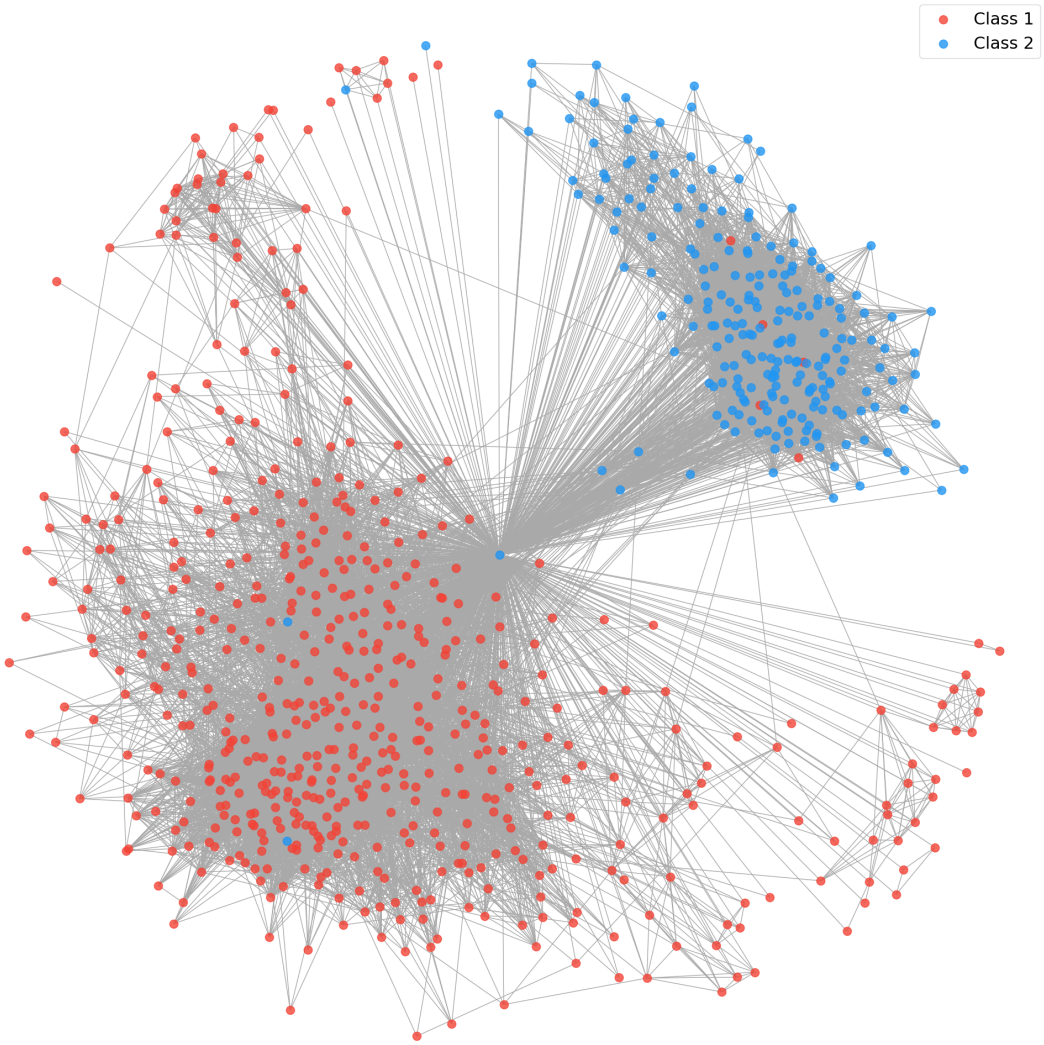


Figure C.6: Graph of Facebook 2 dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.7 Facebook 3

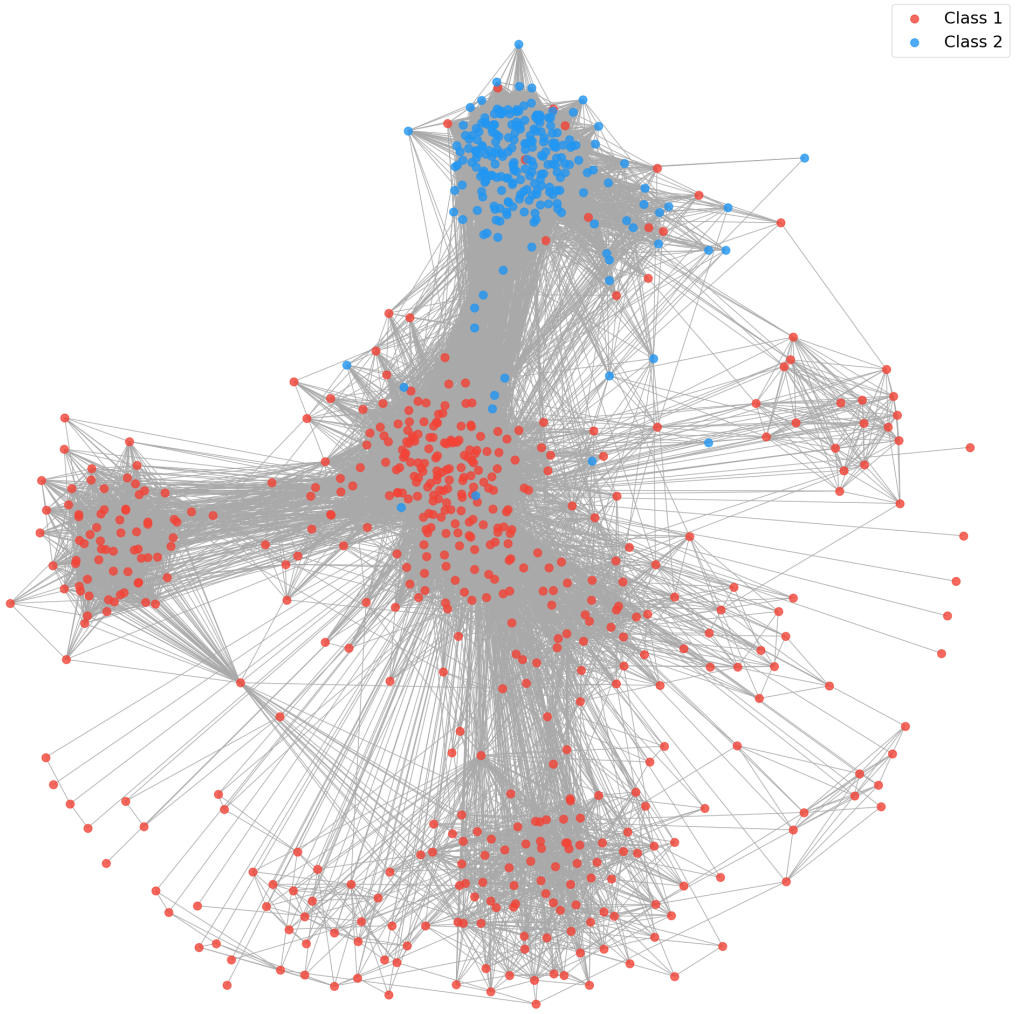


Figure C.7: Graph of Facebook 3 dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.8 CiteSeer

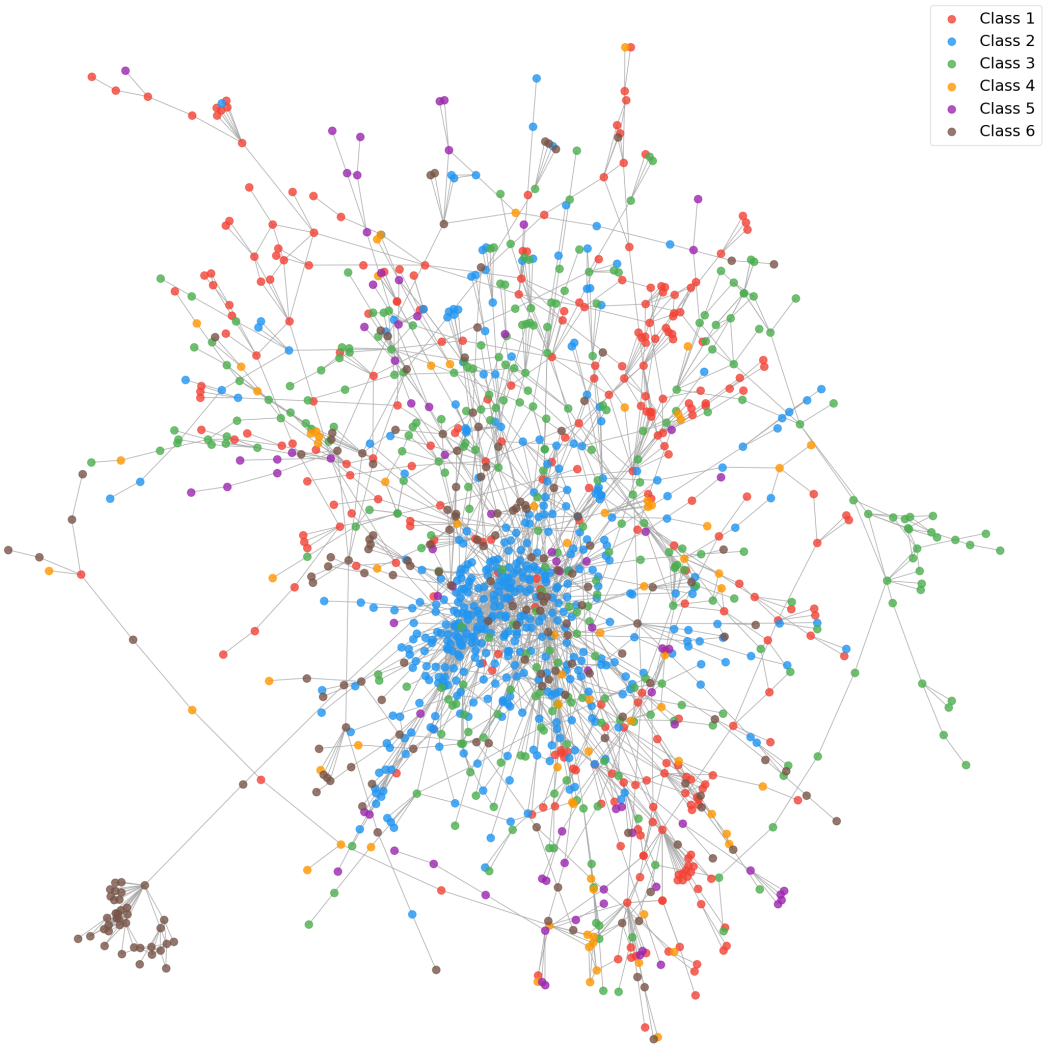


Figure C.8: Graph of CiteSeer dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.9 Cora

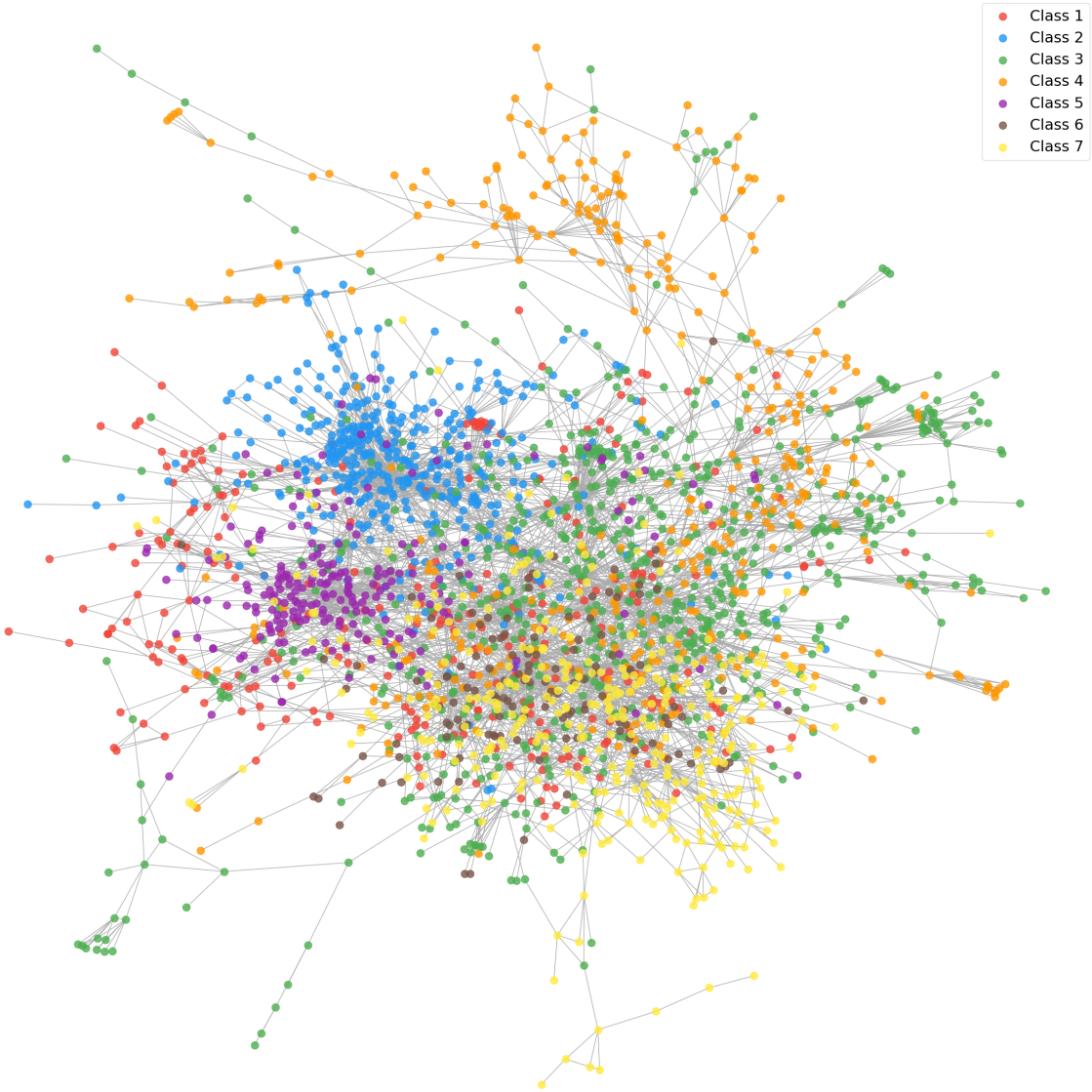


Figure C.9: Graph of Cora dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.10 Wikipedia

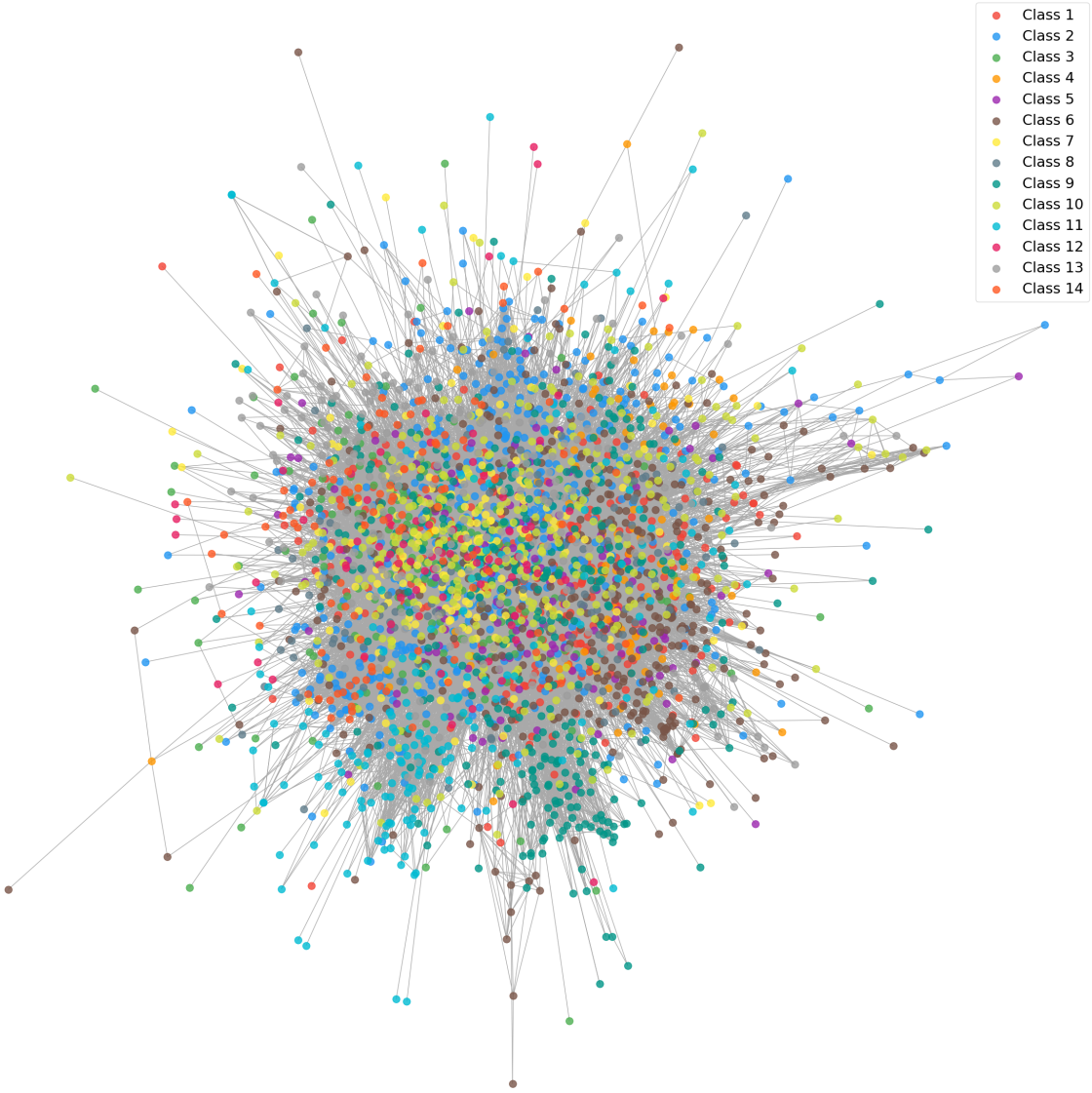


Figure C.10: Graph of Wikipedia dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.11 Amazon 1

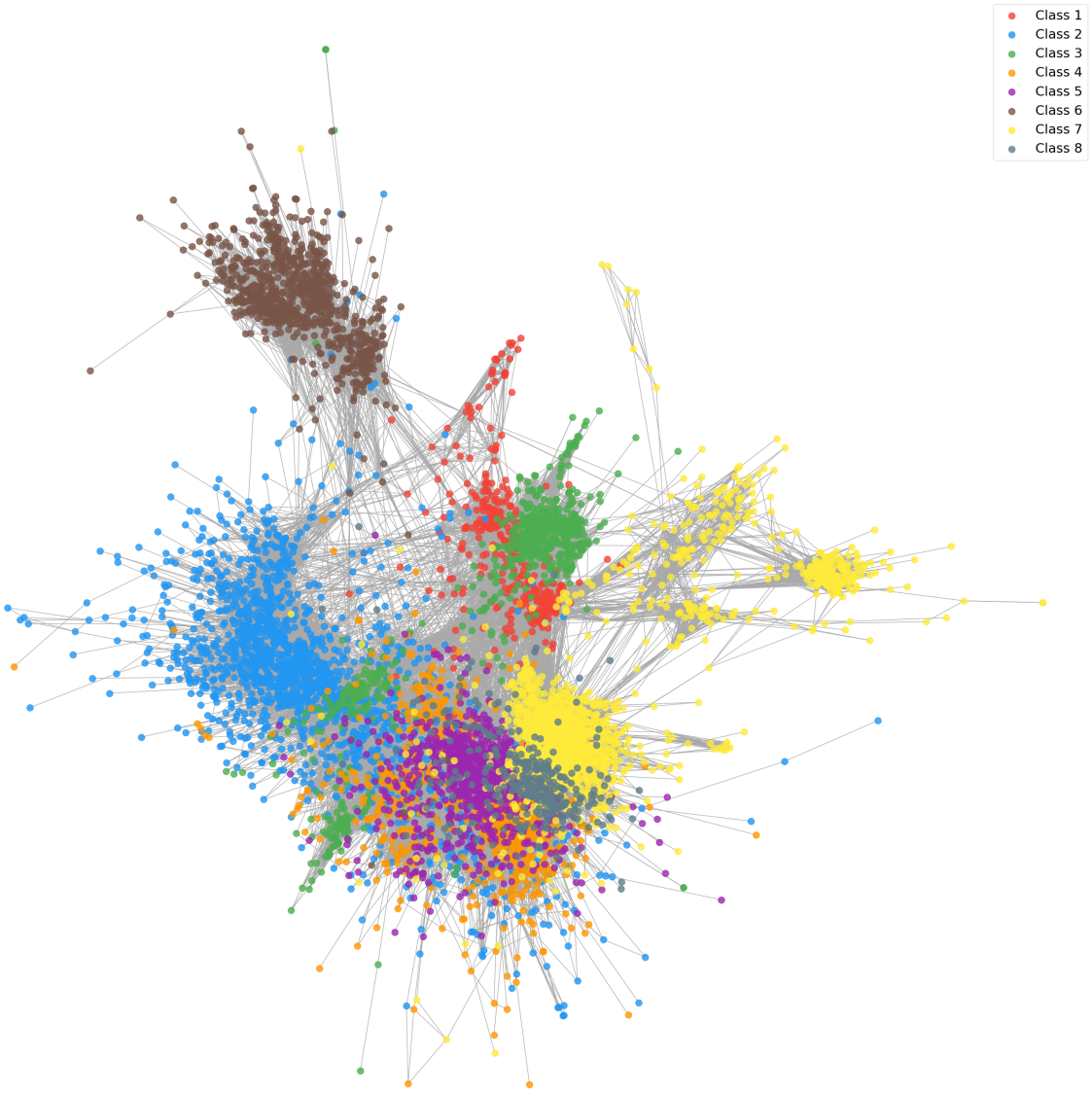


Figure C.11: Graph of Amazon 1 dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

C.12 Amazon 2

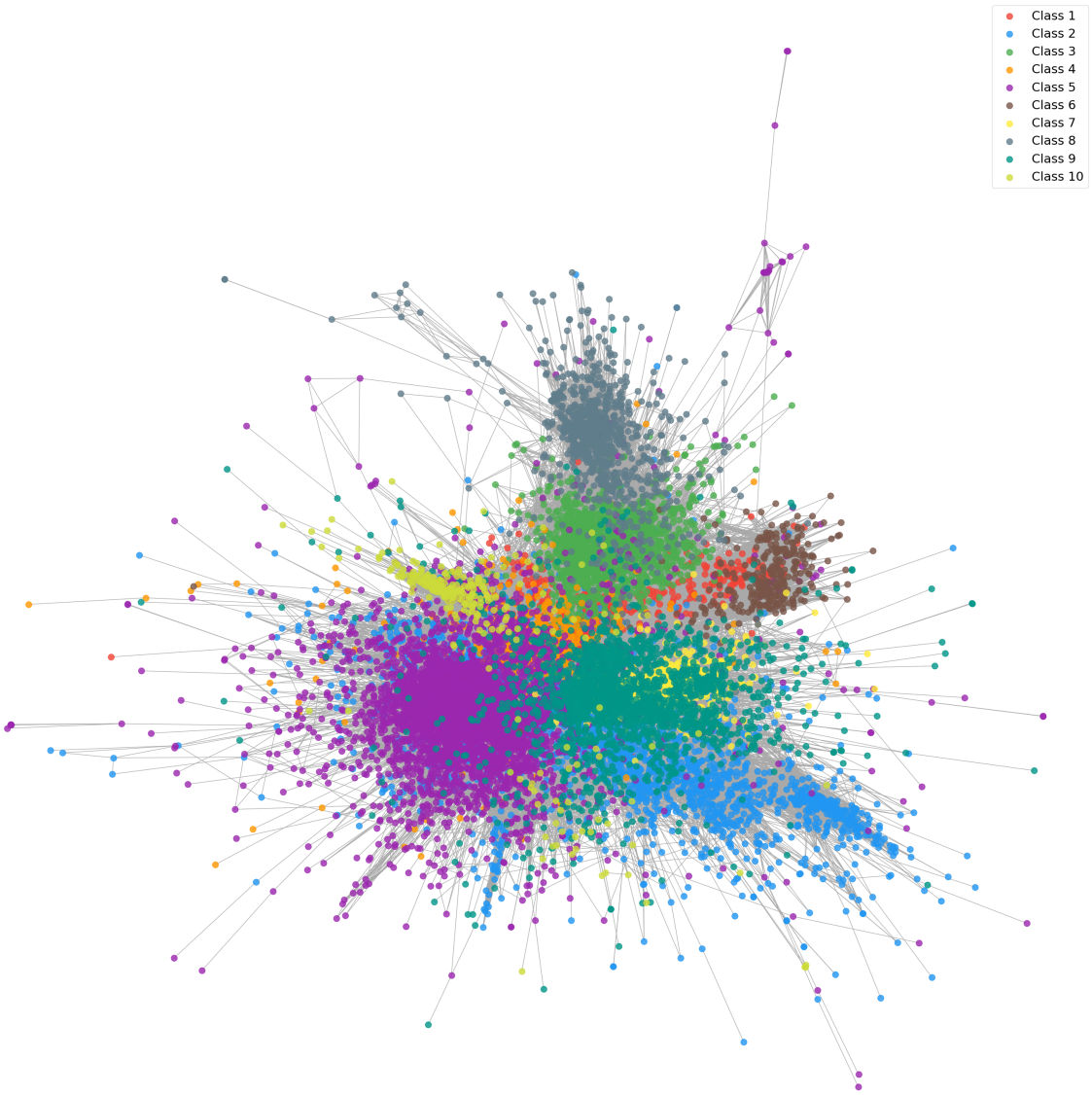


Figure C.12: Graph of Amazon 2 dataset, the nodes are positioned using Fruchterman-Reingold force-directed algorithm [48]. Nodes are depicted by circles which color represents the class membership and edges are represented by grey stripes.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**